Aplicación de pruebas de “software”

Breve descripción:

En este componente, el aprendiz identificará que un “software” de calidad implica la utilización de diferentes tipos de pruebas, metodologías y procedimientos estándares para el análisis, diseño, programación y prueba del “software”. Esto con el objetivo de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, así como de elevar la productividad tanto en la labor de desarrollo como en el control de calidad.

Abril 2024

Tabla de contenido

Introducción

En este componente formativo, denominado "Aplicación de pruebas de software", se estudiarán los diferentes elementos necesarios para la realización de pruebas. Por lo tanto, se abordarán las siguientes áreas de aprendizaje: tipos de pruebas funcionales y no funcionales, pruebas ágiles (“Agile Testing”) y la elaboración de informes de resultados.

Es importante resaltar que, en el desarrollo de aplicaciones web, uno de los principales desafíos es asegurar que la calidad del producto sea óptima y libre de errores. En este contexto, las pruebas de “software” constituyen un componente fundamental del proceso de aseguramiento de la calidad y la adopción de buenas prácticas.

Las pruebas de “software” engloban el conjunto de actividades realizadas para identificar posibles fallos de funcionamiento, configuración o usabilidad en programas o aplicaciones, mediante la evaluación de su comportamiento. La complejidad e interoperabilidad de los sistemas informáticos, programas y aplicaciones han aumentado significativamente, lo que a su vez ha incrementado las posibilidades de defectos.

Para aplicar pruebas de “software” eficazmente, se deben estudiar los diferentes elementos que las componen. Así, se profundizará en los tipos de pruebas funcionales y no funcionales, las pruebas ágiles y la elaboración de informes de resultados. Cabe destacar que, en el contexto del desarrollo de aplicaciones web, verificar la óptima calidad del producto sin errores es esencial, haciendo que las pruebas de “software” sean un pilar fundamental en el proceso de desarrollo.

1. Realización de pruebas de “Software”

Para realizar las pruebas de “software”, es necesario establecer qué tipos de pruebas serán requeridas por el producto. Esto se debe a que no es necesario utilizar todos los tipos de pruebas existentes, sino más bien aquellas que sean necesarias para cada caso específico. Por esta razón, es crucial que el QA o el líder de pruebas determine qué tipos de prueba se adaptan mejor al proyecto en evaluación. 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 principalmente en dos categorías: pruebas funcionales y pruebas no funcionales. Considerando los diversos aspectos y componentes de un producto de “software”, se requerirán distintos tipos de pruebas, como pruebas unitarias, pruebas de integración, pruebas de estrés, pruebas de rendimiento, pruebas de escalabilidad, entre otras. Cada uno de estos tipos de pruebas determina la calidad de la aplicación, desde el código fuente hasta la interacción y experiencia del usuario.

Pruebas y calidad de “software”

Les invitamos a explorar el video y descubrir los diferentes tipos de pruebas y enriquecer su conocimiento en esta área clave del desarrollo de “software”.
https://www.youtube.com/watch?v=TLs-a1Cok-w

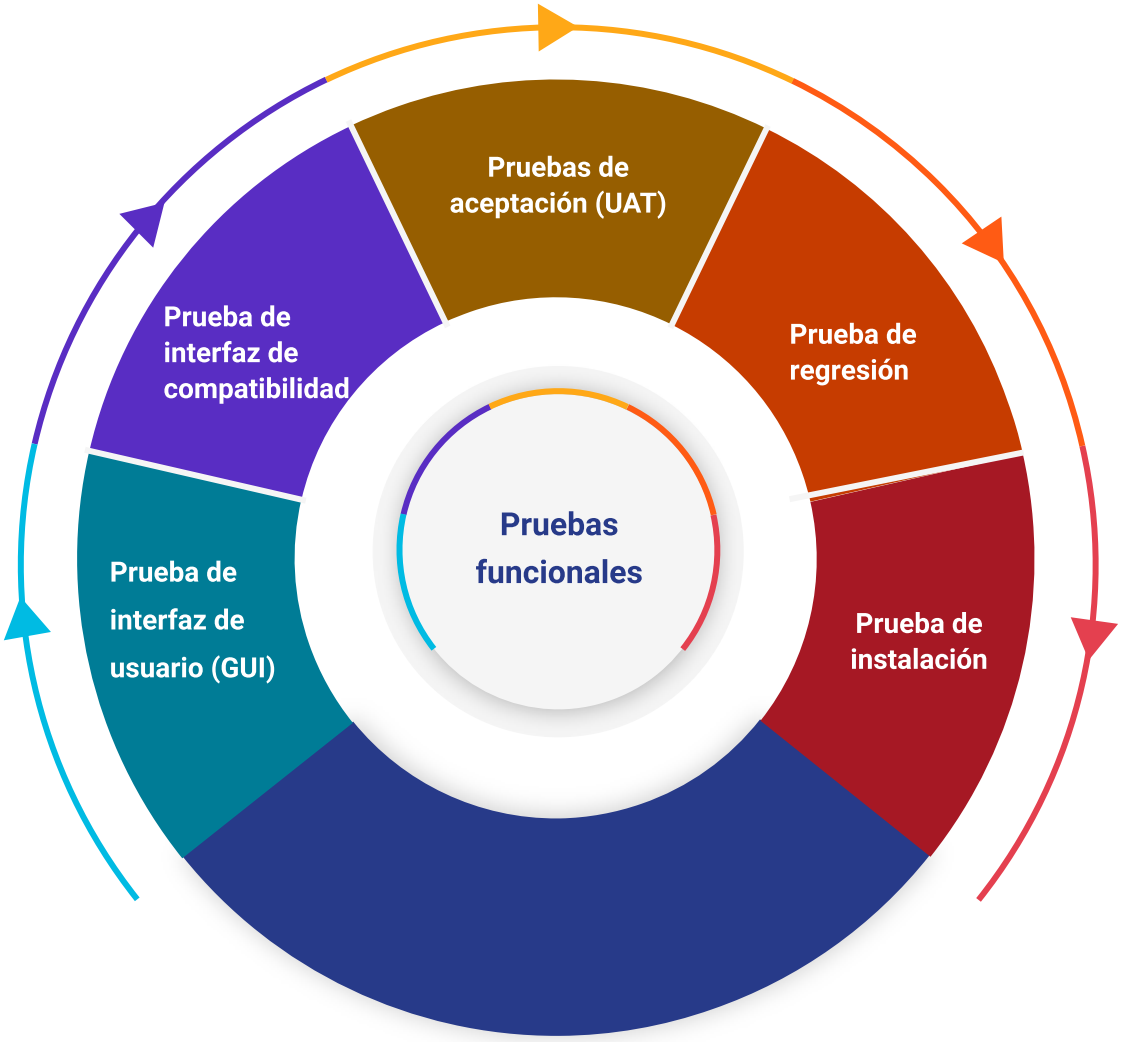
Algunos de los tipos de pruebas más comunes se describen en los siguientes puntos.

Pruebas funcionales

Se realizan para verificar las características cruciales para el negocio, incluyendo la funcionalidad y la usabilidad del “software”. Estas pruebas aseguran que las funcionalidades y características del “software” funcionen y se comporten conforme a lo esperado, sin inconvenientes. Validan la aplicación completa de acuerdo con las especificaciones detalladas en el Documento de Requisitos del “Software” (SRS, por sus siglas en inglés). Dentro de este conjunto de 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.

Figura 1. Tipos de pruebas funcionales



Pruebas unitarias

Estas pruebas se centran en validar elementos como componentes, unidades o piezas de forma individual e independiente dentro de un producto de "software", siguiendo el principio del Ciclo de Vida del Desarrollo de "Software" (SDLC, por sus siglas en inglés). Cualquier bloque de código, ya sea un procedimiento, función, método, constructor, clase o módulo, puede ser considerado una unidad y, por lo tanto, debe ser objeto de pruebas unitarias. Esto permite determinar su comportamiento y

validar si los resultados obtenidos se alinean con los esperados. Las pruebas unitarias deben ser las primeras en realizarse durante la fase de desarrollo y deben ser implementadas por los desarrolladores.

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

Generalmente, se llevan a cabo durante la fase de desarrollo de las aplicaciones o "software". Esta tarea es ejecutada por los programadores, aunque los aseguradores de calidad también pueden participar en su realización. Las razones para desarrollar este tipo de pruebas incluyen:

- Son rápidas de ejecutar, lo que permite realizarlas en gran cantidad.
- Funcionan como documentación de apoyo para el proyecto.
- Demuestran que la lógica del código funciona correctamente en todos los casos.
- Facilitan a los programadores el entendimiento profundo del código base, permitiendo realizar cambios de manera oportuna.
- Contribuyen a obtener un código de alta calidad.

En algunos casos, las pruebas pueden realizarse de manera manual, pero es preferible utilizar herramientas que permitan ejecutar el servicio de la mejor manera posible. Existen muchas herramientas disponibles en el mercado, y estas varían en función del lenguaje de programación que se esté utilizando. A continuación, mencionaremos algunas de las más conocidas:

- **XUnit.**

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

- **Nunit.**

  Es una herramienta de pruebas de plataformas .NET.

- **Junit.**

  Es una herramienta de pruebas de aplicaciones Java.

- **PhpUnit.**

  Es una herramienta de pruebas de programación PHP.

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

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

Pruebas de integración

Estas pruebas se centran en validar distintos módulos o componentes de un producto de "software" de tal forma que se verifique su funcionamiento como grupo funcional. Productos como aplicaciones de "software", sistemas de información o aplicaciones web están compuestos por diversos componentes, módulos y submódulos distribuidos en distintos niveles y capas, que colaboran y se coordinan para habilitar diferentes funcionalidades.

El objetivo de estas pruebas es asegurar que la integración entre estos componentes sea adecuada y detectar 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 con lo anterior, es importante tener en cuenta los siguientes objetos típicos de prueba:

- Bases de datos de subsistemas.
- Datos de configuración.
- Configuración del sistema.
- Infraestructura.
- Interfaces.

Pruebas no funcionales

Estas pruebas, similares a las pruebas funcionales, se diferencian en que se realizan bajo condiciones de carga de rendimiento para evaluar aspectos como la fiabilidad, usabilidad, escalabilidad, instalación, confiabilidad y seguridad del “software”. Generalmente, se llevan a cabo mediante 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.

Algunos aspectos, como el rendimiento de una aplicación, pueden ser complejos; sin embargo, esta prueba permite evaluar la calidad del “software” a probar. La calidad se basa principalmente en factores como el tiempo, la precisión, la estabilidad, la corrección y la durabilidad del producto bajo diversas circunstancias adversas. A partir de estos criterios de calidad, es posible explorar distintos tipos de pruebas no funcionales, tales como:

- **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 estrés**

  Nos permite saber cuántos usuarios o peticiones puede soportar el programa, con esta prueba determinamos si el sistema es estable.
- **Pruebas de volumen**

  Consiste en probar el funcionamiento del programa con ciertos volúmenes de datos.
- **Pruebas de usabilidad**

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

- **Pruebas de robustez**

  Son las encargadas de verificar la capacidad del programa para soportar entradas o instrucciones incorrectas. Una de sus características principales es implementar mecanismos de “software” de detección de errores en el “hardware”.

Pruebas de rendimiento

Las pruebas de rendimiento son un tipo de pruebas no funcionales realizadas con el propósito de determinar la estabilidad, escalabilidad y velocidad de una funcionalidad específica en el producto de “software”. El objetivo de estas pruebas es evaluar el comportamiento de la aplicación considerando diferentes parámetros de referencia del sistema y de la red, como la capacidad de la CPU, la velocidad de renderizado o carga de páginas, el manejo del tráfico, la concurrencia, el consumo de recursos del servidor, entre otros. Dentro de esta categoría, podemos distinguir las pruebas de carga y las pruebas de estrés.

