**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | Tecnólogo en aseguramiento de la calidad de *software* |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501099 - Probar la solución del *software* de acuerdo con parámetros técnicos y modelos  de referencia | RESULTADOS DE APRENDIZAJE | 220501099-03 Ejecutar pruebas funcionales según los requerimientos del sistema.  220501099-04 Mejorar los procesos de prueba según modelo de trabajo definido. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | CF13 |
| NOMBRE DEL COMPONENTE FORMATIVO | Aplicación de pruebas de *software* |
| BREVE DESCRIPCIÓN | En este módulo el aprendiz identificará que un *software* con calidad implica la utilización de los diferentes tipos de pruebas, metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del *software,* para lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleve la productividad, tanto para la labor de desarrollo como para el control de la calidad. |
| PALABRAS CLAVE | Calidad, diseño de prueba, herramientas de prueba, niveles de  prueba, *software, testing.* |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 2 - Ciencias naturales, aplicadas y relacionadas |
| IDIOMA | Español |

1. **Tabla de Contenidos:**

**Introducción**

**1. Realización de pruebas de software**

1.1 Tipos de pruebas

1.1.1 Pruebas funcionales

1.1.2 Pruebas unitarias

1.1.3 Pruebas de integración

1.1.4 Pruebas no funcionales

1.1.5 Pruebas de rendimiento

1.1.6 Ventajas y desventajas de los tipos de pruebas

1.1.7 Documentos de caso de pruebas

1.2 *Agile testing*

1.2.1 *Test Driven Development* (TDD)

1.2.2 *Acceptance Test Driven Development* (ATDD)

1.2.3 *Behaviour Driven Development* (BDD)

1.2.4 Testing exploratorio

1.2.5 Automatización de pruebas de regresión

1.2.6 Pruebas de exploratorias, usabilidad y aceptación

1.2.7 Pruebas desempeño, carga y seguridad

1.3. Elaboración de informe de resultados

**2. Elaboración del plan de mejora**

2.1. Identificar el área de mejora

2.2. Detectar las principales causas del problema

2.3. Formular el objetivo

2.4. Seleccionar las acciones de mejora

2.5. Realizar una planeación

2.6. Seguimiento del plan de mejora

1. **Introducción**

En este componente formativo “Aplicación de pruebas de *software*” se estudiarán los diferentes elementos para la realización de pruebas, por lo cual se abordarán las siguientes áreas de aprendizaje, tipos de pruebas funcionales y no funcionales, pruebas ágiles (*Agile testing*) y elaboración de informe de resultados.

Cabe resaltar que cuando se desarrollan aplicaciones *web*, uno de los desafíos es verificar que la calidad del producto sea óptima y sin errores, de este modo las pruebas de *software* son parte fundamental del proceso de calidad y adopción de buenas prácticas. Para comenzar el recorrido por el mismo, le invitamos a ver el recurso didáctico que se muestra a continuación.

Video

DI\_CF13\_Introduccion



1. **DESARROLLO DE CONTENIDOS:** 
   * + 1. **Realización de pruebas de *software***

En este punto para realizar las pruebas de *software* se debe establecer qué tipos de pruebas serán requeridas por el producto, puesto que para probar un producto no se requiere utilizar todos los tipos de prueba que existen sino más bien los que sean necesarios para cada caso, por esta razón, es importante que el QA o líder de pruebas determine qué tipos de prueba se adaptan al proyecto en evaluación. En el plan de pruebas se deben listar los tipos de pruebas a realizar, por lo que a continuación se especifican y describen los tipos de pruebas que se pueden utilizar.

**1.1 Tipos de pruebas**

Las pruebas de *software* se pueden clasificar en dos tipos principales: pruebas funcionales y pruebas no funcionales. Teniendo en cuenta los diferentes frentes y aspectos de un producto *software*, con lo cual se requerirán diferentes tipos de pruebas, tales como: pruebas unitarias, pruebas de integración, pruebas de estrés, pruebas de rendimiento, pruebas de escalabilidad, entre otras. Cada uno de estos tipos de pruebas de *software* determinan la visibilidad de la aplicación, desde el código fuente hasta la interacción y experiencia con el usuario. Entrando en detalles, a continuación se invita a observar el siguiente video donde podrá identificar los tipos de pruebas de *software*.

Storyboard

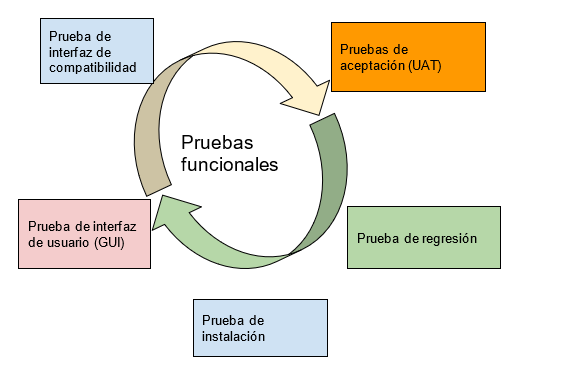
DI\_CF13\_1.1 Tipos de pruebas



Algunos de los tipos de pruebas más comunes se describen en los siguientes puntos.

* + 1. **Pruebas funcionales**

Figura 1. Tipos de pruebas funcionales

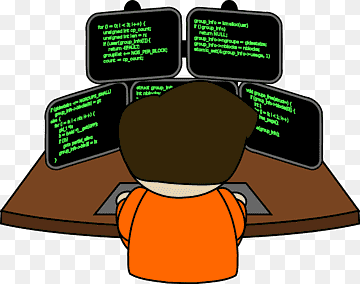


Se llevan a cabo para verificar las características esenciales para el negocio, la funcionalidad y la usabilidad. Estas garantizan que las características y funciones del *software* operan y se comportan según lo esperado sin problemas. Valida toda la aplicación respecto a las especificaciones definidas en el documento de requisitos de *software* (SRS). En estas pruebas se incluyen las pruebas unitarias, pruebas de regresión, pruebas de interfaz, entre otras.

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.

* + 1. **Pruebas unitarias**

Figura 2. Pruebas unitarias

Estas pruebas se enfocan en validar elementos como componentes/unidades/piezas de forma individual e independiente de un producto *software* dado el principio SDLC. Cualquier bloque de código sea procedimiento, función, método, constructor, clase o módulo, puede ser una unidad y deberá ser considerado y sometido a pruebas unitarias, con lo cual se podrá determinar su comportamiento y validarlo versus el resultado esperado. Estas pruebas deben ser las primeras que se lleven a cabo en la fase de desarrollo y las deberán implementar los desarrolladores.

De acuerdo a Mera (2016) también se las conoce como *unit testing* son pequeños *tests* en los cuales se revisa que el comportamiento de un objeto y su lógica funcione adecuadamente.

Por lo general se realiza durante la fase de desarrollo de las aplicaciones o *software,* esta es ejecutada por los programadores, aunque también pueden ser realizadas por los aseguradores de calidad. Las razones por las cuales se desarrolla este tipo de pruebas son:

* Son rápidas de realizar, por lo que se pueden realizar en gran cantidad.
* Sirven como documentación de apoyo para el proyecto.
* Demuestran que la lógica del código funciona en todos los casos.
* Permite a los programadores a entender bien el código base, lo que permite realizar cambios oportunos.
* 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, hay muchas herramientas en el mercado y estas varían en función del lenguaje de programación que se esté utilizando, aquí se mencionan algunas de las más conocidas.

Tarjetas Verticales

DI\_CF13\_1.1.2 Herramientas para pruebas unitarias

Para realizar las pruebas unitarias debemos tener en cuenta los siguientes criterios.

Figura 3. Criterios de pruebas unitarias

* Se debe probar un sólo código a la vez.
* Hay que realizar pruebas frecuentemente mientras se programa.
* Se debe corregir los *bugs* identificados en las pruebas antes de continuar.
* Cualquier cambio que se realice también debe pasar el *test*.

**1.1.3 Pruebas de integración**

Figura 4. Pruebas de integración

Estas pruebas se enfocan en validar diferentes módulos o componentes de un producto *software* de tal manera que se valide como un grupo funcional. Una aplicación de *software*, un sistema de información o una aplicación *web*, entre otros productos *software*  están compuestos por diferentes componentes, módulos y submódulos distribuidos en diferentes niveles y capas, que trabajan juntos y se orquestan en diferentes funcionalidades. La finalidad de estas pruebas es validar que la integración entre estos componentes sea correcta y encontrar posibles problemas o errores relacionados con la sinergia y comunicación entre ellos. 

Según Mera (2016) *“Las pruebas de integración se encargan de probar las interfaces entre los componentes o módulos; por ejemplo, el componente validación de usuario con el sistema operativo, el sistema de archivos en integración con el hardware” .*

De acuerdo a esto, se debe tener en cuenta los objetos típicos de prueba:

* Bases de datos de subsistemas
* Datos de configuración
* Configuración del sistema
* Infraestructura
* interfaces

**1.1.4 Pruebas no funcionales**

Estas pruebas son como las pruebas funcionales, con la diferencia que se prueban bajo carga de rendimiento para poder establecer la fiabilidad, usabilidad, escalabilidad, instalación, confiabilidad y seguridad. Estas pruebas generalmente se ejecutan por medio de herramientas o soluciones automatizadas.

