**Datos de identificación del programa de formación**

|  |  |
| --- | --- |
| **PROGRAMA DE FORMACIÓN** | Desarrollo web full stack |

|  |  |  |  |
| --- | --- | --- | --- |
| **COMPETENCIA** | **220501099 - Realización de pruebas de calidad y funcionamiento a la solución de *software*.** | **RESULTADOS DE APRENDIZAJE** | **220501099-02 - Ejecutar las pruebas del *software* desarrollado de acuerdo con las especificaciones técnicas.** |

|  |  |
| --- | --- |
| **NÚMERO DEL COMPONENTE FORMATIVO** | **19** |
| **NOMBRE DEL COMPONENTE FORMATIVO** | **Pruebas y métricas** |
| **BREVE DESCRIPCIÓN** | El componente formativo, está diseñado con el fin que el estudiante adquiera los conocimientos sobre conceptos necesarios para su formación, tales como: técnicas de pruebas, pruebas, *Rational Test.* |
| **PALABRAS CLAVE** | ***Rational test*, técnicas de pruebas, pruebas** |

|  |  |
| --- | --- |
| **ÁREA OCUPACIONAL** | **2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS** |
| **IDIOMA** | **Castellano** |

TABLA DE CONTENIDOS

**1. Trazar la ruta de desarrollo de *software***

**2. Técnicas de prueba**

**3. Pruebas**

**4 *Rational Test***

**INTRODUCCIÓN**

|  |
| --- |
| Cuadro de texto |
| Apreciado aprendiz, bienvenido a este componente formativo, donde se abordarán las temáticas referentes a las pruebas de *software*, algunos *frameworks* y librerías para programación, ruta de desarrollo de s*oftware* de calidad, algunas técnicas de pruebas y el tema de *rational test*.  Después de haber codificado sus productos, deben ser probados para medir la calidad de estos, usando los diferentes tipos de pruebas que también servirán para controlar y encontrar defectos o errores que puedan tener en su codificación.  Le invitamos a ver el siguiente video donde se abordarán, de forma general, las temáticas que se estudiarán a lo largo del presente componente formativo: |