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.

Ventajas y desventajas de los tipos de pruebas.

Aplicar los distintos tipos de pruebas puede asegurar que el "software" funcione conforme a lo esperado en todo momento, garantizando así su calidad. Por lo tanto, es esencial, para su aplicación, identificar las ventajas y desventajas de cada uno, con el fin de determinar las diferencias entre ellos. Podemos observar las ventajas de estos tipos de pruebas en el siguiente recurso.

Pruebas de rendimiento

- Evalúa la velocidad y escalabilidad de la aplicación.
- Determina los cuellos de botella para las mejoras de rendimiento.
- Detecta errores que se pasan por alto en las pruebas funcionales.
- Mejoras de características y optimización del sistema.
- Garantiza la fiabilidad de la aplicación web bajo una gran carga.

Pruebas funcionales

- Se asegura de que la aplicación esté libre de defectos.
- Garantiza el comportamiento requerido de las funcionalidades.

- Verifica que la arquitectura sea la correcta con la seguridad requerida.
- Mejora las funcionalidades generales y la calidad.

Pruebas de integración

- Asegura de que en la aplicación todos los módulos se encuentren bien integrados y funcionen juntos según lo requerido.
- Detecta conflictos y problemas interconectados para solucionarlos antes de crear un problema más grande.
- Verifica la fiabilidad, estabilidad y funcionalidad entre módulos diferentes.

Pruebas unitarias

- Detecta errores en las nuevas funcionalidades desarrolladas de manera temprana.
- Detecta problemas desde el inicio, esto permite reducir los costos de las pruebas.
- Con una mejor refactorización del código mejora la calidad del código.
- Permite una buena documentación y simplifica la integración.

Desventajas de los tipos de pruebas

Todos los tipos de pruebas mejoran la experiencia del usuario, las funcionalidades y garantizan la calidad del "software". Esto significa que, más que desventajas, lo que encontramos son ventajas en su aplicación. Sin embargo, el tiempo y el costo asociados a la realización de pruebas de "software" podrían considerarse como desventajas. Llevar a cabo pruebas demanda recursos y esfuerzos, y existe el

riesgo de obtener resultados inexactos. No obstante, omitir las pruebas en las aplicaciones podría acarrear problemas serios al producto.

Documentos de caso de pruebas

Los productos derivados del proceso de desarrollo de "software" son conocidos como artefactos; entre ellos se incluyen el código fuente, los defectos identificados, el plan de pruebas, los casos de prueba y los "scripts" de prueba. El aseguramiento de la calidad del "software" (SQA, por sus siglas en inglés) involucra artefactos específicos que se generan en las distintas fases del proceso de SQA, los cuales se detallan a continuación:

- **Planificación**

  En esta etapa se realiza el plan de pruebas como documento que describe un enfoque sistemático para probar un sistema, así como el informe de resultados de pruebas que organiza y presenta un análisis resumido de los resultados de las pruebas para su auditoría y evaluación.

- **Construcción**

  En esta etapa se encuentran los casos y "script" de prueba como conjunto de condiciones con las cuales un "tester" debe determinar si un sistema funciona según lo requerido.

- **Ejecución**

  En esta etapa se obtiene el resultado de la ejecución de pruebas, recopilando la información relacionada con los resultados de la ejecución de cada "script" de prueba.

En el diseño de las pruebas de “software”, es crucial identificar y describir los casos de prueba. Para documentar un caso de prueba, es necesario comprender que este incluye un conjunto de variables o condiciones bajo las cuales un probador determina si un sistema funciona de acuerdo con lo esperado. Asimismo, es importante destacar que la documentación de un caso de prueba se convierte en una tarea manejable si se dispone de la información necesaria para su elaboración. Al momento de probar un “software”, la documentación de casos de prueba es de gran utilidad, ya que se transforma en una herramienta esencial en el proceso de registro, seguimiento y control.

A continuación, se nombran los elementos principales que debe tener un caso de prueba.

Tabla 1. Elementos de un caso de prueba

Elemento	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 qué probará, en ciertas ocasiones se incluye el ambiente de pruebas.
Número de orden Ejecución	Orden en el 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 el cual se debe encontrar el sistema antes de comenzar la prueba.

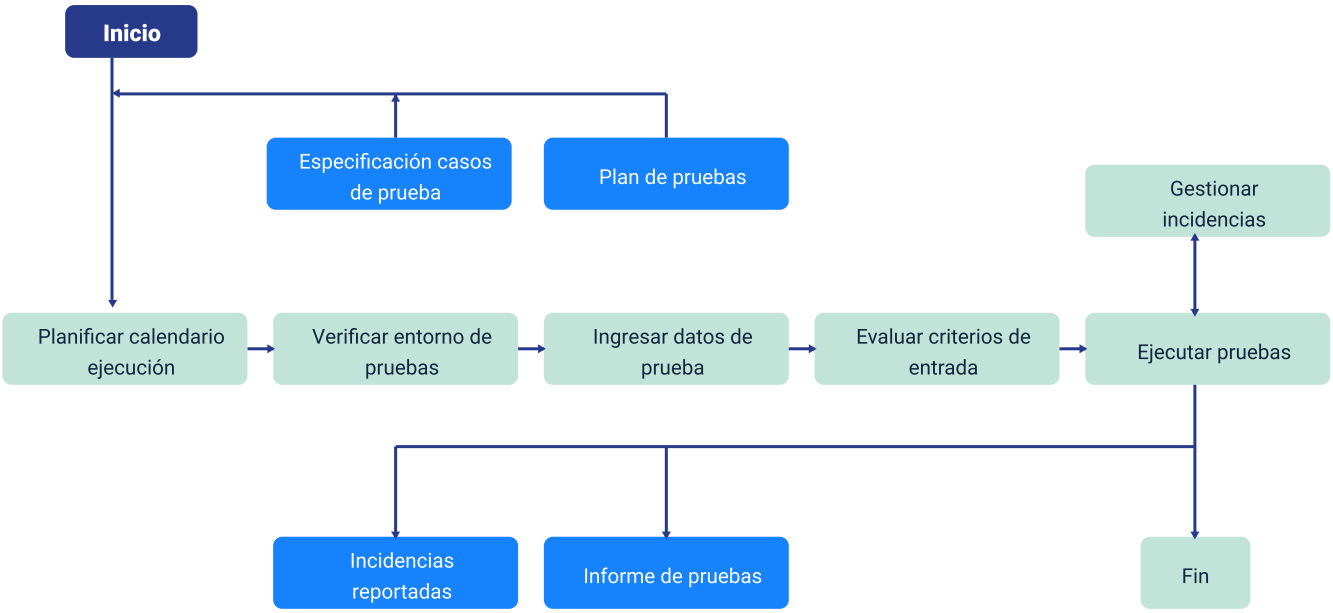
Elemento	Caso de prueba
Postcondición	El estado en que debe encontrarse el sistema luego de ejecutar la prueba.
Resultado esperado	Objetivo que debe ser alcanzado posterior a ejecutarse la prueba.

Identificar, describir y documentar casos de prueba facilita el trabajo cuando es necesario evaluar y comparar las modificaciones a los requerimientos inicialmente definidos por el cliente a lo largo del tiempo. Este seguimiento permite verificar que los resultados obtenidos estén acordes con lo requerido o solicitado, logrando así la satisfacción del cliente. Esto convierte a los casos de prueba en un aliado principal al momento de garantizar la calidad del "software".

Ejecución de las pruebas

La ejecución de pruebas comienza con la creación de los datos de prueba necesarios para llevar a cabo los casos de prueba establecidos. Esta ejecución puede ser tanto automatizada como manual; en ambos casos, es fundamental registrar y documentar los fallos detectados en el sistema, así como sus correcciones posteriores. En la siguiente figura se ilustra el flujo de actividades relacionadas con la ejecución de pruebas.

Figura 2. Diagrama Actividades



El diagrama de actividades mencionado anteriormente detalla el flujo que debe considerarse al momento de ejecutar las pruebas de "software". Por esta razón, a continuación se describen cada una de las actividades implicadas.

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 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.

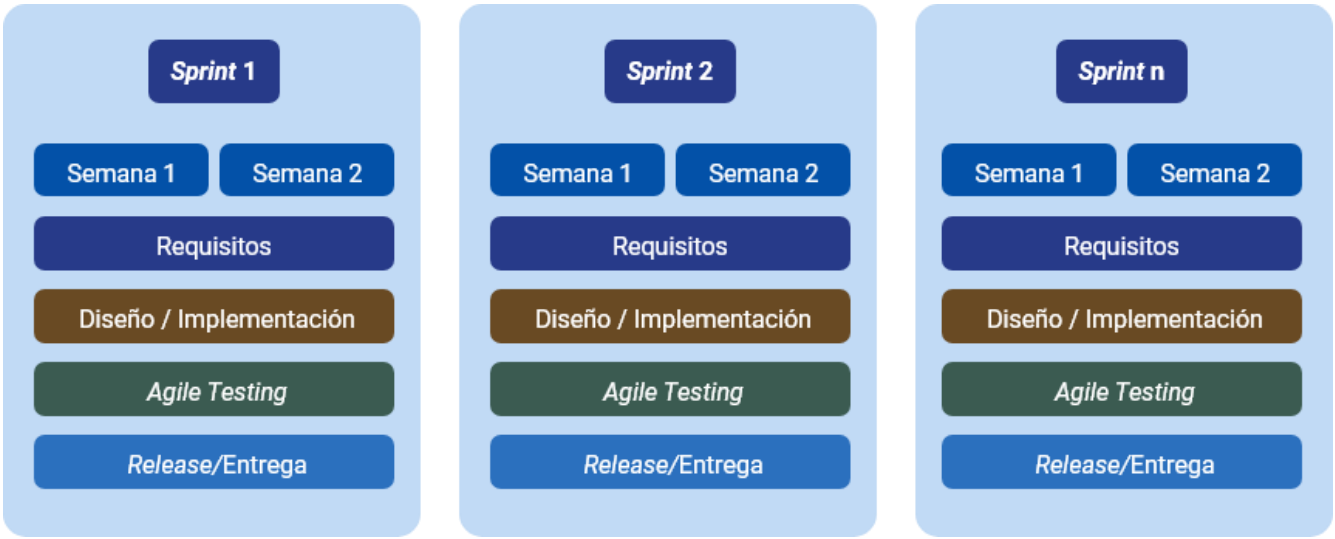
1.2. "Agile Testing"

Las metodologías ágiles proporcionan un conjunto de directrices y principios diseñados para agilizar y priorizar la entrega de productos sobre la documentación exhaustiva, simplificando los procesos e involucrando al cliente final desde las etapas iniciales del proyecto. Es importante destacar que una metodología ágil se compone de iteraciones o "sprints", cada uno de duración relativamente corta, generalmente de dos a cuatro semanas. Al finalizar cada iteración, se disponen de funcionalidades del "software" que pueden ser entregadas y utilizadas por el cliente.