*Según Mera (2016) en este tipo de pruebas se comprueban los requisitos basados en la operación de un software. Para verificar la carga que resiste el producto software, identificando si su rendimiento es el adecuado o si es estable a nivel de contacto con el servidor.*

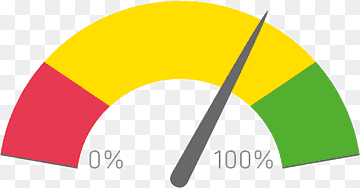
Hay algunos aspectos que pueden ser complejos, como el rendimiento de una aplicación, sin embargo, esta prueba permite comprobar la calidad del *software* que se va a probar. La calidad depende principalmente del tiempo, la precisión, la estabilidad, la corrección y la durabilidad de un producto en diversas circunstancias adversas. Con base en estos aspectos de calidad podemos estudiar diferentes tipos de pruebas no funcionales:

Imagen Interactiva

DI\_CF13\_1.1.4 Pruebas no funcionales

**1.1.5 Pruebas de rendimiento**

Figura 5. Pruebas de rendimiento



Son un tipo de pruebas no funcionales, las cuales se realizan con la finalidad de determinar la estabilidad, escalabilidad y velocidad de una funcionalidad en el producto *software*. El objetivo de este tipo de pruebas es verificar el comportamiento de la aplicación teniendo en cuenta diferentes puntos de referencia del sistema y de la red, tales como la CPU, la velocidad de renderizado o carga de la página, el control de tráfico, la concurrencia, consumo de recursos de servidor, entre otros. En esta categoría podemos encontrar las pruebas de carga y de esfuerzo.

*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.*

El mismo autor expresa que 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 que parte está ese problema para poder solucionarlo.

**Verificar el cumplimiento de los 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 botellas si se debe a problemas de hardware como CPU, memoria, o es el ancho de banda.



**1.1.6 Ventajas y desventajas de los tipos de pruebas**

Aplicar los tipos de pruebas puede asegurar que el *software* funcione como se espera en todo momento y garantizar su calidad, por lo tanto, es necesario para su aplicación, identificar sus ventajas y desventajas para determinar las diferencias entre cada tipo; se puede observar en el siguiente recurso las ventajas de estos tipos de pruebas.

Acordeón

DI\_CF13\_1.1.6 Ventajas de los tipos de pruebas

**Desventajas de los tipos de pruebas**

Figura 6. Tiempo y costo

Todos los tipos de pruebas mejoran la experiencia del usuario, las funcionalidades y garantizan la calidad, es decir, no hay desventajas sino más bien ventajas, por lo tanto, lo que sí se podría establecer como desventaja sería el tiempo **y el costo** relacionado con la prueba de s*oftware*. Realizar pruebas requiere de recursos y esfuerzos, y se puede dar un riesgo asociado con resultados inexactos. No obstante, el no realizar pruebas a las aplicaciones podría traer problemas al producto.

**1.1.7 Documentos de caso de pruebas**

Los productos que resultan del proceso de desarrollo de *software* son identificados como artefactos, por ejemplo, el código fuente, defectos, plan, casos y *script* de pruebas.

El aseguramiento de la calidad de *software* se compone de artefactos específicos que se forman en las diversas etapas del proceso de SQA, como se muestra en la siguiente infografía.



Figura 7. Testeo de código



En el diseño de las pruebas de *software* se deben identificar y describir casos de prueba, por lo tanto para documentar un caso de pruebas se debe comprender que este contiene un conjunto de variables o condiciones en la 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 tenemos la información necesaria para su proceso de elaboración, teniendo en cuenta que al momento de probar un *software* es de gran 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.

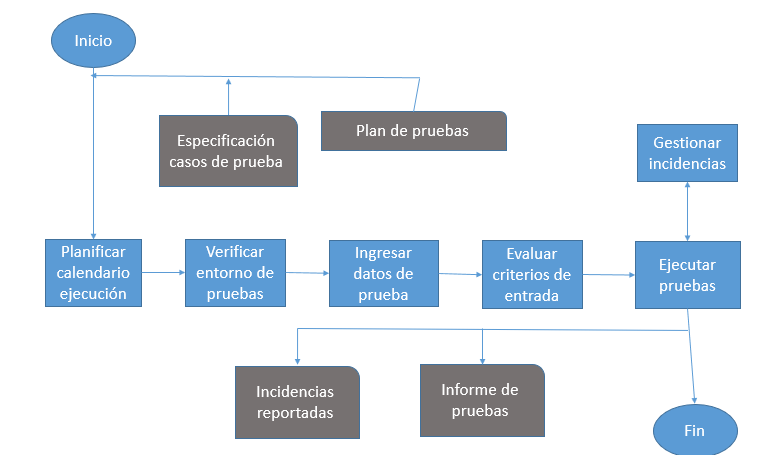
Tabla 1. Elementos de un caso de prueba

|  |  |
| --- | --- |
| Caso de prueba | |
| Identificador | Puede ser alfanumérico o numérico |
| Nombre | Nombre del caso de prueba de manera concisa |
| Descripción | Objetivo del caso de prueba, también describe que 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 de 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. |

Identificar, describir y documentar casos de pruebas facilita el trabajo cuando se requiere evaluar y comparar las modificaciones a los requerimientos que inicialmente ha definido el cliente a lo largo del tiempo, este seguimiento permite comprobar que los resultados obtenidos están acordes con lo requerido o solicitado, logrando así la satisfacción del cliente, lo que los define como un aliado principal al momento de garantizar la calidad del *software*.

**Ejecución de las pruebas**

Figura 8. Diagrama actividades



La ejecución de pruebas inicia creando los datos de prueba requeridos para ejecutar los casos de prueba establecidos. La ejecución de los casos de prueba se puede realizar de forma automatizada o manual, en cualquiera de las dos formas se debe registrar y documentar los fallos detectados en el sistema y su posterior corrección. En la Figura 8, se puede observar el flujo de las actividades en relación a la ejecución de pruebas.

El diagrama de actividades anterior expresa el flujo que se debe tener en cuenta al momento de ejecutar las pruebas de *software*, es por eso que a continuación se describen cada una de las actividades.

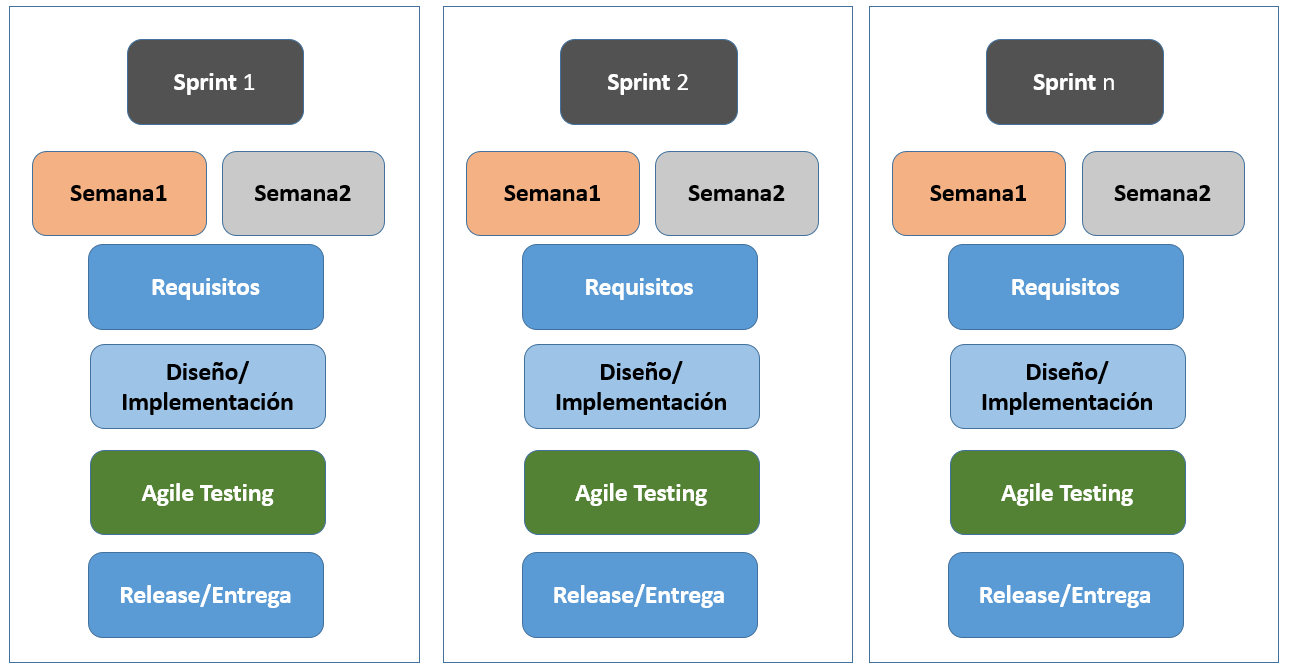
* **Planificar calendario detallado de ejecución:** definir el calendario que especifique la ejecución de los casos de prueba, teniendo en cuenta el plan y los casos de prueba.
* **Verificar el entorno de pruebas:** verificar que el ambiente de pruebas tenga un funcionamiento correcto.
* **Integrar los datos de pruebas:** realizar una recolección de los datos de prueba a utilizar en la ejecución de pruebas.
* **Evaluar criterios de entrada:** analizar los criterios de entrada contenidos en el plan de pruebas, de tal forma que se pueda verificar la ejecución de las pruebas.
* **Ejecutar las pruebas:** se incluyen las siguientes actividades:
  + Ejecutar casos y *script* de prueba.
  + Analizar y registrar los resultados.