**GUION DE VIDEO INTRODUCTORIO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de recurso** | Video spot animado | | | |
| **NOTA** | **La totalidad del texto locutado para el vídeo debe tener mínimo 490 y máximo 510 palabras** | | | |
| **Título** | **Pruebas y métricas** | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración (voz en off)** | **Texto** |
| **Escena 1** |  | NA | Estimado aprendiz, queremos darle la bienvenida al presente componente formativo que corresponde a la temática de “Pruebas y métricas” del programa de tecnología en Desarrollo Web *Full Stack* del SENA, que, entre otras cosas, fortalecerá de forma teórica los conocimientos acerca de la calidad en el ciclo de desarrollo del *software*, los diferentes tipos de pruebas y los más usados. | Pruebas y métricas.  Calidad en el ciclo de desarrollo del *software.* |
| **Escena 2** |  | NA | Veamos, una de las fases en el ciclo de desarrollo clásico del *software* es la fase de pruebas, en donde se pueden realizar tanto pruebas funcionales como no funcionales, refiriéndonos a funcionales como aquellas pruebas de regresión, pruebas unitarias y de integración, pruebas de humo, entre otras; y a las no funcionales como, por ejemplo, las pruebas de *performance* o su usabilidad en general. | Fase de pruebas.  Pruebas funcionales y no funcionales. |
| **Escena 3** |  | NA | En este orden de ideas, para poder entregar productos de valor y calidad es importante realizar correctamente las pruebas de nuestras aplicaciones, dado que hoy en día son bastante importantes y solicitadas. Además, veremos varios enfoques teóricos sobre requisitos funcionales, no funcionales, calidad del *software* y métricas. Así mismo, trabajaremos en el *software* enfocado en la web, sin embargo, es importante saber que para la ingeniería web se usan muchos enfoques de la ingeniería de *software*. | Requisitos funcionales, no funcionales, calidad del *software* y métricas. |
| **Escena 4** |  | NA | En el presente componente formativo tendrá la oportunidad de ver diferentes conceptos, empezando por algunos temas como calidad en el desarrollo de *software*, se mostrarán conceptos sobre diferentes técnicas de pruebas de *software*, en las que se pueden encontrar: las pruebas basadas en especificación, y a su vez se verán las técnicas más destacadas de estas, como por ejemplo: partición en clases de equivalencia, análisis de valores límites, método de categoría partición, grafos, transición de estados por pares, basadas en especificaciones normales, y aleatorias. | Calidad en el desarrollo de *software.* |
| **Escena 5** |  | NA | Posteriormente, veremos las técnicas de prueba basadas en código, basadas en errores, basadas en el uso, basadas en la naturaleza de la aplicación y en qué tipo de aplicaciones las podemos encontrar; en particular, las aplicaciones web, en lo que se centra su formación. Así mismo, abordaremos un capítulo de pruebas en donde tendrá nociones sobre: automatización de pruebas, también sobre virtualización de servicios y los enfoques *Shift Left*, sus beneficios y su implementación. | Colocar aquí el texto que aparecerá dentro del video |
| **Escena 6** |  |  | Por último, veremos un capítulo relacionado con *Rational Test* o prueba racional, que incluye, básicamente, las siguientes temáticas: *rational test virtualization server y rational test workbench*. Todo lo anterior está enfocado en el ciclo de desarrollo del *software* y ,sobre todo, en la fase de pruebas. Solo hemos mencionado lo que veremos, pero a continuación, profundizaremos en qué consisten cada uno de los temas para lograr un mayor acercamiento y entendimiento de los mismos. |  |
|  |  |  | Le damos nuevamente la bienvenida a este componente formativo, esperamos que logre adquirir los conocimientos que hasta aquí hemos mencionado. Sabemos que esta ruta de aprendizaje será también de su agrado y beneficio. Recuerde que, en cualquier momento, cuenta con la ayuda y el acompañamiento de su instructor SENA para aclarar cualquier duda que pueda presentar y profundizar en el tema.  ¡Muchos éxitos en esta nueva etapa de aprendizaje! |  |
| **Nombre del archivo** | 228125\_V1 | | | |

**1. Trazar la ruta de desarrollo de *software***

|  |
| --- |
| Cuadro de texto |
| Para empezar, hay que aclarar que el proceso de desarrollo de *software* está asociado a una serie de tareas informáticas orientadas a la creación, diseño, ejecución y compatibilidad de *software*. En otras palabras, el *software* implica un conjunto de instrucciones o programas que le indican a un computador qué es lo que debe hacer. Además, funciona de manera independiente del *hardware* y esto permite que las computadoras puedan ser programables. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Infografía estática |
| **Texto introductorio** | Veamos, a continuación, el siguiente esquema que ilustra el proceso clásico de desarrollo de *software:* |
| **Imagen** | |