Recordando las metodologías ágiles, se puede especificar que las pruebas ágiles ("Agile Testing") son una práctica que se lleva a cabo en cada iteración. Existen dos tipos de planificaciones: la planificación de lanzamiento/entrega, que se refiere a las entregas frecuentes al cliente, es decir, la puesta en producción de nuevas versiones del

producto; y la planificación de “sprint”/iteración, que indica que un proyecto se ejecuta en bloques temporales cortos y fijos. Sin embargo, estas planificaciones pueden coincidir en un mismo momento, es decir, puede haber una sola planificación que incluya actividades tanto de lanzamiento como de “sprint”, tal como se muestra en la siguiente figura:

Figura 3. Actividades “Sprints Release”



Por otro lado, es importante considerar que la metodología de desarrollo de “software” tradicional, conocida como cascada, emplea diferentes tipos de pruebas de “software” que evalúan tanto los requerimientos funcionales como los no funcionales de manera secuencial. Sin embargo, en las metodologías ágiles, y específicamente en las pruebas ágiles, también se contemplan diversos tipos de pruebas. La diferencia radica en que en las metodologías ágiles se utilizan “sprints” o iteraciones cortas y hay una interacción continua entre los equipos de desarrollo, diseño y pruebas.

Las pruebas ágiles se basan en principios que se alinean con el desarrollo ágil de "software". Además, existen ciertos cuadrantes que se deben tener en cuenta al momento de planificarlas.

A continuación, se describen tanto los principios como los cuadrantes:

Principios de las pruebas ágiles ("Agile Testing")

El "Agile Testing" se fundamenta en los siguientes principios, aplicables a un desarrollo de "software" exitoso:

- **Integración continua**

  Los equipos de diseño y desarrollo ejecutan pruebas continuamente, puesto que es una forma de asegurar el avance continuo del producto.

- **Entrega continua**

  Se soporta en la integración continua, en este punto los desarrolladores realizan un control y la cantidad de veces que se realiza la entrega continua.

- **Elaborar menos documentación**

  Los participantes de los equipos en metodologías ágiles pueden crear listas y tomar notas para centrarse en probar el "software", lo que permite no enfocarse en detalles secundarios que tomarían mucho tiempo.

- **Responder con velocidad a la retroalimentación**

  Los involucrados en el negocio del producto están estrechamente conectados con las interacciones del producto, además el tiempo de respuesta se reduce debido a la realimentación continua en cada parte de la interacción.

- **Proporcionar retroalimentación constante**

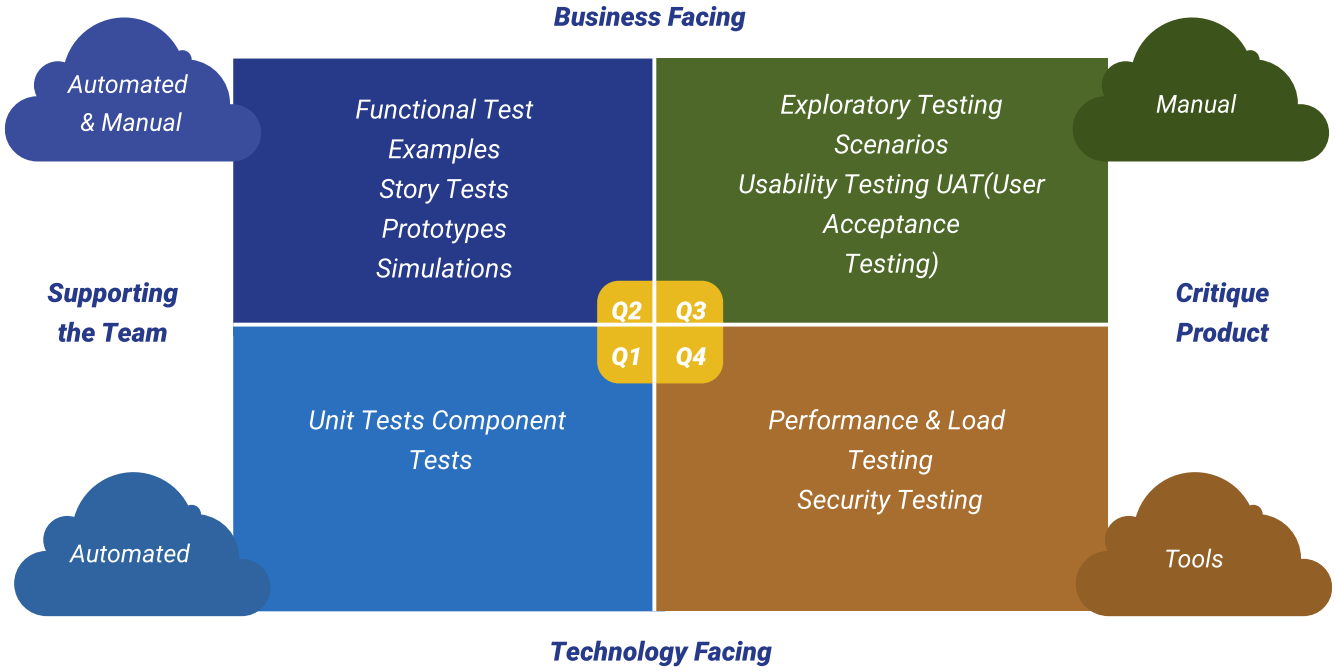
  En las pruebas ágiles la retroalimentación es continua, de modo que el diseño del producto logre los propósitos del negocio requeridos.

Cuadrantes de las pruebas ágiles (“Agile Testing”)

Para planificar las pruebas ágiles, podemos basarnos en los cuadrantes de “Agile Testing”, que proporcionan una biblioteca de tipos de pruebas aplicables para satisfacer los requerimientos. Estos cuatro cuadrantes ofrecen una clasificación útil para la planificación de las pruebas ágiles, asegurando que se consideren los métodos y recursos necesarios para lograr productos de “software” de alta calidad.

Las pruebas definidas por los cuadrantes se centran en brindar apoyo al equipo de desarrollo a lo largo del proceso de creación del producto, ya que guían el desarrollo de la funcionalidad. Posteriormente, facilitan la introducción de nuevo código y la refactorización, asegurando que no se produzcan resultados inesperados en el comportamiento del sistema. En la siguiente figura, se pueden observar los cuadrantes de las pruebas ágiles, los cuales se organizan en torno a dos ejes principales: las pruebas que apoyan al equipo y las tecnologías que facilitan el proceso.

Figura 4. Cuadrantes pruebas ágiles



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

- **Pruebas de cara al negocio (“Business Facing”)**

  Son pruebas del Q2 y Q3, pruebas funcionales y pruebas UAT, se enfoca en realizar pruebas manuales y criticando al producto con las pruebas de aceptación.

- **Pruebas de cara a la tecnología (“Technology Facing”)**

  Pueden ser automatizadas y utilizando herramientas, incluye el Q1 y Q4.

- **Pruebas soportando la programación o soporte del equipo (“Supporting the Team”)**

  Comprenden pruebas automatizadas y manuales del cuadrante Q2 y Q1 donde se realizan pruebas unitarias y de componentes y pruebas funcionales con prototipos y simuladores.

- **Pruebas de críticas al producto (“Critique the Product”)**

  Estas pruebas pueden tener un carácter positivo, puesto que a partir de ellas se pueden sugerir o realizar mejoras. Cuando se ejecutan estas pruebas, la idea es simular lo más que se pueda un ambiente real en el cual serán ejecutadas, así mismo estas pruebas frecuentemente son realizadas por los usuarios finales como pruebas de aceptación (UAT).

- **El primer cuadrante (Q1)**

  El primer cuadrante (Q1) contiene los siguientes tipos de pruebas:

  - Pruebas unitarias.
  - Pruebas de componente.

- **El segundo cuadrante (Q2)**

  El segundo cuadrante (Q2) definen las funcionalidades que el cliente solicita:

  - Pruebas funcionales.
  - Pruebas de historias.
  - Pruebas de Prototipos y simulaciones.

- **El tercer cuadrante (Q3)**

  El tercer cuadrante (Q3) contiene los siguientes tipos de pruebas:

  - Pruebas exploratorias.
  - Pruebas de usabilidad.
  - Pruebas de escenarios.
  - Pruebas de aceptación de usuario.

- **El cuarto cuadrante (Q4)**

El cuarto cuadrante (Q4) contiene pruebas técnicas. Relacionadas con los requerimientos no funcionales, por lo que validan el cumplimiento de estos, analizando la seguridad, el desempeño y la robustez.

- Pruebas de rendimiento y de estrés.
- Pruebas de seguridad.
- Pruebas de robustez.

Ahora, se describen algunas recomendaciones para tener en cuenta al momento de trabajar con pruebas de "software" ágiles ("Agile Testing").

- **Gestión de pruebas por "sprint"**
  - Integrar pruebas unitarias, funcionales y de carga en la planificación desde el principio.
  - Identificar y rectificar problemas de forma temprana, antes que se conviertan en problemas complejos que puedan influir el proyecto.
- **Integración de analistas en desarrollo**
  - Los "testers" tienen el conocimiento para asegurar que se entregue un producto de calidad y que cumpla los requerimientos del cliente.
  - Al integrar a los "testers" al equipo de desarrollo se logra compartir el conocimiento.
- **Revisión de pruebas unitarias por "testers"**
  - Los desarrolladores suelen deliberar de forma distinta que los "testers".
  - Para mejorar las pruebas unitarias, se debe permitir que los analistas de pruebas revisen las pruebas unitarias y realicen sugerencias.
- **Pruebas de carga diarias**

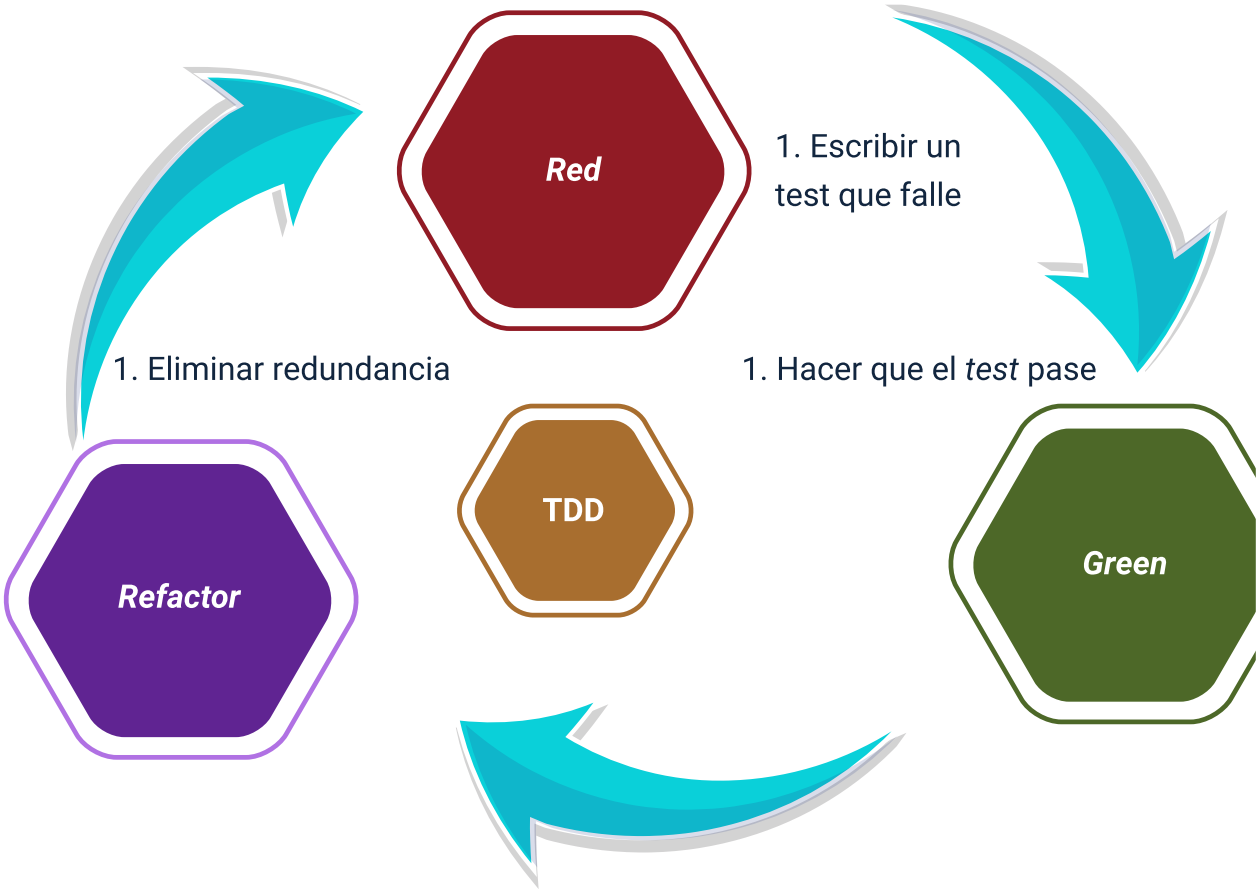
- Las pruebas de estrés son cada vez más críticas para el éxito, considerando que el “software” de hoy se diseña para implementaciones masivas que puedan escalar.
- Deben incorporarse pruebas de estrés en todas las construcciones de “software” a diario.