**1.2 Agile testing**

Las metodologías ágiles proveen un conjunto de pautas y principios que buscan facilitar y priorizar la entrega de producto sobre procesos de documentación exhaustiva, haciéndolos más simples, donde interactúa el cliente final desde las primeras etapas del proyecto, lo cual es de resaltar que una metodología ágil tiene iteraciones o ***sprints*,** cada *sprint* tiene una duración que no es muy extensa, aproximadamente dos a cuatro semanas y así mismo, cuando se finaliza la iteración, es en este momento donde ya existen funcionalidades del *software* que pueden ser utilizadas por el cliente.

Habiendo recordado las metodologías ágiles, se puede especificar que las pruebas ágiles (*Agile testing*) es una práctica que se realiza en cada iteración, por lo tanto existen dos tipos de planificaciones *reléase*/entrega que hace referencia a las entregas frecuentes al cliente, es decir, poner en producción nuevas versiones del producto y *sprint*/iteración que hace referencia a que un proyecto se ejecuta en bloques temporales cortos y fijos, no obstante, estas se pueden dar en un mismo momento, es decir, que puede incluir una sola planificación con actividades tanto del *release* como del *sprint,* tal como se muestra en la figura 9.

Figura 9. Actividades *sprints* *release*



Por otro lado, también se debe tener en cuenta que la metodología de desarrollo de *software* tradicional en cascada utiliza diferentes tipos de pruebas de *software* que probaran tanto requerimientos funcionales como no funcionales de manera secuencial. No obstante, en metodologías ágiles, específicamente en las pruebas ágiles, también se consideran varios tipos de pruebas, pero teniendo en cuenta que se utilizan *sprints* o interacciones cortas e interacciones continúas entre los equipos de desarrollo, diseño y pruebas.

Las pruebas ágiles contienen principios que se aplican al desarrollo ágil de *software*, así mismo existen ciertos cuadrantes que se pueden tener en cuenta al momento de planificarlas, lo cual a continuación se describen tanto los principios como los cuadrantes:

**Principios de las pruebas ágiles (*Agile testing*)**

El *Agile testing* contiene los siguientes principios que son aplicados a un buen desarrollo de *software*:

Pestañas

DI\_CF13\_1.2 Principios de las pruebas agiles

**Cuadrantes de las pruebas ágiles (*Agile testing*)**

Para planificar las pruebas ágiles nos podemos basar en los cuadrantes del *agile testing,* que nos aporta una biblioteca de tipos de pruebas que se pueden usar para cumplir con los requerimientos.

Los cuatro cuadrantes son una clasificación que contribuye al momento de planificar las pruebas ágiles, asegurando que se tengan en cuenta los métodos y recursos para alcanzar productos de *software* de calidad.

LLAMADO A LA ACCION

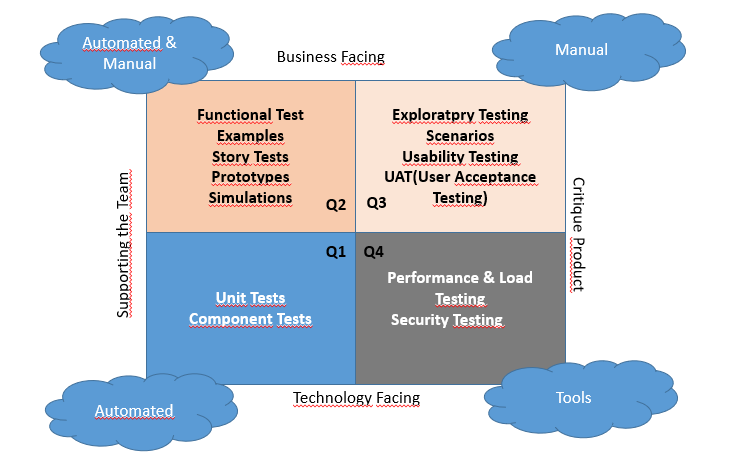
Si es de su interés conocer más sobre los cuadrantes de pruebas ágiles, se le invita a consultar el videotutorial en el material complementario, al cual puede acceder en el siguiente enlace.

https://www.youtube.com/watch?v=gz7A7EMZT\_4



Las pruebas de los cuadrantes se enfocan en apoyar al equipo de desarrollo a través del desarrollo del producto, puesto que orientan el desarrollo de la funcionalidad, posteriormente contribuyen a la introducción de nuevo código y la refactorización sin que genere resultados imprevistos en el comportamiento del sistema. En la Figura 10, se puede observar los cuadrantes de las pruebas ágiles, teniendo como ejes ‘las pruebas de apoyo al equipo’ y las tecnologías que facilitan el proceso.

Figura 10. Cuadrantes pruebas ágiles

****

A continuación, se realizará una exploración de los diferentes cuadrantes expuestos:

Slyders B

DI\_CF13\_1.2 Cuadrantes de pruebas ágiles

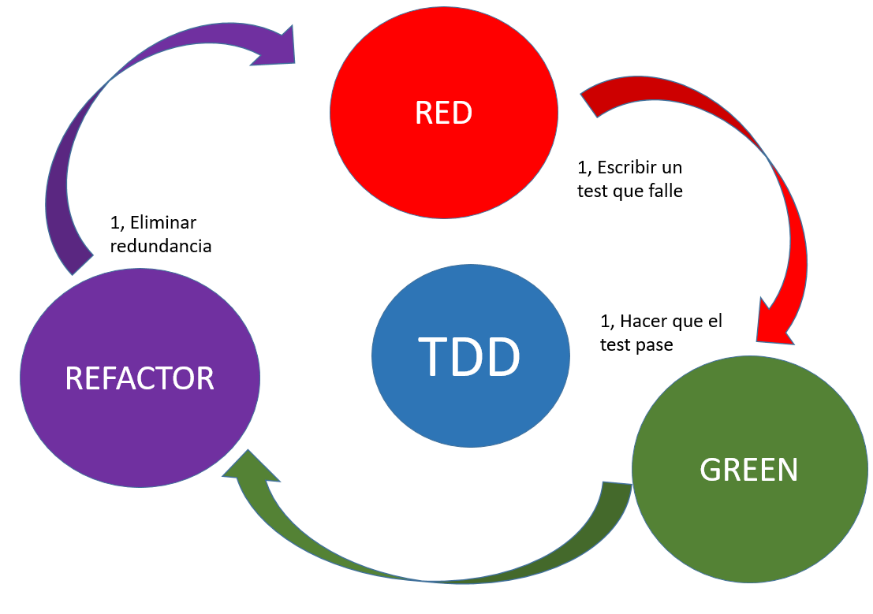
A continuación, se describen algunas recomendaciones para tener en cuenta al momento de trabajar con pruebas de *software* ágiles (*Agile testing*).

Pestañas

CF13\_1.2\_Pruebas de software agiles (Agil Testing)

**1.2.1 *Test Driven Development* (TDD)**

Figura 11. Flujo del TDD



Es una técnica de desarrollo que permite crear *test* antes del código de producción, es decir, en la implementación final. La esencia del TDD es crear un *test* antes de la implementación, de tal modo que se realice la alimentación y el *test* pueda pasar y que además va de la mano con la refactorización. Normalmente se tienen requerimientos que son transmitidos en historias de usuario como criterios de aceptación o pueden ser requerimientos como casos de uso, posteriormente se recrean esos requerimientos por medio de pruebas automatizadas, luego estas guían para realizar el funcionamiento con buenas prácticas de desarrollo.

**En esencia son tres pasos del desarrollo guiado por pruebas TDD:**

***Red***: es así debido a que se crea un *test* que falle y cuando falla en la consola aparece con letras rojas.

***Green*:** este paso es hacer que el *test* pase y es llamado *Green* porque cuando pasa un *test*, normalmente en la consola aparece con letras verdes.

**Refactor**: una vez que el *test* pasa, se va a detectar u observar código que se pueda mejorar y si no se está aplicando una buena práctica, es el momento de hacerlo.

A continuación, se mostrará en términos generales como sería el flujo del TDD con pasos y un ejemplo sencillo, no obstante también se hace la invitación a observar el videotutorial llamado “Pruebas unitarias y *Test-Driven Development*”.