|  |
| --- |
| Cuadro de texto |
| En este orden de ideas, es importante entender que la ruta de desarrollo de *software* va desde el análisis, el diseño, la implementación, las pruebas, hasta el mantenimiento. Antes de seguir abordando esta secuencia temática, es importante hacer una claridad acerca del concepto de calidad, que, de hecho, no es algo sencillo de definir y que ha tenido una evolución a través de los años. Echeverry (2008) define calidad como: “Totalidad de propiedades y características de un producto, proceso o servicio que le confiere su aptitud para satisfacer unas necesidades expresadas o implícitas”, definición tomada de la ISO 8402. Por otro lado, en Significados (2022), se da otro concepto sobre calidad: “Capacidad que posee un objeto para satisfacer necesidades implícitas o explícitas, según un parámetro, un cumplimiento de requisitos de calidad”. Según la IEEE, se entiende como: “el grado con el que un sistema, componente o proceso, cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 1 |
| **Introducción** | Para cualquier persona, un producto o servicio puede ser considerado de “buena” calidad, mientras que, para otros, de “mala” calidad. El *software* se puede considerar como un servicio o producto y la calidad se puede aplicar también a nuestros productos *software*. En el desarrollo de *software,* los atributos de calidad se dan a partir de los requisitos No funcionales del sistema, que, como su nombre lo dice, son aquellos que el *software* no hace “directamente”. Por otro lado, una definición “formal” de requisitos No funcionales puede encontrarse en un sinnúmero de textos sobre ingeniería de *software*; sin embargo, aquí lo vamos a entender como “la manera en que los requisitos funcionales necesitan ser alcanzados”.  Veamos algunos de los requisitos No funcionales, según Tsu (2022): |
|  | |
| Requisitos de performance o rendimiento: se refiere a la efectividad con que una aplicación ejecuta una tarea, el tiempo que tarda en realizarla. | |
| Requisitos en tiempo real (*real-time*): cuando un programa necesita funcionar en tiempo real, lo que significa que debe completar el procesamiento dentro de un período de tiempo determinado, el rendimiento es un problema. | |
| Requisitos de modificabilidad: una aplicación bien diseñada debería contar con un fácil mantenimiento y modificabilidad si en alguna de las fases del desarrollo de *software* se pretende cambiar algo en su funcionalidad. | |
| Requisitos de seguridad: las aplicaciones construidas deben ser seguras, no solo seguridad se refiere a que alguien pueda penetrar en ella y hacer daños o acceder a información privada, sino que sea tolerable a errores, se puedan controlar dichos errores, registre errores y transacciones u operaciones realizadas, incluso su código o estructura sea mantenible. | |
| Requisitos de usabilidad: los usuarios finales deben poder manipular el producto final de manera “fácil”, debe ser una aplicación amigable en su manejo e incluso interfaz gráfica de usuario. En una aplicación web, por ejemplo, el menú de navegación sea cómodo de manejar, además se pueda visualizar en distintos dispositivos como tabletas, móviles, etc. Incluso para el caso de aplicaciones web está relacionado con la accesibilidad web. | |

Nota: algunos mencionan que un requisito es lo mismo que un **requerimiento**. En teoría podrían ser iguales, pero más exactamente, un requerimiento es lo que el cliente solicita, mientras un requisito es lo que debe cumplir el *software*.

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Slider pasos | |
| **Introducción** | Entre algunos atributos de calidad de *software* que se pueden mencionar, según la ISO 9126, están: | |
| **Funcionalidad:** | es el grado en que el *software* satisface las necesidades indicadas. |  |
| **Confiabilidad:** | cantidad de tiempo que el *software* está disponible para su uso. |  |
| **Usabilidad:** | grado en que el *software* es fácil de usar. |  |
| **Eficiencia:** | grado en que el *software* hace óptimo el uso de los recursos del sistema. |  |
| **Facilidad de mantenimiento:** | la facilidad con que una modificación puede ser realizada. |  |
| **Portabilidad:** | la facilidad con que el *software* puede ser llevado de un entorno a otro. |  |

|  |
| --- |
| Cuadro de texto |
| Los factores que afectan la calidad se pueden categorizar en dos (2) grandes grupos, tales como: factores que se pueden medir directamente y factores que se pueden medir sólo indirectamente. En todos los casos debería aparecer la medición, se debe comparar el *software* con una referencia y llegar a una conclusión sobre su calidad. La propuesta de McCal en Pressman (2006) sobre los factores que afectan la calidad del *software* están enfocados en tres aspectos importantes como lo son: sus características operativas, su capacidad de cambios, y su adaptabilidad a nuevos entornos. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Infografía estática |
| **Texto introductorio** | Estos factores fueron propuestos en los años 70, sin embargo, hoy en día, siguen siendo válidos, incluso para aplicaciones modernas de *software*: |
| **Imagen**  Diagrama  Descripción generada automáticamente | |
|  |  |

|  |
| --- |
| Cuadro de texto |
| ~~Para una mayor claridad, veamos la siguiente tabla en la que se ilustra la relación entre los factores y métricas de calidad.~~ |

|  |  |
| --- | --- |
| **Tipo de recurso** | Banner de presentación |
| **Texto introductorio** | Para una mayor claridad de este tema, veamos la siguiente tabla en la que se ilustra la relación entre los factores y métricas de calidad: |
|  | |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Slider Citas | |
| **Introducción** | Antes de referirnos al término de **métrica**, lea esta definición: | |
| “…La medición es el proceso por el que se asignan números o símbolos a los atributos de las entidades en el mundo real, de tal manera que las definan de acuerdo con unas reglas claramente definidas…”  Ramírez, J. (2022). Métricas y técnicas, sitio web. | |  |