- **Automatización de pruebas de “software”**
  - Hacer uso de herramientas de automatización de pruebas. De esta forma será más eficiente y eliminará inexactitudes.
  - La automatización de pruebas permitirá ejecutar con mayor eficiencia las pruebas de regresión cada vez que se incorpore actualizaciones.

“Test Driven Development” (TDD)

Es una técnica de desarrollo que permite crear pruebas antes del código de producción, es decir, antes de la implementación final. La esencia del TDD (Desarrollo Guiado por Pruebas) consiste en elaborar una prueba antes de proceder con la implementación, de tal manera que el código desarrollado permita que la prueba se ejecute satisfactoriamente. Además, esta técnica se complementa con la refactorización. Generalmente, se tienen requerimientos que se expresan en historias de usuario como criterios de aceptación o pueden ser descritos como casos de uso. Posteriormente, estos requerimientos se traducen en pruebas automatizadas que, a su vez, orientan el desarrollo del código siguiendo buenas prácticas.

Figura 5. Flujo del TDD



El Desarrollo Guiado por Pruebas (TDD, por sus siglas en inglés) se fundamenta en tres pasos esenciales:

Red: este paso se denomina así porque implica la creación de una prueba que inicialmente falla, lo cual se refleja con letras rojas en la consola al ejecutarla.

Green: el objetivo de este paso es modificar el código para que la prueba anteriormente fallida ahora pase exitosamente. Se llama Green porque, al pasar la prueba, los resultados suelen mostrarse en letras verdes en la consola.

Refactor: una vez que la prueba ha sido superada, se procede a revisar y mejorar el código para optimizarlo, limpiarlo o aplicar mejores prácticas, sin alterar su funcionalidad.

En la siguiente tabla podemos verificar cada uno de los pasos guiados por las pruebas TDD donde se explica cada uno de los pasos con sus ejemplos.

Tabla 2. Flujo del TDD

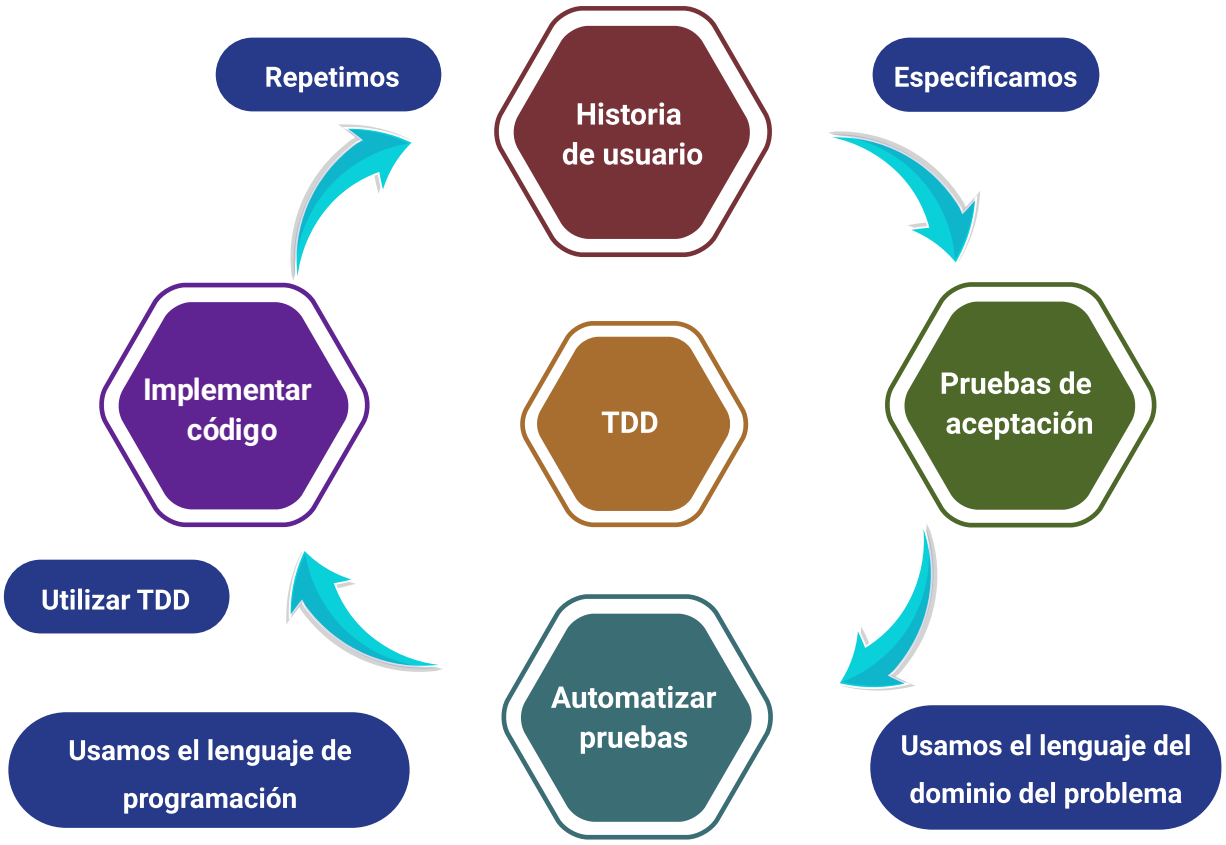
Paso	Pasos	Ejemplo
Red	1. Lo primero es que el cliente escriba una historia de usuario.	Supóngase que el cliente solicita que se desarrolle una aplicación con operaciones matemáticas sencillas que sume y reste.
Red	2. Los criterios de aceptación de esta historia se escriben con el cliente, deben estar 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.
Red	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)); }
Red	4. Se verifica que la prueba falla.	RED 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; } }

Paso	Pasos	Ejemplo
Green	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.
Refactor	8. Se refactoriza y se limpia el código.	Es necesario pasar todos los “test” después de refactorizar.

“Acceptance Test Driven Development” (ATDD)

El Desarrollo Guiado por Pruebas de Aceptación (ATDD, por sus siglas en inglés) es una metodología de desarrollo que enfatiza la comunicación entre los clientes empresariales, los desarrolladores y los “tester”. Antes de que los desarrolladores empiecen a escribir código, ATDD incorpora las pruebas de aceptación, poniendo especial énfasis en la redacción de estas pruebas. El proceso de ATDD se desarrolla a través de los siguientes pasos:

Figura 6. ATDD



- **Seleccionar la historia de usuario:** en este punto, se asume que se ha realizado una priorización para determinar cuál es la próxima historia de usuario a abordar.
- **Escribir la prueba de aceptación:** en esta etapa, todos los involucrados deben participar de manera colaborativa para definir las pruebas que validen la correcta implementación de la historia.
- **Implementar la historia de usuario - Automatizar las pruebas:** existen varias maneras de llevar a cabo este paso, pero lo crucial es que, a partir de las pruebas de aceptación definidas anteriormente, se obtengan pruebas ejecutables. Estas pruebas permitirán verificar el progreso de la implementación de la funcionalidad.

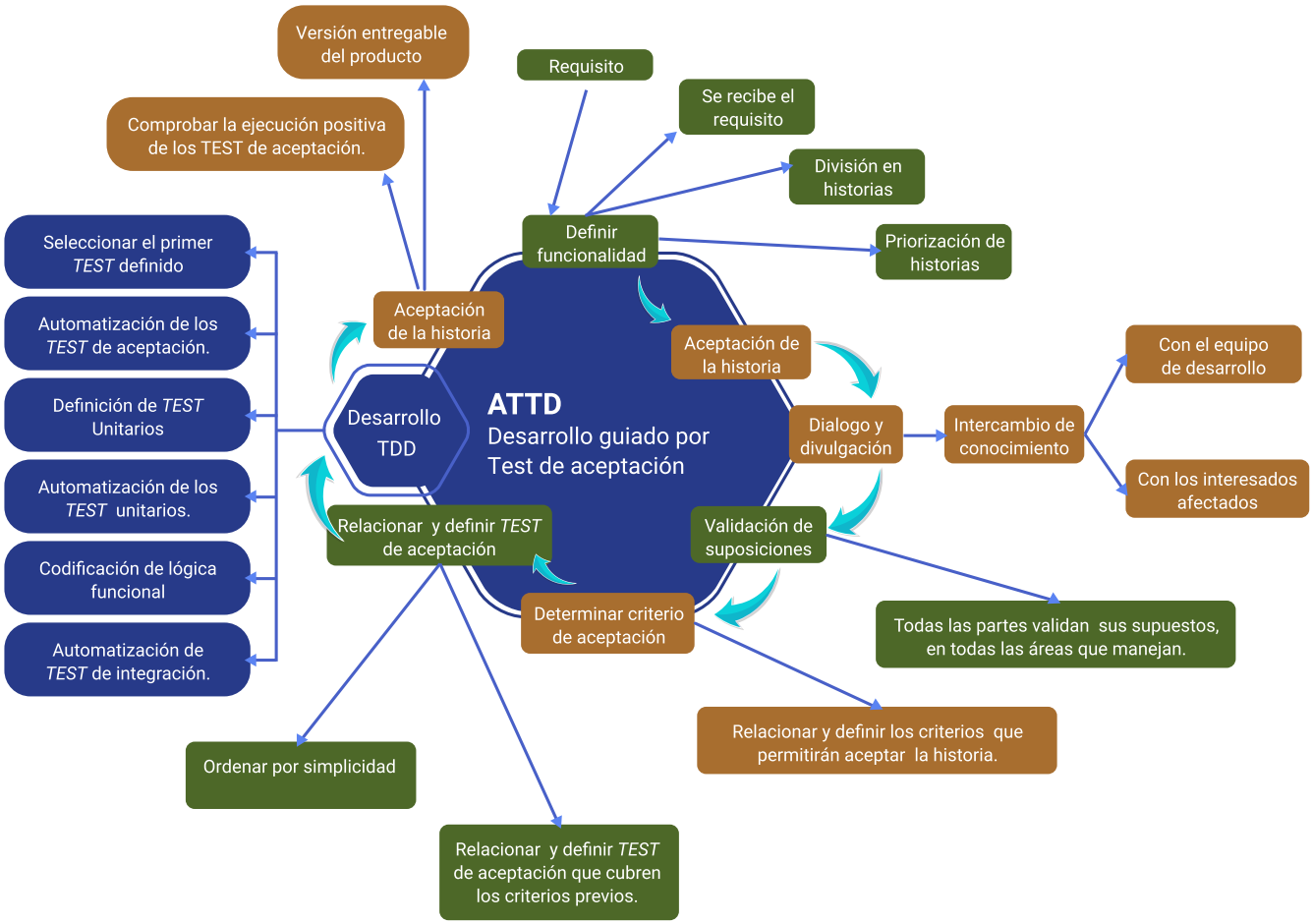
- **Implementar el código:** aunque hay diversas formas de realizar este paso, una de ellas es mediante el uso de TDD (Desarrollo Guiado por Pruebas), que asegura que el código cumpla con los requisitos establecidos en las pruebas de aceptación.
- **Realizar pequeños cambios/Refactorización:** este paso es fundamental tanto en ATDD como en TDD. Se trata de una práctica iterativa que concluye cuando los requerimientos del cliente han sido plenamente satisfechos, optimizando y limpiando el código sin alterar su funcionalidad.