LLAMADO A LA ACCIÓN

Si es de su interés conocer más sobre el proceso de realizar las pruebas unitarias y *Test-Driven Development,* se le invita a consultar el videotutorial en el material complementario, al cual puede acceder en el siguiente enlace.

https://www.youtube.com/watch?v=YuRdaR6wwWU

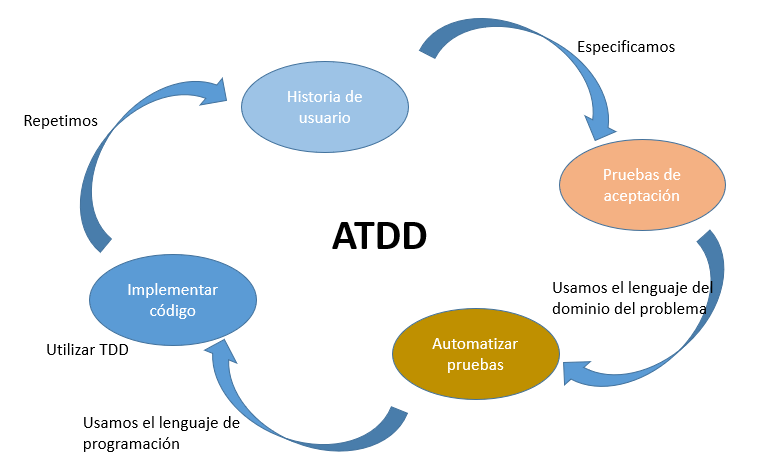
En la siguiente tabla se puede verificar cada uno de los pasos guiados por las pruebas TDD, donde se explican cada uno y se dan ejemplos.

Tabla 2. Flujo del TDD

|  |  |  |
| --- | --- | --- |
|  | Pasos | Ejemplo |
| *Red* | 1. Lo primero es que el cliente escriba una historia de usuario. | Supóngase que el cliente solicita que se desarrolle una aplicación con operaciones matemáticas sencillas que sume y reste. |
| 2. Los criterios de aceptación de esta historia se escriben con el cliente, deben estar bien detallados. | Se define con el cliente que el criterio de aceptación está relacionado con: si se introduce en la aplicación dos números y se suma o resta, entonces la aplicación visualiza el resultado en pantalla. |
| 3. Se debe seleccionar el criterio de aceptación más sencillo y se traduce en una prueba unitaria. | De acuerdo a este criterio, se empieza a especificar o definir el funcionamiento del algoritmo para la suma y la resta y luego se convierte el criterio de aceptación en una prueba específica, por ejemplo, un algoritmo que si se introduce los números 3 y 8 devuelve 11.  Ejemplo de cómo sería la clase para calcular: public void testSuma() {  assertEquals(8, Operaciones.suma(3,8)); } |
| 4. Se verifica que la prueba falla. | Si se intenta pasar este *test* resultará en error, debido a que la clase Operaciones todavía no existe. |
| Green | 5. Se escribe el código que hace pasar la prueba. | El siguiente paso es escribir el código de la clase, en este momento ya se sabe cómo se va a comportar.  public class Operaciones { public static int suma (int a, int b) { int c = a + b; return c; } } |
| 6. Se ejecutan todas las pruebas automatizadas. | Posteriormente se ejecuta la prueba y en este momento se tiene el código funcionado con la prueba pasada. |
| Refactor | 7. Se ejecutan todas las pruebas automatizadas. | Cuando esté todo funcionando, se pasa a refactorizar y a eliminar código duplicado. |
| 8. Se refactoriza y se limpia el código. | Es necesario pasar todos los *test* después de refactorizar. |

**1.2.2 *Acceptance Test Driven Development* (ATDD)**

Figura 12. Imagen ATDD



El desarrollo guiado por pruebas de aceptación (ATDD) es una metodología de desarrollo basada en la comunicación entre los clientes empresariales, los desarrolladores y los *testers.* Antes de que los desarrolladores comiencen a codificar, ATDD incluye las pruebas de aceptación, pero destaca la escritura de pruebas de aceptación. El proceso ATDD sigue estos pasos:

1. **Seleccione la historia de usuario:** en este punto se asume que se realizó una priorización que permite conocer cuál historia de usuario es la próxima a elegir.

2. **Escribir la prueba de aceptación:** en este punto todos los involucrados deben participar colaborativamente.

3. **Implementar la historia de usuario - Automatizar las pruebas:** para este paso existen variadas formas de llevarla a cabo, sin embargo, lo importante es que, a partir de las pruebas de aceptación del paso anterior, se obtienen pruebas ejecutables, que permitirán conocer cuánto de la funcionalidad aún falta.

4. **Implementar el código:** al igual que el paso anterior, existen diversas formas de hacerlo, una de ellas es utilizando el TDD.

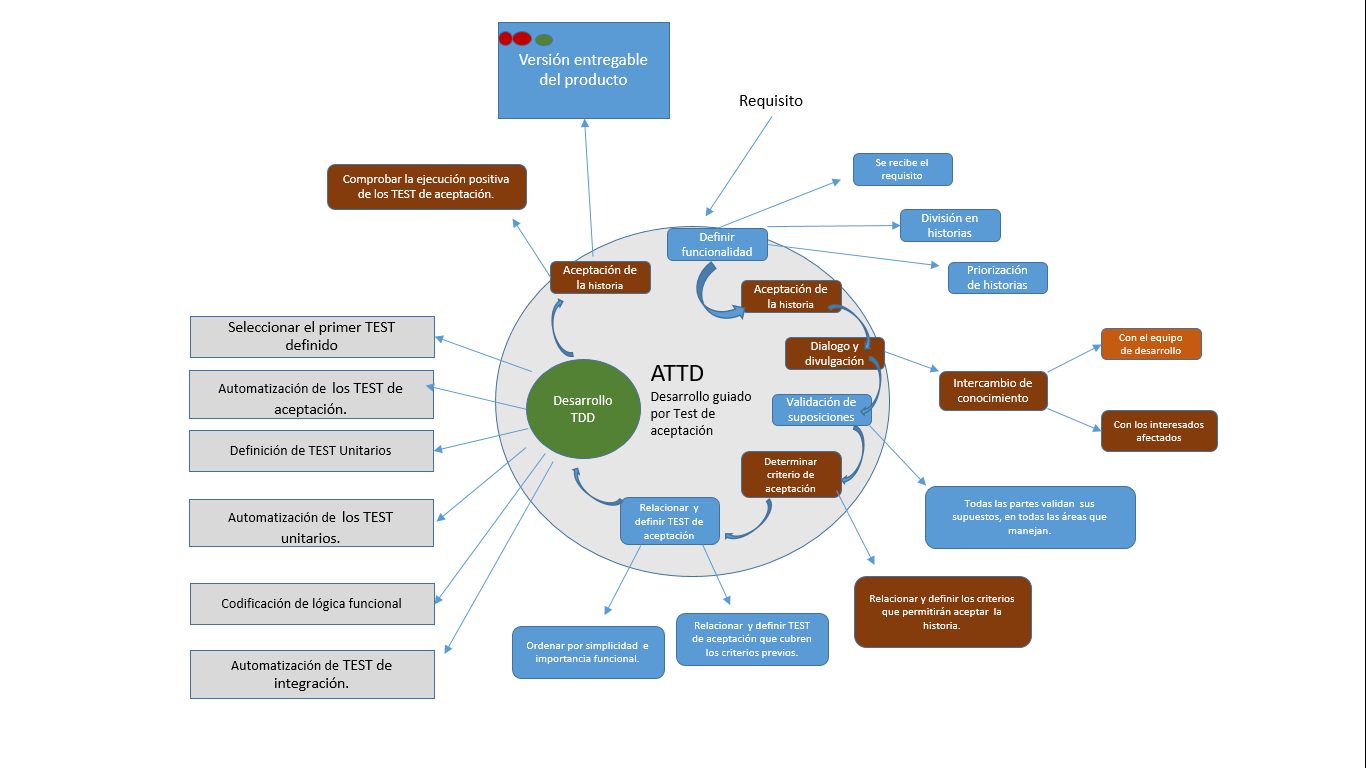
5. **Realizar pequeños cambios/refactorización:** es la base de esta práctica al igual que TDD, es una práctica iterativa que finaliza cuando los requerimientos del cliente han sido satisfechos.

De acuerdo con lo anterior, se parte de un desarrollo al que los programadores han aplicado TDD. Si lo han hecho correctamente, deberían haber obtenido un código limpio y refactorizado, que ha sido probado por pruebas unitarias. El código, por lo tanto, es bueno, pero ¿sucede lo mismo con la aplicación?

ATDD no es una técnica de desarrollo o una buena práctica; tiene un enfoque superior. Es una metodología de trabajo para desarrollar *software*, que busca comprobar si los pasos que se siguen en el desarrollo son los mejores para llegar al objetivo. Con TDD solo se ve si lo que se desarrolla está codificado correctamente y sin errores, es un enfoque miope, solo centrado en desarrollar código de calidad, no la aplicación en su conjunto.