De acuerdo con lo expuesto anteriormente, se inicia desde un desarrollo al cual los programadores han implementado TDD. Si este proceso se ha llevado a cabo de manera adecuada, deberían haber logrado un código limpio y refactorizado, que además ha sido validado mediante pruebas unitarias. Así, el código resultante es de alta calidad; sin embargo, surge la pregunta: ¿ocurre lo mismo con la aplicación en su totalidad?

ATDD no constituye simplemente una técnica de desarrollo o una buena práctica, sino que representa un enfoque más amplio. Se trata de una metodología de trabajo diseñada para el desarrollo de "software", la cual tiene como objetivo verificar si los procedimientos adoptados durante el desarrollo son los más adecuados para alcanzar los objetivos deseados. Mientras que TDD se enfoca únicamente en confirmar que lo desarrollado esté codificado de forma correcta y libre de errores, lo cual representa una visión limitada centrada exclusivamente en la calidad del código y no en la aplicación como un todo.

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

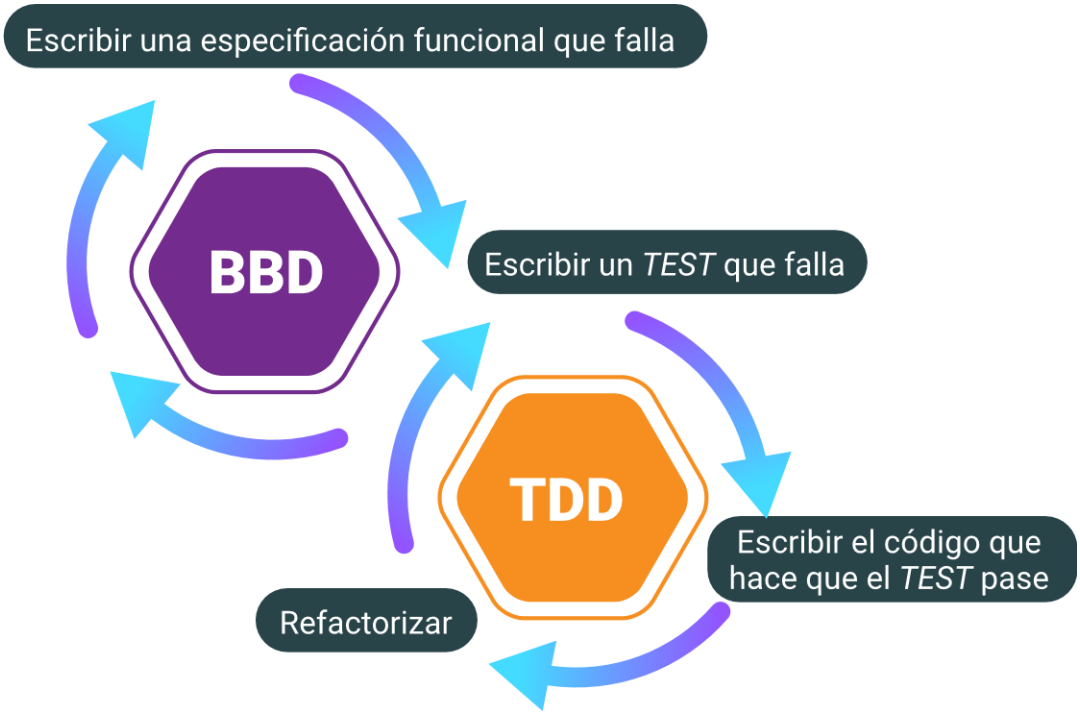
Figura 7. ATDD



"Behaviour Driven Development" (BDD)

El Desarrollo Dirigido por Comportamiento (BDD, por sus siglas en inglés) fue propuesto inicialmente por Dan North como una síntesis y refinamiento de prácticas en ingeniería de "software", destinadas a asistir a los equipos en la generación y entrega rápida de "software" de mayor calidad. El proceso de BDD es similar al de TDD y sigue los siguientes pasos:

Figura 8. Imagen BDD



a) Escribir un escenario.
b) Ejecutar el escenario que falla.
c) Escribir la prueba que corresponde a las especificaciones del escenario.
d) Escribir el código más simple para pasar la prueba y el escenario.
e) Refactorizar para eliminar la duplicación.

BDD, basado en la técnica de TDD, aborda ciertas limitaciones de esta última. Un problema fundamental de TDD surge cuando el "software" supera correctamente una prueba pero no cumple con la funcionalidad esperada. Pasar una prueba no garantiza necesariamente que el "software" realice lo que el cliente anticipa.

Ejemplo:

- Inicialmente, se define la característica o funcionalidad que se desea implementar, basándose en una historia de usuario.

Característica:

- Un usuario desea acceder a una plataforma web universitaria para descargar sus apuntes.

A continuación, se presentan dos escenarios derivados de la historia de usuario especificada

El desarrollo dirigido por comportamiento (BDD)

El objetivo de BDD es que las historias de usuario dirijan el desarrollo del proyecto "software". Además, BDD permite comprobar que el "software" implementado cumple con la funcionalidad requerida.

- 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.

Primer escenario

- El usuario entra correctamente.
- Tiene un email y una contraseña.
- Verifica que el nombre de usuario existe.
- Verifica que la contraseña es correcta.

- El usuario puede acceder a su cuenta, su información y puede descargar sus apuntes.

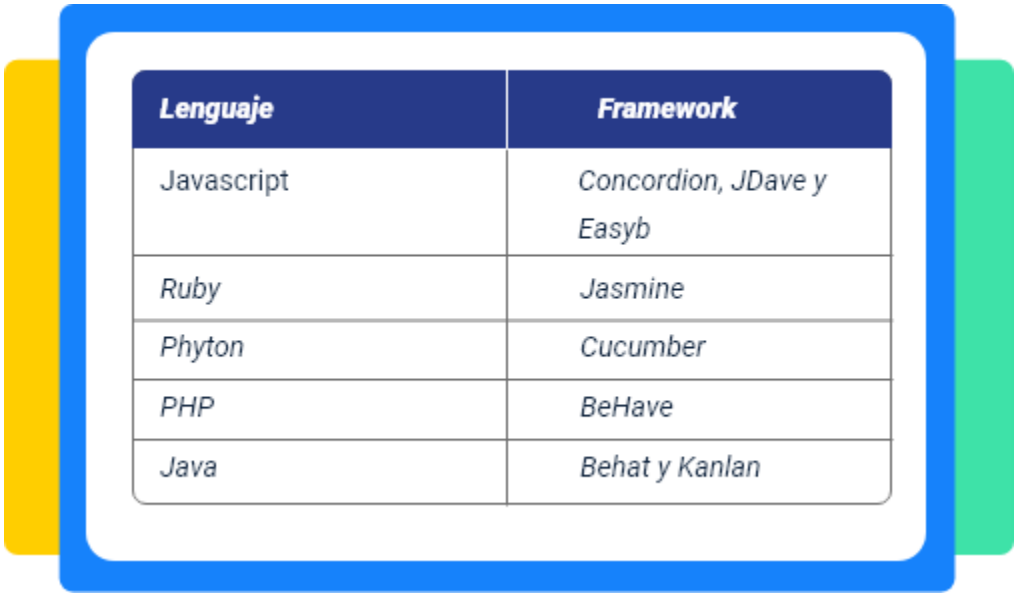
Segundo escenario

- El usuario no recuerda la contraseña.
- Tiene un email.
- Cuando el usuario hace clic en: "No recuerdo la contraseña".
- Luego el usuario inserta su email.
- Posteriormente el email existe.
- Se envía un correo de verificación.
- Y el usuario debe validar el enlace.
- Y el usuario debe insertar nueva contraseña.
- El usuario tiene que volver a entrar con su nueva contraseña.
- Se ejecuta primer escenario.

“Frameworks” de BDD por lenguaje

Una vez definida, BDD utiliza herramientas para convertir esta historia de usuario en código. Para ello se utilizan "contenedores" o “frameworks”; hay tantos contenedores como lenguajes de programación. Normalmente están disponibles para la mayoría de los lenguajes de programación. A continuación, se muestran algunos:

Figura 9. "Frameworks" de BDD por lenguaje



Lenguaje	Framework
Javascript	*Concordion, JDave y Easyb*
Ruby	*Jasmine*
Phyton	*Cucumber*
PHP	*BeHave*
Java	*Behat y Kanlan*

"Testing" exploratorio

Las pruebas exploratorias han captado significativamente la atención dentro de la comunidad de pruebas de "software". Este método de pruebas se caracteriza por una interacción simultánea entre el testeo, el aprendizaje y el diseño de nuevas pruebas.

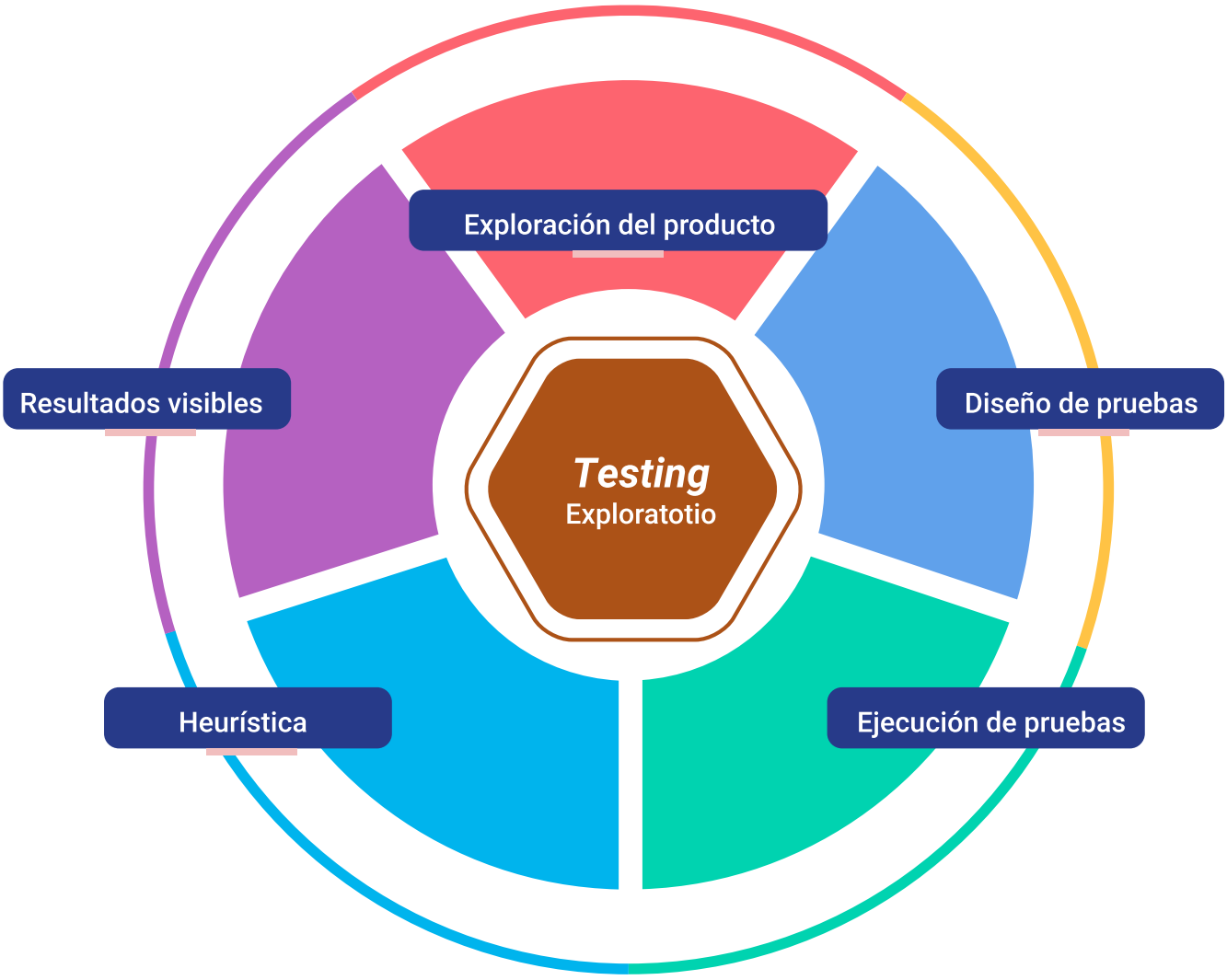
En la práctica, casi todos los "testers" realizan pruebas exploratorias, ya sea de manera consciente o no. Un ejemplo claro de esto son las pruebas de regresión de errores. Para confirmar la corrección de un error, el "tester" podría iniciar siguiendo los pasos detallados en el informe de errores. Sin embargo, una vez que el programa supera esta prueba inicial, es probable que el "tester" lleve a cabo pruebas adicionales para asegurarse de que el error ha sido completamente resuelto.

El "testing" exploratorio integra el diseño, la ejecución y el aprendizaje sobre la aplicación en prueba, fomentando un proceso de exploración y comprensión profunda

del producto. A diferencia de otros enfoques, no se basa en un guion o "script" de pruebas preestablecido, lo que lo hace particularmente valioso en metodologías ágiles.

Después de observar el proceso exploratorio del "software", se han identificado las siguientes pruebas exploratorias:

Figura 10. Pruebas exploratorias



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.

Automatización de pruebas de regresión

Las pruebas de regresión garantizan que los cambios realizados en una aplicación o sistema no introduzcan errores en las funcionalidades existentes. Sin embargo, realizadas de manera manual, estas pruebas pueden ser muy consumidoras de tiempo, llegando a ser más costosas que las pruebas de las nuevas funcionalidades que se están evaluando. Además, la intervención manual incrementa la susceptibilidad a errores debido al factor humano.

Estas pruebas se realizan siguiendo flujos completos, lo que implica ejecutarlas considerando todas las opciones disponibles según la lógica de negocio. Es crucial incluir en las pruebas las partes del código que han sido modificadas, para asegurar que los cambios no afecten negativamente la interacción con el entorno en el que el "software" opera.

Es esencial seleccionar cuidadosamente los flujos a probar, optimizando el proceso para evitar fallos costosos en producción. A pesar de que las pruebas de regresión pueden ser lentas, su automatización permite que sean tanto asertivas como rápidas. Además, la disponibilidad de "scripts" de pruebas facilita que cualquier persona, sin necesidad de un equipo de "testers" especializados, pueda ejecutarlas. Esto es particularmente útil para aplicaciones que han estado en producción durante largo tiempo, ya que reduce el riesgo asociado a la falta de conocimiento sobre los flujos de trabajo por parte del equipo.

Este tipo de pruebas, ya sean lanzadas por los desarrolladores o los "testers", son susceptibles de automatización. Para su implementación, se utilizan herramientas comunes a las pruebas unitarias y de sistemas. Sin embargo, también se dispone de:

Herramientas especializadas de código abierto diseñadas específicamente para facilitar y optimizar este proceso:

Winrunner, VTest, Canoo, AdventNet y QEngine.

Otras herramientas comerciales para aplicar este tipo de pruebas son:

HP UFT, Seapine QA Wizart Pro, IBM Rational Test Workbench, SmartBear Test Complete, SOASTA, Ranorex UI y OpKey.

Pruebas de exploratorias, usabilidad y aceptación

Las pruebas de usabilidad y las pruebas de aceptación del usuario, aunque pueden presentar semejanzas en términos de resultados y objetivos, difieren significativamente en su enfoque. Las pruebas de usabilidad se centran en identificar errores o deficiencias en el sistema que puedan afectar la experiencia del usuario, con

el objetivo de mejorar el producto. En contraste, las pruebas de aceptación del usuario buscan demostrar que el sistema cumple con los requisitos y expectativas establecidos, señalando así que el producto está listo para su lanzamiento. Por lo tanto, mientras las pruebas de usabilidad se enfocan en perfeccionar la interacción del usuario con el producto, las pruebas de aceptación del usuario se orientan a validar la completitud y adecuación funcional del sistema. Lo invitamos a explorar el siguiente video y conocer más sobre este tipo de pruebas:

Video 1. Pruebas de exploratorias, usabilidad y aceptación

Enlace de reproducción del video

Síntesis del video: Pruebas de exploratorias, usabilidad y aceptación
En el mundo del desarrollo de “software”, las pruebas juegan un papel crucial. Hoy exploraremos tres tipos fundamentales: pruebas exploratorias, de usabilidad y de aceptación. Pruebas exploratorias El “testing” exploratorio es un enfoque dinámico. Combina el aprendizaje, el diseño de pruebas y su ejecución de manera simultánea. Es una técnica flexible que permite a los “testers” adaptar su enfoque en tiempo real, basándose en los hallazgos obtenidos durante el proceso. Pruebas de usabilidad La usabilidad trasciende la estética, centrando su valor en la funcionalidad y experiencia del usuario. A través de las pruebas de usabilidad, evaluamos cómo los usuarios interactúan con el producto, enfocándonos en la facilidad de uso e intuitivita. Este método es indispensable para optimizar la experiencia del usuario antes de lanzar el producto al mercado. Pruebas de aceptación Las pruebas de aceptación se enfocan en el usuario final. Mediante tareas específicas, evaluamos si el sistema cumple con los requisitos y expectativas del usuario. Estas pruebas son vitales para asegurar que el “software” no solo funcione según lo previsto sino que también resuelva efectivamente las necesidades del usuario.

Las pruebas exploratorias, de usabilidad y de aceptación son esenciales en el ciclo de vida del desarrollo de "software". Garantizan que el producto final sea robusto, eficiente y, sobre todo, que satisfaga las necesidades y expectativas de los usuarios. Integrar estas pruebas desde las primeras etapas del desarrollo asegura la entrega de soluciones de "software" de alta calidad.

La excelencia en el desarrollo de "software" se logra a través de pruebas rigurosas y continuas. Invitamos a los profesionales del sector a adoptar estos enfoques para mejorar la calidad y la satisfacción del usuario en sus productos.

Pruebas de desempeño, carga y seguridad

Concluir la programación de una aplicación marca un momento de éxito; con el paso del tiempo, tener un producto terminado representa un logro significativo. Sin embargo, este hito también señala el momento crucial para probar diversos aspectos del "software", con el fin de asegurar su correcto funcionamiento y controlar su calidad. Factores como el desempeño, la seguridad y la capacidad de carga son elementos críticos que deben evaluarse en todo producto de "software" desarrollado, los cuales se describen a continuación.

Pruebas de desempeño

Las pruebas de desempeño evalúan cómo se comporta un sistema en términos de estabilidad y rapidez de respuesta bajo ciertas condiciones, como la carga de trabajo, la concurrencia, el ancho de banda, y las características específicas de equipos o servidores, tales como la RAM y la CPU. Además, estas pruebas permiten identificar, medir, investigar, validar o verificar otros aspectos de calidad, incluyendo la seguridad, la escalabilidad y el consumo de recursos. Constituyen un estándar en la ejecución,

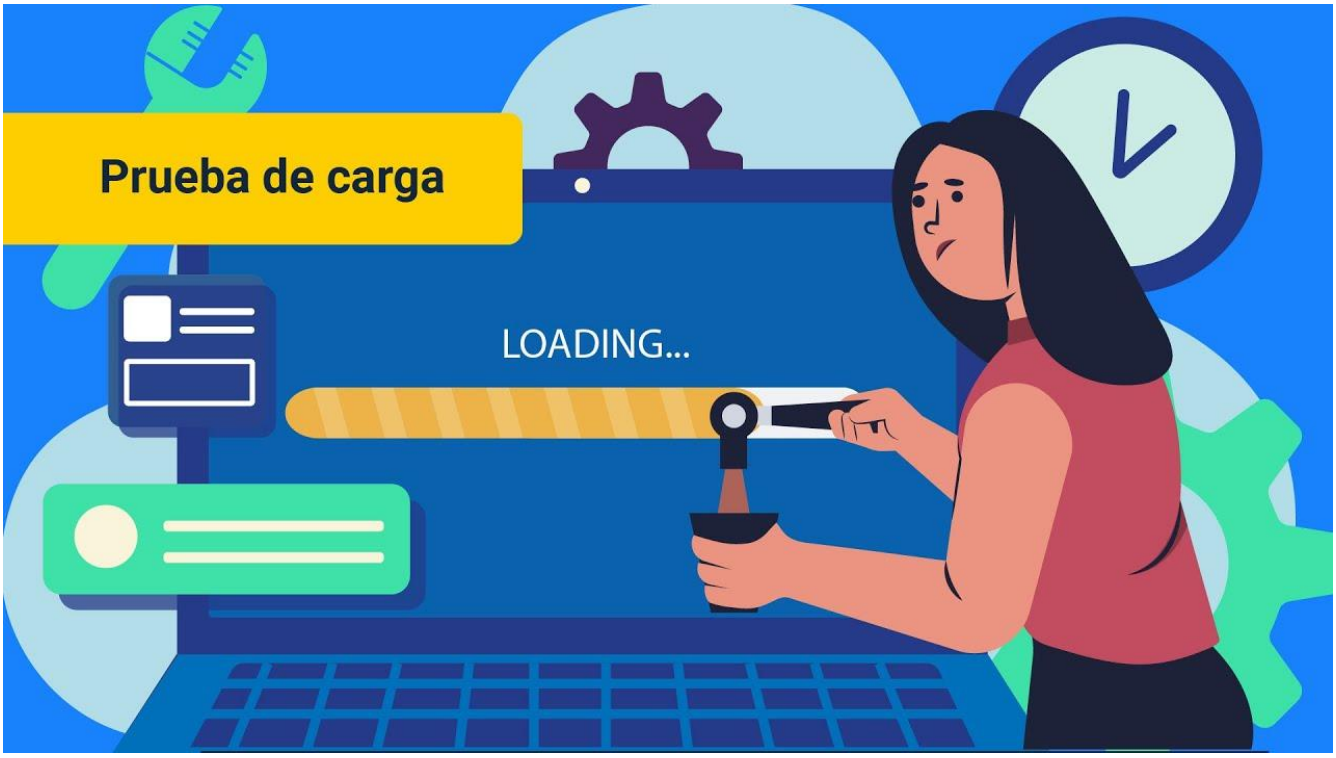
diseño y arquitectura de un producto de “software”. Entre las pruebas de desempeño más importantes se encuentra la prueba de carga, la cual se detallará a continuación.