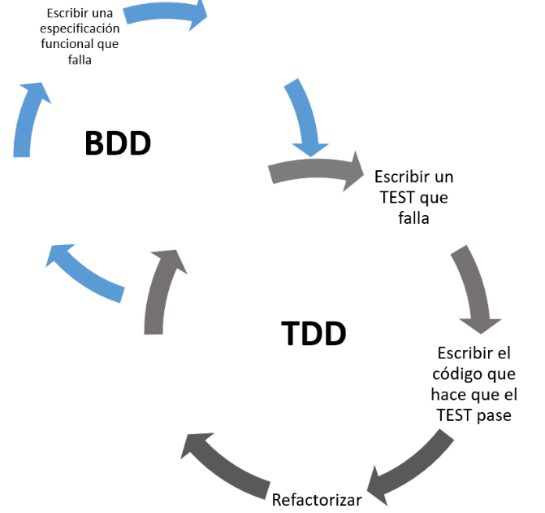
Al igual que ocurre en TDD, ATDD consta de una serie de pasos secuenciales que se presentan en la siguiente figura 13.

Figura 13. ATDD



**1.2.3 *Behaviour Driven Development* (BDD)**

Figura 13. Imagen BDD

El desarrollo dirigido por comportamiento (BDD) propuesto inicialmente por Dan North, es una síntesis y refinamiento de las prácticas de ingeniería de *software* que ayudan a los equipos generar y entregar *software* de mayor calidad rápidamente. El proceso de BDD es similar a TDD y sigue estos pasos: 

1. Escribir un escenario.

2. Ejecutar el escenario que falla.

3. Escribir la prueba que corresponde a las especificaciones del escenario.

4. Escribir el código más simple para pasar la prueba y el escenario.

5. Refactorizar para eliminar la duplicación.

BDD está basado en la técnica TDD, y corrige ciertas deficiencias que presenta TDD. El principal problema de TDD se produce cuando un *software* pasa correctamente un *test* y no consigue la funcionalidad deseada. Que un *software* pase correctamente un *test* no significa que haga lo que el cliente esperaba de él.

Un ejemplo muy sencillo es el siguiente: primero definimos la característica o funcionalidad a implementar en la historia de usuario.

Característica: un usuario quiere acceder a una plataforma *web* universitaria para descargar sus apuntes. Posteriormente se invita a ver la siguiente imagen que contiene dos escenarios a partir de la historia de usuario definida.

Figura 13. Infografía ejemplo BDD



**1.2.4 *Testing* exploratorio**

Las pruebas exploratorias han sido de gran interés en la comunidad de pruebas de *software*.

Este enfoque de pruebas generalmente implica probar, aprender y diseñar nuevas pruebas como actividades que interactúan simultáneamente.

Casi todos los *testers* exploran cómo es el rendimiento de su trabajo, ya sea que lo reconozcan o no. Por ejemplo, considere las pruebas de regresión de errores. Para verificar si se corrigió un error, el *tester* puede comenzar con los pasos exactos enumerados en el informe de errores, pero después de que el programa pase esta prueba simple, el *teste*r probablemente intentará pruebas adicionales para verificar si el error está completamente solucionado.

El *testing* exploratorio es una combinación del diseño, ejecución y aprendizaje sobre el aplicativo que está bajo prueba, lo cual tiene un proceso de exploración y aprendizaje del producto bajo prueba. No se utiliza un guion o *script* de pruebas previamente diseñado, además es útil en metodologías ágiles.

LLAMADO A LA ACCIÓN

Si es de su interés conocer más sobre el *Testing* exploratorio, se le invita a consultar el *webinar* en el material complementario, al cual puede acceder en el siguiente enlace.

https://www.youtube.com/watch?v=j0eAvYSAbA8



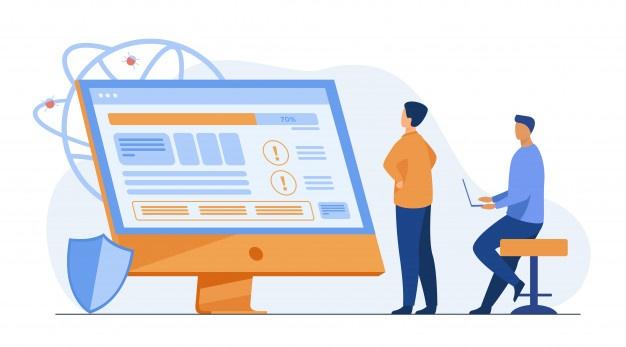
Después de observar el proceso exploratorio del *software*, se puede identificar las pruebas exploratorias que son:

Figura 14. Imagen *testing* exploratorio



* **Exploración del producto**: para conocer a fondo cómo cumplir con los requisitos hay que registrar los objetivos, las funciones, los tipos de datos que se procesan y las zonas de inestabilidad del producto.
* **Diseño de pruebas:** crear diferentes estrategias para observar y evaluar por completo el producto.
* **Ejecución de pruebas:** explorar el producto para poder formular una hipótesis de cómo funciona y cuáles pueden ser sus puntos débiles.
* **Heurística**: reglas generales que ayudarán a cómo probar correctamente el producto.
* **Resultados revisables**: cuando se finalicen las pruebas exploratorias, el *tester* debe ser capaz de explicar cualquier aspecto del programa y mostrar cómo se cumplen los requisitos indicados en el procedimiento.

**1.2.5 Automatización de pruebas de regresión**

Figura 15. Automatización de pruebas

Aseguran que el cambio realizado en la aplicación o sistema no provoca una falla en la funcionalidad anterior, además consume mucho tiempo si es de manera manual, ya que en ocasiones es más caro la prueba de regresión que lo nuevo que se va a probar y es propenso a tener errores por el factor humano.

Las pruebas de regresión se efectúan por flujos completos, es ejecutar y realizar las pruebas con las distintas opciones con la que se cuentan, con base en la lógica del negocio, pueden contener a las partes del código modificado, porque una vez que ese elemento tiene cambios, se debe asegurar que hay una conexión que no tiene problemas con el ambiente en donde va a vivir.

Se debe analizar los flujos que son los que más convienen y no limitarse a probarlos, debido a que será más costoso que falle en producción.

Estas pruebas toman tiempo en ejecutarlas, sin embargo, al automatizar pruebas, las pruebas de regresión serán asertivas y rápidas, también si se cuenta con los ***scripts*** de las pruebas cualquier persona podrá ejecutarlas, sin requerir de un grupo de *testers* capacitados para estas pruebas, de hecho, si la aplicación original ya tiene mucho tiempo desde la primera vez que se liberó a producción, entonces el riesgo de no contar con alguien que tenga el conocimiento de los flujos se reduce.

**Figura 17. Pruebas de regresión**

Este tipo de pruebas son lanzadas por los desarrolladores o los *tester*, y es posible su automatización. Para este tipo de pruebas se utilizan las mismas herramientas que se usan en las pruebas unitarias y en las de sistemas, pero también tienen herramientas especializadas de código abierto como:

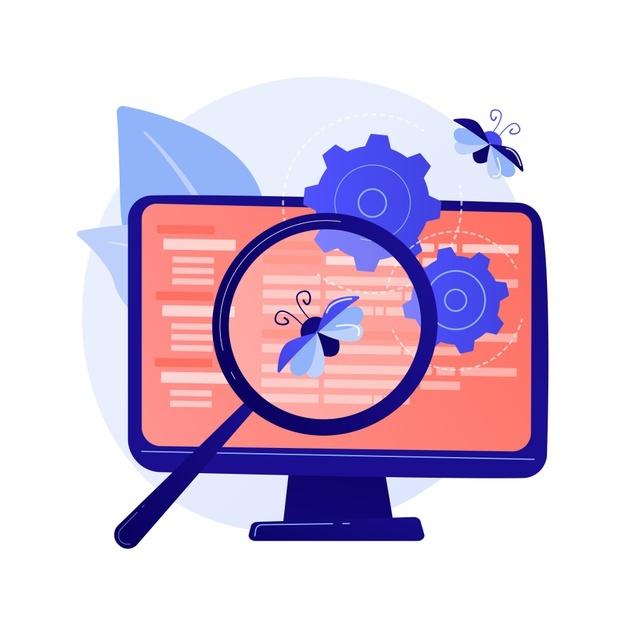


**1.2.6 Pruebas de exploratorias, usabilidad y aceptación**

Las pruebas de usabilidad y las pruebas de aceptación del usuario pueden tener diversas semejanzas en términos de resultados y objetivos. No obstante, se podría decir que la primera encuentra errores en el sistema y la segunda demuestra que el sistema está listo, es decir, uno trata de realizar un mejor producto y el otro trata de terminarlo.

**Pruebas exploratorias**

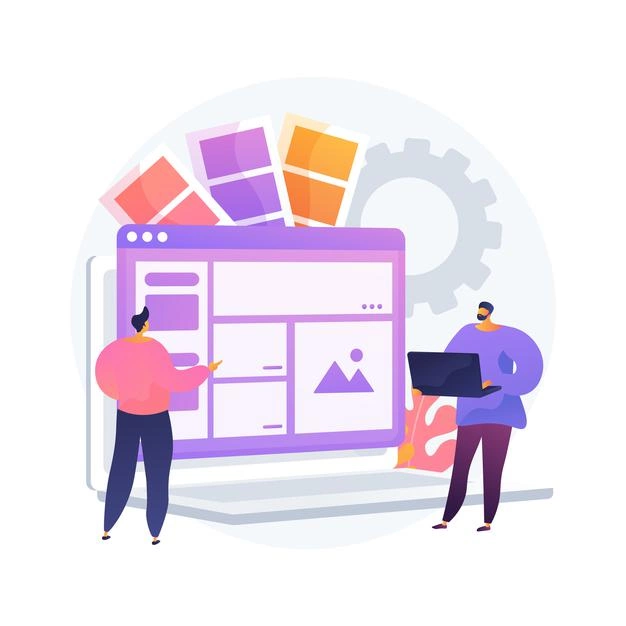
Figura 16. *Testing* exploratorio



El *testing* exploratorio se define como el aprendizaje, el diseño de casos de prueba y la ejecución de las pruebas en forma simultánea. En otras palabras, es una técnica de prueba en la cual quien prueba controla activamente el diseño mientras son realizadas, y utiliza la información obtenida en la exploración para diseñar nuevas y mejores pruebas.

**Pruebas de usabilidad**

Figura 17. Usabilidad



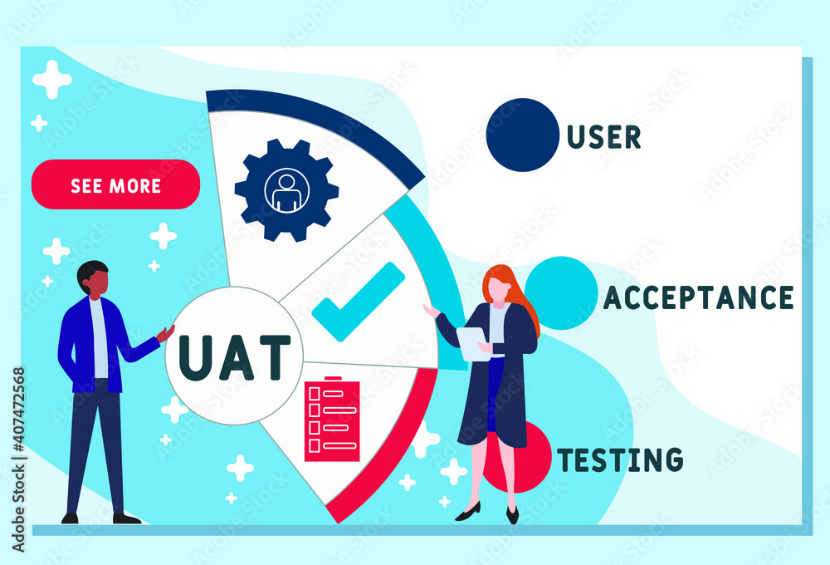
El diseño no es solo como se ve y como se siente. El diseño es cómo funciona, y para conocer cómo funciona se debe de tener el conocimiento de usabilidad, por lo tanto, es un método para evaluar la experiencia del usuario de una aplicación o producto *software*. Cuando se prueba la usabilidad con un conjunto representativo de clientes o usuarios, los evaluadores de la experiencia de usuario saben por ejemplo si los usuarios reales pueden usar la aplicación o el producto *software* de forma sencilla e intuitiva.

La usabilidad es una parte crítica del proceso de desarrollo *software*, debido a su naturaleza iterativa. Una evaluación de usabilidad es la mejor manera de poner un producto en manos de usuarios reales para ver si lo usan y cómo lo usan antes del lanzamiento del producto.

**Pruebas de aceptación**

Estas pruebas se realizan asignando a los usuarios unas tareas muy específicas con pasos a seguir para comprobar la funcionalidad de un sistema, es decir, identificar los *bugs*, que posteriormente los desarrolladores deben trabajar para solucionarlos.

Figura 18. Pruebas de aceptación

Durante las pruebas de aceptación de usuario, el usuario solicita un producto de *software* para asegurarse de que cumple con sus requerimientos, satisface sus expectativas y que su resultado es factible, es decir, que el *software* le permita realizar su trabajo.

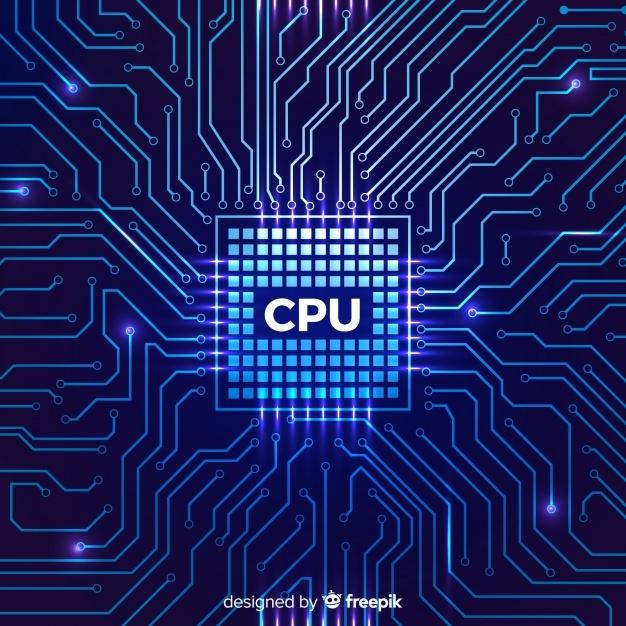
La UAT se centra en el usuario debido a la colaboración entre el equipo de desarrollo y los usuarios finales al momento de detallar los criterios de aceptación para cada historia de usuario desarrollada. En este sentido los desarrolladores o el equipo de desarrollo los utiliza con el propósito de diseñar casos de prueba automatizados que se ejecutan cada vez que se realiza una compilación de integración.

**1.2.7 Pruebas desempeño, carga y seguridad**

Es un momento exitoso cuando se finaliza la programación de una aplicación, lo cual, al transcurrir el tiempo, tener un producto terminado es un logro importante. No obstante, es momento también de probar varios aspectos del *software*, para asegurar su funcionamiento y controlar su calidad. Desempeño, seguridad y carga son factores importantes que se deben revisar con todo el producto *software* que se desarrolle, a continuación se describe cada uno:

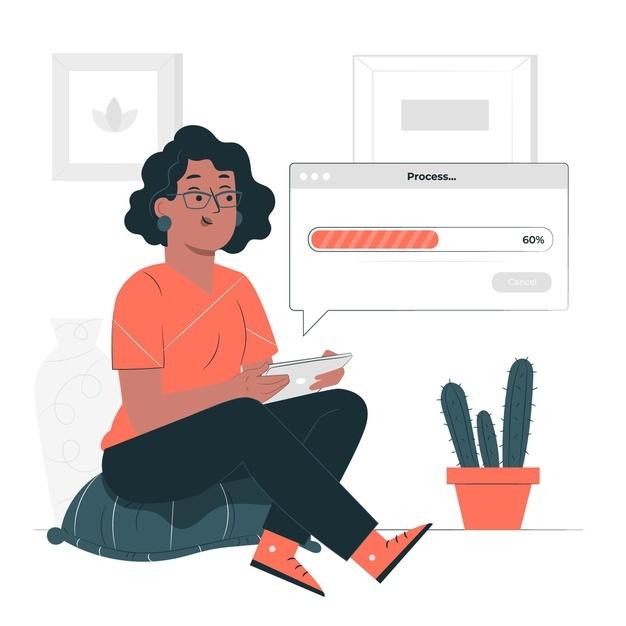
**Pruebas de Desempeño**

Figura 19. Memoria de CPU

Las pruebas de desempeño permiten determinar el comportamiento de un sistema en términos de estabilidad y respuesta con respecto a un escenario en particular, referente a la carga de trabajo, concurrencia, ancho de banda, características particulares de equipos o servidores, como Ram, CPU. También permiten identificar, medir, investigar, validar o verificar otros aspectos de calidad, tales como la seguridad, escalabilidad y el consumo de recursos. Estas pruebas constituyen unos estándares de ejecución en la implementación, diseño y arquitectura de un producto *software*. Por lo tanto, una de las pruebas de desempeño esenciales es la prueba de carga que se menciona a continuación. 

**Pruebas de carga**

Figura 20. Proceso de carga



Las pruebas de carga se centran en determinar o validar las características de rendimiento del sistema o la aplicación sometida a un *test* cuando se enfrenta a diferentes volúmenes de cargas de trabajo prevista durante las operaciones de producción (Meier, y otros, 2007).

**Pruebas de seguridad**

Figura 21. Seguridad de los datos

Las pruebas de seguridad son la serie de actividades que son ejecutadas para encontrar fallas y vulnerabilidades, con la finalidad de minimizar el impacto de ataques, acceso a información susceptible y la pérdida de información. 

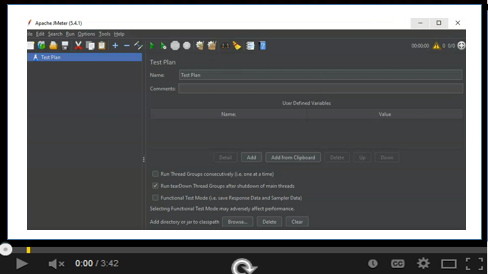
Estas pruebas son primordiales dado que los productos *software* deben asegurar siempre la confidencialidad, disponibilidad e integridad de los datos y funcionalidades que maneja el producto

y es importante denotar lo crítico que sería la pérdida o corrupción de esta información en el contexto empresarial. Estas pruebas deben ir de la mano y ejecución en todas las fases del proyecto.