Pruebas de carga

Las pruebas de carga se enfocan en determinar o validar las características de rendimiento de un sistema o aplicación bajo prueba, al enfrentarse a distintos volúmenes de cargas de trabajo, previstos durante las operaciones de producción. Estas pruebas son cruciales para comprender cómo se comportará la aplicación en condiciones reales y garantizar que pueda manejar el tráfico esperado sin degradar su rendimiento. (Meier et al., 2007).

Para ilustrar mejor el concepto de las pruebas de carga, se presenta el siguiente videotutorial:

Video 2. Prueba de carga



Enlace de reproducción del video

Síntesis del video: Prueba de carga
Video tutorial del experto donde se realiza paso a paso la descarga del JDK (Java Development Kit), desde la página de ORACLE, con la creación de reportes y su ejecución.

Pruebas de seguridad

Las pruebas de seguridad comprenden un conjunto de actividades ejecutadas con el objetivo de identificar fallas y vulnerabilidades, buscando minimizar el impacto de ataques, el acceso no autorizado a información sensible y la pérdida de datos. Estas pruebas son esenciales porque los productos de "software" deben garantizar constantemente la confidencialidad, disponibilidad e integridad de los datos y funcionalidades que gestionan. La pérdida o corrupción de esta información puede tener consecuencias críticas en el ámbito empresarial. Por tanto, es crucial que estas pruebas se realicen de manera continuada a lo largo de todas las fases del proyecto, asegurando la protección efectiva contra amenazas emergentes.

A continuación, se presenta el siguiente videotutorial de instalación del "software" OWASP para realizar una prueba de seguridad:

Video 3. Prueba de seguridad

Enlace de reproducción del video

Síntesis del video: Prueba de seguridad
Video tutorial del experto donde se realiza la descarga de OWASP ZAP para hacer una prueba de seguridad, con todas las indicaciones para hacer la instalación según la configuración del equipo, y finalmente hacer las pruebas de seguridad en algunas páginas de ejemplo en donde se ven reflejadas las vulnerabilidades de la página probada.

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:

a) Analizar el impacto del defecto.
b) Investigar el problema, en qué condiciones se produce este fallo.
c) Analizar la severidad, si es alta o baja.
d) Variar los pasos realizados, se pueden ejecutar en orden diferente para observar si existe alguna variante en el resultado esperado.
e) Variar opciones de configuración, si estamos probando una aplicación web se puede probar con diferentes navegadores.
f) Determinar condiciones específicas bajo las cuales se reproduce el defecto, si estamos probando entradas de datos, entonces probar con diferentes datos.

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

La gestión de incidencias desempeña un papel crucial en el proceso de aseguramiento de la calidad del "software", ya que es en esta etapa donde se identifican y abordan los errores ("bugs"). El objetivo principal de tratar las incidencias detectadas es generar las correcciones necesarias para minimizar la posibilidad de que estos errores se repitan. Este proceso asegura que el "software" no solo cumpla con

los estándares de calidad requeridos, sino que también mejore continuamente, incrementando su fiabilidad y eficiencia.

Ciclo de vida de una incidencia

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

Figura 11. Ciclo de vida de una incidencia de software

- Nuevo: durante las pruebas de "software", los errores de "software" deben ser identificados y registrados por el "tester" de "software", siendo ese instante donde se reporta el error.
- Abierto: el equipo de desarrollo adquiere el incidente y empieza a analizarlo.
- Asignado: el incidente es asignado al desarrollador.
- Diferido: si el incidente aplica y es de bajo impacto.
- Reabierto: cuando el incidente es corregido y es retornado al equipo de "testers" de "software".

- Verificado: si la prueba es positiva.
- Cerrado: posterior a una verificación por parte del equipo de “testing”.

El proceso mencionado se adecúa a proyectos de “software” en fase de desarrollo, pero no así en ambientes 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:

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

Por otra parte, para estructurar el informe de incidentes se requiere tener en cuenta los siguientes puntos:

- Identificador.
- Resumen y descripción del incidente.
- Descripción de datos (fecha/hora, entradas, resultados esperados).
- Impacto que los incidentes tendrán con respecto a las pruebas.

Reporte de defectos

Cuando se confirma la existencia de un defecto, se elabora un reporte. Este documento detalla los fallos identificados durante el proceso de pruebas, subrayando la importancia de que cada defecto detectado cuente con un reporte individual. Esto

facilita la trazabilidad y su corrección posterior. A continuación, se describe la estructura típica de un reporte de defectos.

a) Identificador.
b) Título.
c) Reportado por.
d) Asignado a.
e) Fecha de creación.
f) Severidad.
g) Nombre de la aplicación.
h) Descripción.
i) Método de reproducción.
j) Información adicional.

Informe de resultados de pruebas

El informe de resultados de pruebas presenta y organiza un análisis conciso de los resultados obtenidos en las pruebas, así como las principales métricas utilizadas para la evaluación y verificación. Por tanto, la responsabilidad de este documento recae en el gerente de pruebas. Este informe debe incluir los siguientes puntos:

a) Alcance de las pruebas, es decir lo que es necesario incluir y no incluir.
b) Resumen de pruebas.
    - Casos de prueba planificados.
    - Casos de prueba ejecutados.
    - Casos de prueba exitosos.
    - Casos de prueba fallidos.
    - Casos de prueba bloqueados.

c) Listado de defectos detectados.

d) Conclusiones.

2. Elaboración del plan de mejora

Cuando una organización alcanza el éxito, suele deberse a su capacidad para integrar la mejora continua en cada uno de sus procesos y actividades diarias. La mejora se logra cuando la organización toma en cuenta tanto sus fortalezas como sus debilidades. Sin embargo, identificar la situación actual de la organización a través de un diagnóstico es también crucial. Una vez realizado el diagnóstico, se puede establecer la estrategia a seguir para que los destinatarios de los servicios perciban de manera significativa la mejora implementada.

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.

Figura 12. Ejemplo estructura de tabla de fortalezas, debilidades y área de mejora

Fortalezas	Debilidades	Áreas de mejora
1.	1.	1.
2.	2.	2.
3.	3.	3.

Resolver problemas y mejorar un área específica comienza por identificar las causas que los originaron. Por lo tanto, existen varias herramientas metodológicas diseñadas para esta finalidad. A continuación, se mencionan algunas de ellas:

Figura 13. Diagrama de espina de pescado comentado

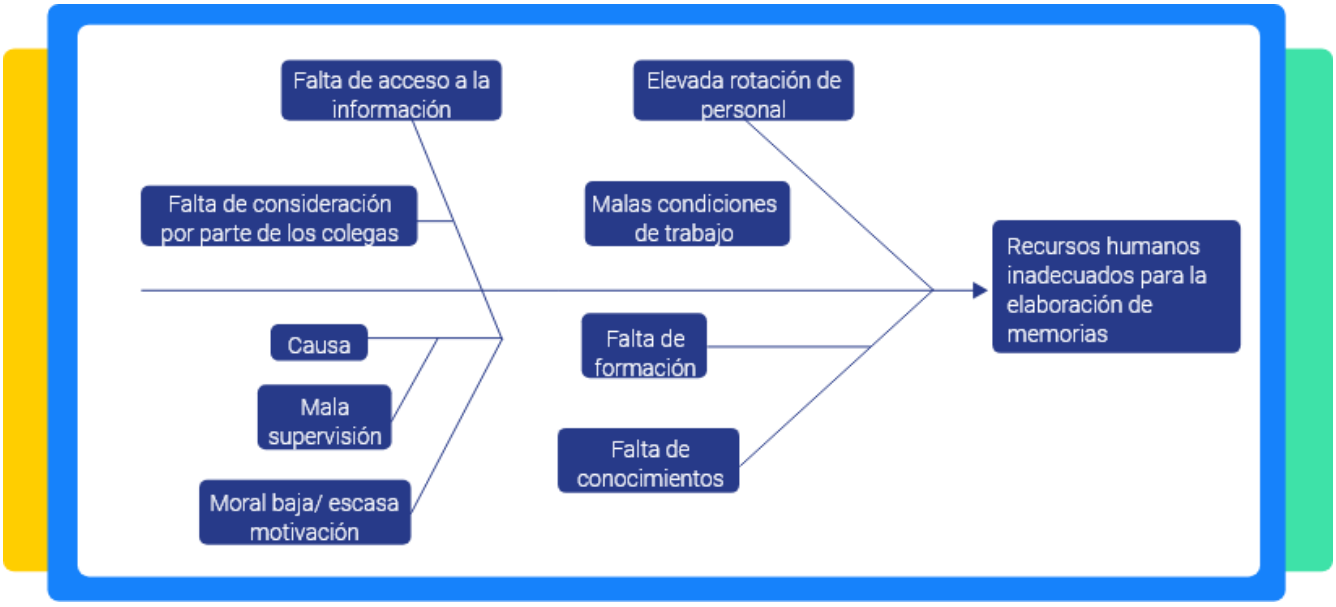


Figura 14. El diagrama de espina (causa-efecto)