Para ilustrar mejor el concepto de las pruebas de carga se presenta el siguiente video tutorial:

Video Tutorial

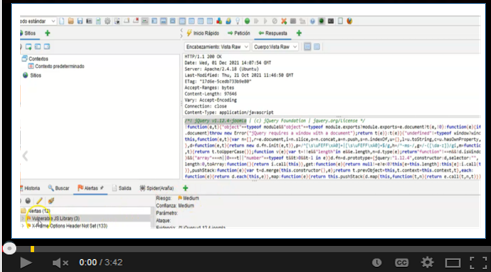
DI\_CF13\_1.2.7\_Prueba de carga



A continuación, se presenta el siguiente video tutorial de instalación del *software* **OWASP** para realizar una prueba de seguridad:

Video Tutorial

DI\_CF13\_1.2.7\_Prueba de seguridad



**1.3. Elaboración de informe de resultados**

Una vez ejecutadas las pruebas se deben analizar los resultados y los fallos detectados, teniendo en cuenta el reporte de defectos, directrices para detectarlos y el informe de resultados de pruebas. Para ello se puede realizar lo descrito a continuación:

CF13\_1.3 Elaboración de informe de resultados -Tarjeta Avatar Slide

El informe de resultados incluye la detección de incidencias o errores de *software*, puesto que es aquí donde se evalúan y corrigen problemas, por lo tanto, a continuación, se describe la gestión de incidencias y su ciclo de vida.

**Incidencias detectadas**

Figura 22. Errores *bugs*

La gestión de incidencias es un elemento central e importante en el proceso de calidad de *software* puesto que es en este punto donde se tienen en cuenta los errores (*bugs)*. Por lo tanto, el propósito de las incidencias detectadas es precisamente generar su corrección, de tal manera que sea mínimamente probable que el error se repita.

**Ciclo de vida de una incidencia**

Un sistema de gestión de incidentes debe manejar los errores de *software* teniendo en cuenta el siguiente proceso:

Imagen Interactiva modal

DI\_CF13\_1.3 Ciclo de vida de una incidencia de software (Bug)

El anterior proceso se adapta a proyectos de *software* en fase de desarrollo, mas no en ambiente de producción.

**Redacción del reporte de Incidencias**

El propósito del informe de incidentes es documentar cada incidente, por lo tanto, se debe seguir un procedimiento para gestionar de forma sistémica y ordenada las incidencias de producto halladas a lo largo de la ejecución de los casos de prueba. Este procedimiento tiene las siguientes actividades:

* Verificar con las áreas correspondientes las incidencias.
* Verificar que el estado de las incidencias esté actualizado.
* Realizar repruebas y regresión, es decir, generar otra vez la ejecución de los casos de prueba necesarios.



**Reporte de defectos**

Cuando se está seguro de que existe un defecto, es decir, cuando están confirmados, se procede a redactar un reporte de defectos, el cual contiene los fallos detectados a lo largo del esfuerzo de pruebas. También hay que tener en cuenta que cada defecto detectado requiere de un reporte de defectos lo que facilita la trazabilidad y su posterior corrección.

A continuación, se presenta la estructura de un reporte de defectos.

* + - 1. Identificador
      2. Título
      3. Reportado por
      4. Asignado a
      5. Fecha de creación
      6. Severidad
      7. Nombre de la aplicación
      8. Descripción
      9. Método de reproducción
      10. Información adicional

**Informe de resultados de pruebas**

El informe de resultados de pruebas presenta y ordena un análisis sintético de los resultados de las pruebas y las medidas principales de prueba para la evaluación y verificación, por lo tanto, este documento es responsabilidad del gerente de pruebas. Este informe debe contener los siguientes puntos:

1. Alcance de las pruebas, es decir, lo que es necesario incluir y no incluir.
2. Resumen de pruebas:

* Casos de prueba planificados
* Casos de prueba ejecutados
* Casos de prueba exitosos
* Casos de prueba fallidos
* Casos de prueba bloqueados

1. Listado de defectos detectados.
2. Conclusiones.

**2. Elaboración del plan de mejora**

Cuando una organización es exitosa seguramente lo es por su capacidad de incluir la mejora continua en cada proceso y en sus actividades diarias. La mejora se efectúa cuando la organización tiene en cuenta tanto las fortalezas como las debilidades. Sin embargo, también es importante identificar la situación en la que se encuentra la organización por medio de un diagnóstico. Cuando ya se tiene el diagnóstico se puede establecer la estrategia a seguir con el fin de que el destinatario de los servicios observe significativamente, la mejora implantada.

**2.1 Identificar el área de mejora**

Una vez elaborado el diagnóstico, se conocen las principales fortalezas y debilidades. Por lo tanto, es importante identificar las áreas de mejora apoyándose en las fortalezas para lograr superar las debilidades, siendo esta una opción de cambio óptima. A continuación, se muestra una tabla como una opción para identificar el área de mejora de acuerdo con las fortalezas y debilidades.

|  |  |  |
| --- | --- | --- |
| Fortalezas | Debilidades | Áreas de mejora |
| 1. | 1. | 1. |
| 2. | 2. | 2. |
| (..) | (..) | (..) |

Tabla 3. Tabla de fortalezas, debilidades y área de mejora

**2.2 Detectar las principales causas del problema**

Solucionar problemas y mejorar un área en específico, empieza conociendo las causas que las originaron. Por lo tanto, hay actualmente varias herramientas metodológicas que las identifican. A continuación, se nombran algunas:

Tarjetas-tipo Over

DI\_CF13\_2.2 Herramientas metodológicas

El uso de estas herramientas aporta al análisis del problema en mayor profundidad y se convierten en pautas al momento de establecer las acciones de mejora.

**2.3 Formular el objetivo**

Una vez conocidas e identificadas las causas del problema y las áreas de mejora, se pueden establecer los objetivos y determinar el tiempo para lograrlos. Al momento de escribir los objetivos estos deben ser:

* Precisos o concretos.
* Expresar de forma indiscutible el resultado que se quiere alcanzar.
* Su redacción debe ser clara.
* Deben ser alcanzables, comprensibles y flexibles, es decir, que se puedan cumplir, interpretar y en algún caso extremo, modificar.

**2.4 Seleccionar las acciones de mejora**

En este punto se debe seleccionar las opciones de mejora para priorizar las más apropiadas. Para esto se pueden utilizar algunas técnicas tales como técnica del grupo nominal y tormenta de ideas entre otras, que permiten establecer acciones de mejora que se convierten en solución para las debilidades. Lo anterior quiere decir que, para cumplir con los objetivos establecidos, se debe de tener un listado de las principales actuaciones. A continuación, se muestra una tabla que muestra que, para seleccionar las acciones de mejora, se deben haber identificado el problema, la causa y su objetivo.

|  |  |
| --- | --- |
| ÁREA DE MEJORA Nº 1: | |
| Descripción del problema |  |
| Causas que provocan el problema |  |
| Objetivo a conseguir |  |
| Acciones de mejora | 1.  2.  (..) |
| Beneficios esperados |  |

Tabla 4. Acciones de mejora

**2.5 Realizar una planeación**

El listado de las principales actuaciones se genera sin tener en cuenta algún orden de prioridad, no obstante, ciertas restricciones que son inherentes a las acciones pueden limitar su comienzo, o la posible eliminación o aplazamiento del plan de mejora, por lo tanto, es importante conocer las restricciones que limitan su viabilidad. Determinar un orden óptimo no se da por el hecho de identificar las acciones relacionadas a los factores con mayor urgencia, sino que se tienen en cuenta otros criterios al tomar esa decisión. A continuación, se describen algunos criterios:

* **Dificultad de la implantación**: en este momento se prioriza el grado de dificultad de menor a mayor.

|  |  |  |  |
| --- | --- | --- | --- |
| Dificultad | | | |
| 1. Mucha | 2. Bastante | 3. Poca | 4. Ninguna |

Tabla 5. Dificultad de la implantación

* **Plazo de implantación:** es de resaltar que existen acciones de mejora, cuyo tiempo está establecido y no requiere de un esfuerzo profundo, por lo tanto, pueden efectuarse a corto plazo. Sin embargo, hay acciones que su realización requiere de un extenso tiempo de implantación.

|  |  |  |  |
| --- | --- | --- | --- |
| Plazo | | | |
| 1. largo | 2. medio | 3. corto | 4. inmediato |

Tabla 6. Plazo de implantación

* **Impacto en la organización:** este es el resultado de la actuación que se va a implantar, medido por medio del grado de mejora obtenido, puesto que, un cambio fundamental tendrá un impacto más grande que cambios continuos pequeños.

|  |  |  |  |
| --- | --- | --- | --- |
| Impacto | | | |
| 1. ninguno | 2. poco | 3. bastante | 4. mucho |

Tabla 7. Impacto en la organización

**2.6. Seguimiento del plan de mejora**

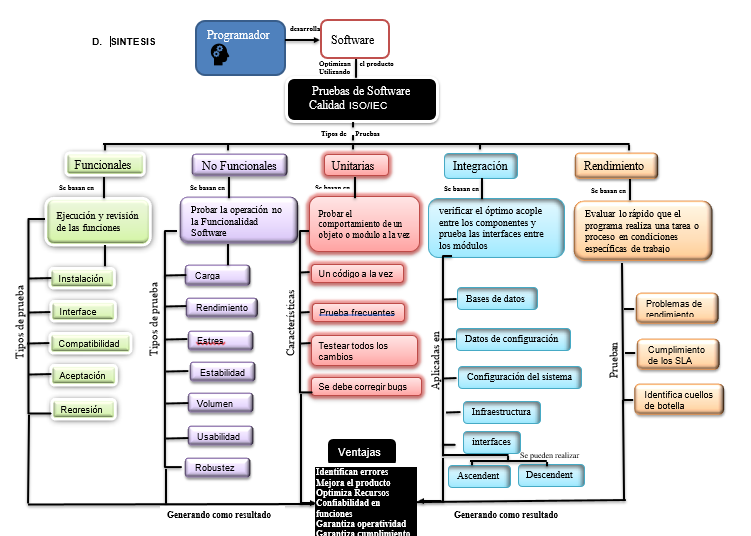
En este punto se elabora un cronograma para la implantación y seguimiento de las acciones de mejora. En este cronograma, se colocarán de forma ordenada las prioridades con los plazos determinados para su desarrollo.

Cuando ya se han seleccionado por orden de prioridad, ahora si se construye el plan de mejoras, incluyendo los elementos para efectuar el seguimiento en detalle del plan que garantizará su eficiencia y eficacia, teniendo en cuenta la tabla que se muestra a continuación:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PLAN DE MEJORAS | | | | | | | |
| Acciones de mejora | Tareas | Responsable de tarea | Tiempos (inicio-final) | Recursos necesarios | Financiación | Indicador seguimiento | Responsable seguimiento |
| 1.1 | a)  b)  c)  (…) |  |  |  |  |  |  |
| 1.2 | a)  b)  c)  (…) |  |  |  |  |  |  |

Tabla 9. Plan de mejoras

1. **Síntesis**

****

1. **ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)**

|  |  |
| --- | --- |
| 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 el término con la definición que corresponde |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | Anexo documento en Word llamado Actividad didáctica 1 |

1. **MATERIAL COMPLEMENTARIO:**

Relacionar el material de apoyo o complementario de los temas abordados en este recurso.

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| **1.1 Tipos de pruebas** | SENA, E. d. (24 de 08 de 2021). Tipos de pruebas de *software*. [Video]. YouTube. | Video | <https://www.youtube.com/watch?v=Uf1Kk52ONYc> |
| **1.1.1 Pruebas funcionales** | IBM (2021). Pruebas Funcionales. *Rational Test Workbench.* | Documento | <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> |
| **1.1.5 Pruebas de Rendimiento** | IBM (2021). Pruebas de rendimiento. *Rational Test Workbench.* | Documento | <https://www.ibm.com/docs/es/rtw/9.0.0?topic=phases-performance-testing> |
| **1.2 Agile Testing** | QA, H.-O. (04 de 09 de 2020). Cuadrantes de prueba. [Video]. YouTube. | Video | <https://www.youtube.com/watch?v=gz7A7EMZT_4> |
| **1.2.1 Test Driven Development (TDD)** | Zapata, M. (07 de 06 de 2020). Pruebas unitarias y *Test-Driven Development.* [Video]. YouTube. | Video | <https://www.youtube.com/watch?v=YuRdaR6wwWU> |
| **1.2.6 Pruebas de Exploratorias, Usabilidad y Aceptación** | SIlva, F. (2015, 27 mayo). Cómo mejorar la usabilidad de tu diseño web. Blog IDA Chile | Estrategia para el éxito de tu negocio. | *Blog* | <https://blog.ida.cl/diseno/como-mejorar-usabilidad-diseno-web/> |
| **1.2.6 Pruebas de Exploratorias, Usabilidad y Aceptación** | Zamora, A., Benitez, J., & M. M. (25 de 06 de 2020). Test de usabilidad: identificando mejoras con nuestros usuarios. [Video]. YouTube. | *Workshop* | https://www.youtube.com/watch?v=EeRtQUljvOM |

1. **GLOSARIO:**

Incorpore aquí las definiciones de los términos claves, requeridas para comprender adecuadamente los contenidos de este recurso educativo. Presentarlo en Orden Alfabético. *Máximo 15 palabras*.

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| **Automatización de pruebas** | Es la práctica que permite controlar la ejecución de un producto *software* de manera automática. |
| **Aplicación** | Una aplicación es un programa informático diseñado como una herramienta para realizar operaciones o funciones específicas. Generalmente, son diseñadas para facilitar ciertas tareas complejas y hacer más sencilla la experiencia informática de las personas. |
| **ATDD** | Desarrollo Orientado a pruebas de Aceptación (*Acceptance Test–Driven Development*) es una metodología de desarrollo basada en la comunicación entre los clientes comerciales, los desarrolladores y los evaluadores. |
| **BDD** | *Behaviour Driven Development* es una estrategia de desarrollo dirigido por comportamiento, se define en un idioma común entre todos los *stakeholders,* lo que mejora la comunicación entre equipos tecnológicos y no técnicos. |
| **Incidencia** | suceso que se produce durante una actividad y puede causar, una disminución de calidad de este. |
| **Integración continua** | es una práctica de desarrollo de *software* por medio de la cual los desarrolladores combinan los cambios en el código en un repositorio. |
| **QA** | Calidad de software (*Quality Software*) trata los conceptos, los métodos, las técnicas, los procedimientos y los estándares necesarios para producir productos y procesos *software* de alta calidad. |
| **TDD** | Desarrollo Guiado por Pruebas (*Test Driven Development*), con el TDD se puede agilizar el proceso de creación de código. Este se centra más por el qué y el por qué antes del cómo. |
| **UX** | Experiencia de usuario (*User Experience*) es el conjunto de factores y elementos relativos a la interacción del usuario con un entorno o dispositivo concretos, dando como resultado una percepción positiva o negativa de dicho servicio, producto o dispositivo |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

* Mera Paz, J. A. (19 de 10 de 2016). Pruebas de Calidad software. Repositorio Institucional. <https://repository.ucc.edu.co/handle/20.500.12494/962>
* Bustamante Ramírez, J. (2011). Sistema de informes para pruebas de software. Repositorio Dspace. <http://bibliotecadigital.iue.edu.co/xmlui/handle/20.500.12717/153>
* Clemente, P. J., & Gómez, A. (2014). Aplicación de un proceso de mejora continua en una asignatura de Desarrollo de *Software* Dirigido por Modelo*s.* [UPCommons. Portal de acceso abierto al conocimiento de la UPC](https://upcommons.upc.edu/). <http://hdl.handle.net/2099/15497>
* Beck, K., & Andres, C. (2004b). *Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series) (2nd ed.). Addison-Wesley.*
* Jeffries, R. (2011). What is Extreme Programming? Ronjeffries.Com. <https://ronjeffries.com/xprog/what-is-extreme-programming/>
* Kruchten, P. (2003). *The Rational Unified Process: An Introduction (3rd Edition) (3rd ed.). Addison-Wesley Professional.*
* Maida, EG, Pacienzia, J. (2015). Metodologías de desarrollo de *software* [en línea]. Tesis de Licenciatura en Sistemas y Computación. Facultad de Química e Ingeniería “Fray Rogelio Bacon”. Universidad Católica Argentina, 2015. <https://bit.ly/3hJMwXP>
* Manifiesto por el Desarrollo Ágil de *Software*. (2001). Agilemanifesto.Org. <https://agilemanifesto.org/iso/es/manifesto.html>
* Martin, J. (1991). *Rapid Application Development.* Macmillan Coll Div.
* Royce, W.W. (1970) *Managing the Development of Large Software Systems.* *Proceedings of IEEE WESCON*, 26, 328-388.
* SCRUMstudy. (2013). *A Guide to the Scrum Body of Knowledge (SBOK Guide) (2013th ed.). VMEdu Inc.*
* Sommerville, I., Galipienso, M. I. A., & Martinez, A. B. (2005). Ingeniería del *Software*. Pearson Educación.

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Ervin Andrade | Experto temático | SENA Regional Cauca CTPI | Diciembre de 2021 |
| Peter Pinchao | Experto temático | SENA Regional Cauca CTPI | Diciembre de 2021 |
| Gloria Lida Álzate Suárez | Diseñadora instruccional | Regional Distrito Capital - Centro para la Industria de la Comunicación Gráfica | Diciembre de 2021 |
| Carolina Coca Salazar | Revisora Metodológica y pedagógica | Regional Distrito Capital- Centro de Diseño y Metrología | Diciembre de 2021 |
| Rafael Neftalí Lizcano Reyes | Asesor pedagógico | Regional Santander - Centro Industrial del Diseño y la Manufactura | Diciembre de 2021 |

1. **CONTROL DE CAMBIOS**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) | Jhana Johanna Bustillo Ardila | Revisión de estilo | Regional Santander - Centro Industrial del Diseño y la Manufactura | Diciembre de 2021 | Revisión de estilo |