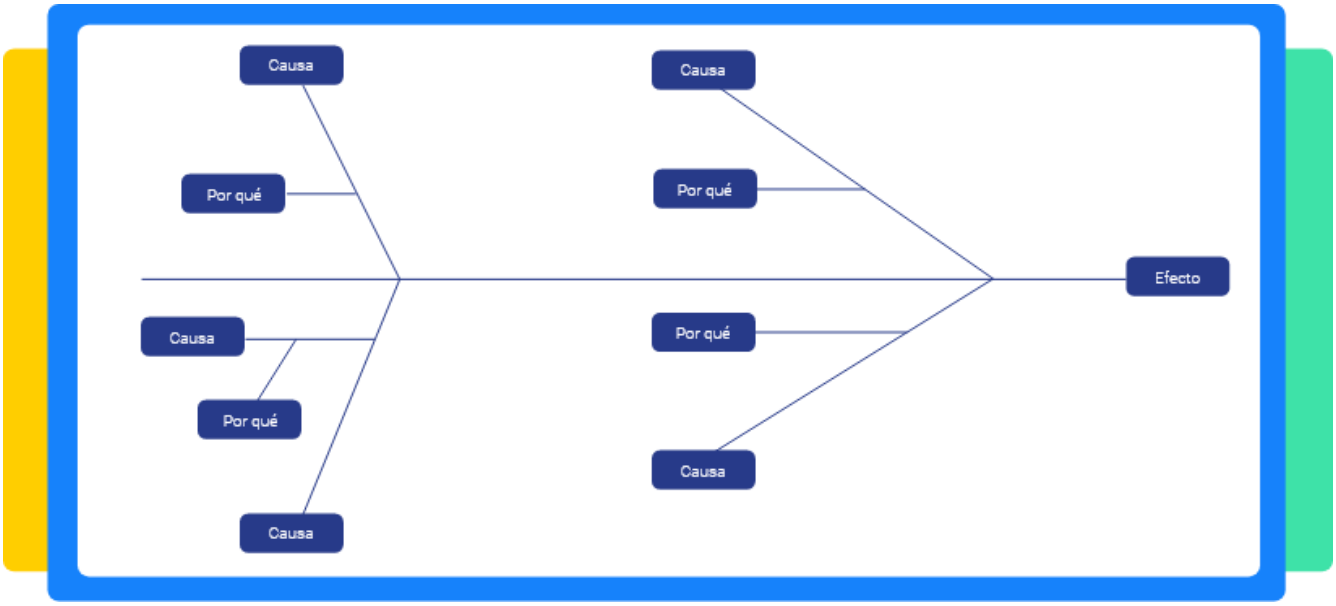


Figura 15. Diagrama de Pareto

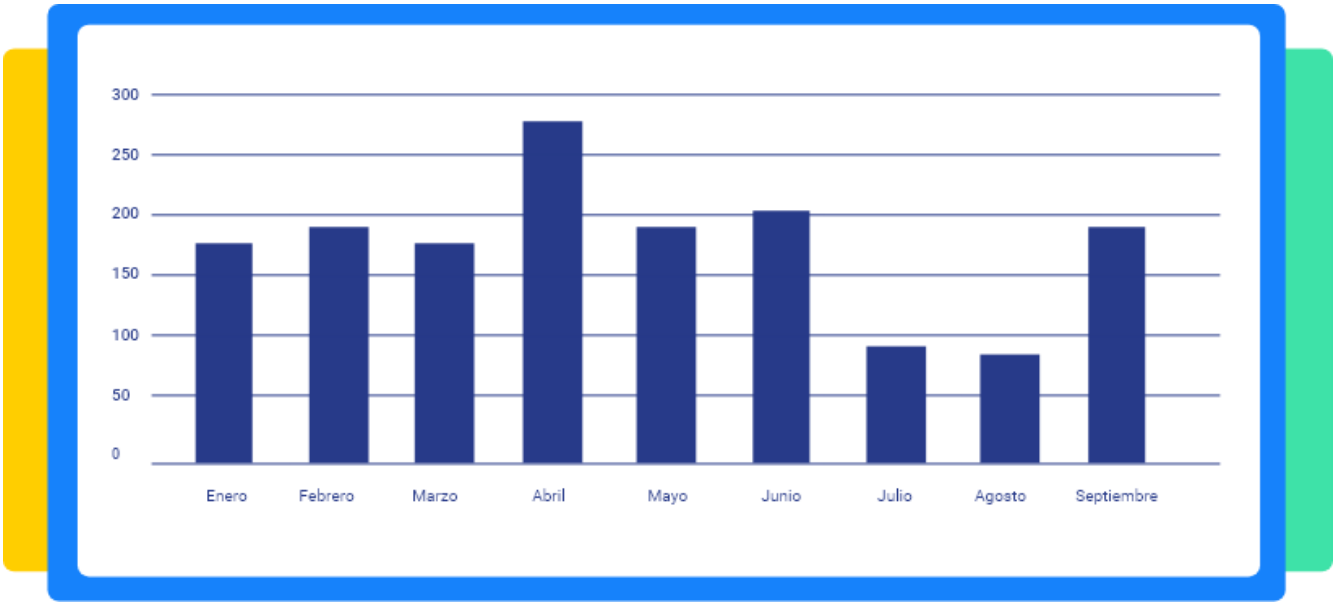


Figura 16. Tormenta de ideas

El uso de estas herramientas contribuye a un análisis más profundo del problema y se convierten en directrices clave al momento de establecer acciones de mejora.

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.

2.1. Seleccionar las acciones de mejora

En este punto, es necesario seleccionar las opciones de mejora para dar prioridad a las más adecuadas. Para ello, se pueden emplear técnicas como la del grupo nominal y la tormenta de ideas, entre otras. Estas técnicas permiten establecer acciones de mejora que se convierten en soluciones para las debilidades identificadas. Esto significa que, para alcanzar los objetivos establecidos, se debe contar con una lista de las principales actuaciones.

A continuación, se presenta una tabla utilizada para seleccionar las acciones de mejora. Para ello, es imprescindible haber identificado previamente el problema, la causa y su objetivo.

Figura 17. Ejemplo estructura de tabla sobre acciones de mejora

ÁREA DE MEJORA N.º 1:	
Descripción del problema	
Causas que provocan el problema	
Objetivo a conseguir	
Acciones de mejora	
Beneficios esperados	

2.2. Realizar una planeación

El listado de las principales actuaciones se genera sin tener en cuenta un orden de prioridad específico. No obstante, ciertas restricciones inherentes a las acciones pueden limitar su inicio, o incluso llevar a la posible eliminación o aplazamiento del plan de mejora. Por lo tanto, es crucial conocer las restricciones que limitan su viabilidad. Determinar un orden óptimo no se basa únicamente en identificar las acciones relacionadas con los factores de mayor urgencia; otros criterios también se consideran al tomar esta decisión. A continuación, se describen algunos de estos criterios:

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

- Mucha
- Bastante
- Poca
- Ninguna

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.

- Largo
- Medio
- Corto
- Inmediato

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.

- Ninguno
- Poco
- Bastante
- Mucho

2.3. Seguimiento del plan de mejora

En este momento, se procede a elaborar un cronograma para la implantación y seguimiento de las acciones de mejora. Dicho cronograma organizará de manera ordenada las prioridades junto con los plazos establecidos para su desarrollo.

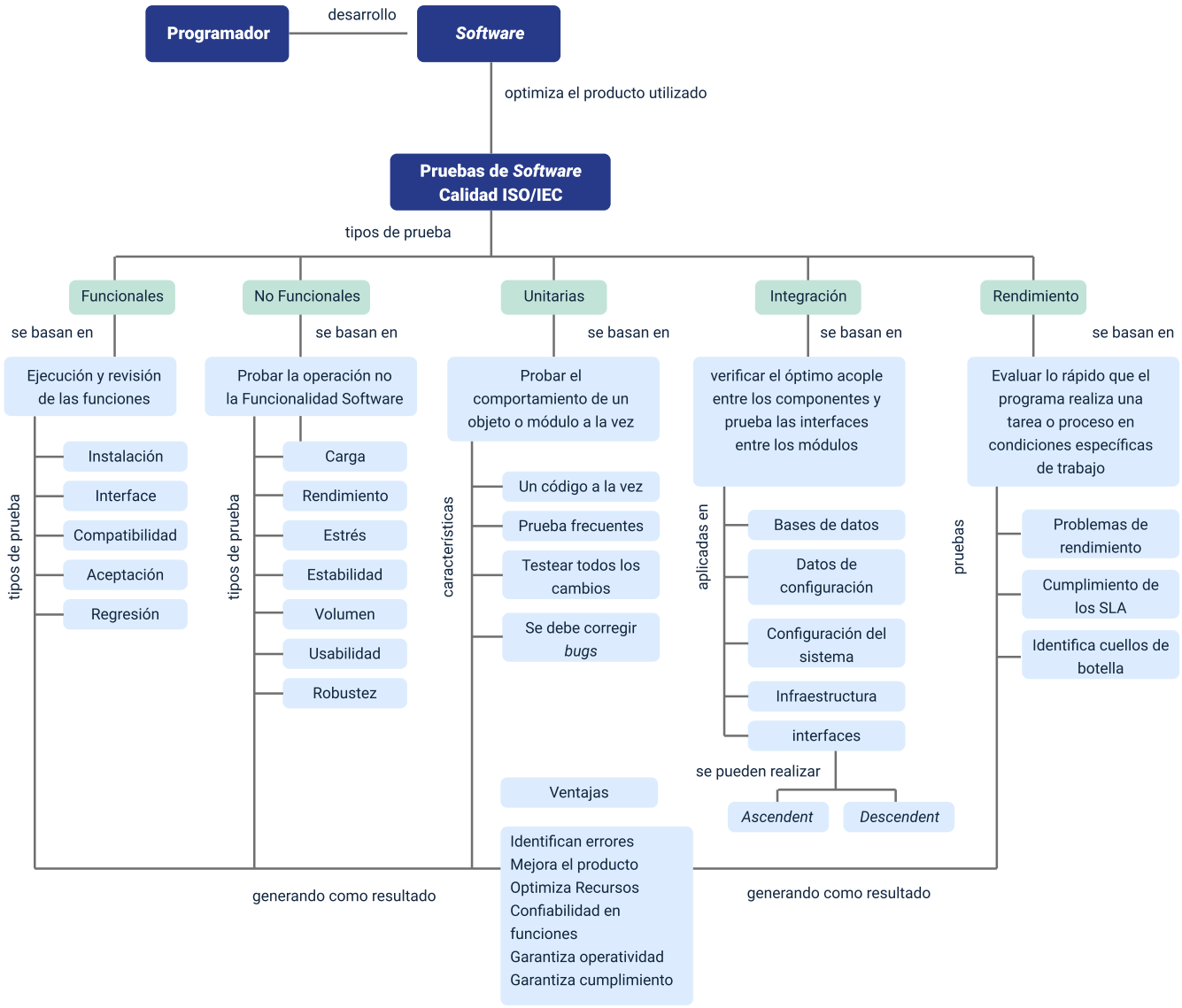
Una vez seleccionadas las acciones por orden de prioridad, se construye el plan de mejoras. Este incluirá los elementos necesarios para realizar un seguimiento detallado del plan, garantizando así su eficiencia y eficacia. Todo ello teniendo en cuenta la información presentada en la siguiente tabla:

Figura 18. Estructura plan de mejora

Acciones de mejora	Tareas	Responsable de tarea	Tiempos (inicio-final)	Recursos necesarios	Financiación	Indicador seguimiento	Responsable seguimiento

Síntesis

A continuación, se muestra un mapa conceptual con los elementos más importantes desarrollados en este componente.



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
1.1 Tipos de pruebas	SENA, E. d. (24 de 08 de 2021). Tipos de pruebas de “software”.	Video	https://www.youtube.com/watch?v=Uf1Kk52ONYc
1.1.1 Pruebas Funcionales	IBM (2021). Pruebas Funcionales.	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.	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	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	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.	Workshop	https://www.youtube.com/watch?v=EeRtQUljvOM

Glosario

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.

Automatización de pruebas: es la práctica que permite controlar la ejecución de un producto "software" de manera automática.

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 porqué 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.

Referencias bibliográficas

Beck, K., & Andres, C. (2004b). Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series) (2nd ed.). Addison-Wesley.

Bustamante Ramírez, J. (2011). Sistema de informes para pruebas de "software". Obtenido de 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 Modelos. Obtenido de http://hdl.handle.net/2099/15497

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://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf

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.

Mera Paz, J. A. (19 de 10 de 2016). Pruebas de Calidad “software”. Obtenido de https://repository.ucc.edu.co/handle/20.500.12494/962

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). Ingenieria del “Software”. Pearson Educacion.

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
Ervin Andrade	Experto Temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
Peter Pinchao	Experto Temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
Paola Alexandra Moya	Evaluadora Instruccional	Centro de Servicios de Salud - Regional Antioquia
Andrés Felipe Herrera Roldán	Diseñador de Contenidos Digitales	Centro de Servicios de Salud - Regional Antioquia
Edwin Sneider Velandia Suárez	Desarrollador Fullstack	Centro de Servicios de Salud - Regional Antioquia
Edgar Mauricio Cortés García	Actividad Didáctica	Centro de Servicios de Salud - Regional Antioquia
Daniela Muñoz Bedoya	Animador y Productor Multimedia	Centro de Servicios de Salud - Regional Antioquia
Andrés Felipe Guevara Ariza	Locución	Centro de Servicios de Salud - Regional Antioquia
Luis Gabriel Urueta Álvarez	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Jaime Hernán Tejada Llano	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia

Nombre	Cargo	Centro de Formación y Regional
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