|  |
| --- |
| Cuadro de texto |
| Cuando se habla de métricas, como su nombre lo dice, es algo que puede ser medible, por ejemplo: la longitud, el tiempo, la masa, el volumen, etc. Pero ocurre que, en el contexto de *software*, estas medidas no son absolutas. Las métricas del *software* son medidas indirectas que van a proporcionar una representación de la calidad del este; las métricas ayudan a los ingenieros de *software* o informáticos a profundizar en el diseño y la construcción de las aplicaciones *software* que realizan y así obtener un producto de calidad. Las métricas se deben tener en cuenta en cada una de las fases del desarrollo de *software*. No obstante, muchos autores afirman que el *software* no puede ser medible. |

Nota: Para el desarrollo de productos *software* de calidad existen un sinnúmero de modelos y *framewoks* que dependen de los autores. Al igual que los requisitos funcionales y atributos de calidad también pueden variar, pero muchos de ellos mencionados anteriormente son en común de manera universal.

**2. Técnicas de prueba**

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Slider Presentación | |
| **Introducción** | Los productos *software* deben ser probados para medir su calidad y que cumplan con las especificaciones planteadas iniciales o los cambios que se puedan realizar durante su proceso de desarrollo; las pruebas hacen parte del ciclo de desarrollo del *software*. A continuación, se abordan conceptos sobre las diferentes técnicas de pruebas: | |
| **Basadas en la especificación**  Estas técnicas usan la especificación del sistema a probar para generar, usando algún tipo de algoritmo, casos de prueba, sin conocer la estructura interna de dicho sistema a probar. También llamadas pruebas de caja negra. La prueba de caja negra se refiere a un método o técnica de prueba en el que se le considera al *software* como una caja negra, sin preocuparse por los detalles de implementación, el conocimiento de la ruta interna y la estructura del código interno del *software*. Esta técnica de prueba se basa completamente en las especificaciones y requisitos del *software*. El enfoque de la prueba de caja negra está en la salida y las entradas del sistema de *software* en lugar del conocimiento interno del programa.  El sistema que se somete a este tipo de pruebas es el que se considera como la “caja negra”, que puede ser cualquier *software* como una base de datos, un sitio web o un sistema operativo, etc. Las pruebas de caja negra se centran en probar la funcionalidad completa del sistema, así como el comportamiento de este.  Esta técnica es fundamental durante las etapas del ciclo de vida de las pruebas de *software*, como las pruebas de regresión, la aceptación, la unidad, el sistema, la integración y el desarrollo de *software*.  Las técnicas de prueba de caja negra son beneficiosas para los usuarios finales que desean realizar la verificación del *software*. Las técnicas más destacadas dentro de estas son:     * BVA o análisis de valor límite. * Partición de clases de equivalencia. * Pruebas de transición de estado. * Prueba de tabla de decisiones. * Prueba basada en grafos. * Técnica de adivinación de errores.   Los tipos de prueba de caja negra pueden ser: funcional, de regresión y no funcional. | |  |
| **Basadas en el código**  Estas técnicas se encargan de generar casos de prueba en función de un criterio de cobertura del código, que indicará cómo generar los casos de prueba, así como el criterio de parada, es decir, cuándo se debe parar de generar casos de prueba porque ya se ha alcanzado el nivel de cobertura del código esperado. Entre los criterios de cobertura existentes se encuentran:   * Criterios basados en el flujo de control: los casos de prueba se generan basándose en el conocimiento de la estructura de control del programa a probar. A su vez, existen diferentes criterios de cobertura como son la cobertura de sentencias, la cobertura de ramas o decisiones, la cobertura de caminos, la cobertura de condiciones, la cobertura de condiciones/decisiones, o el criterio de cobertura de condición/decisión modificada, entre otros. * Criterios basados en el flujo de datos: los casos de prueba se generan basándose en el conocimiento de las operaciones que se realizan sobre las variables en el programa a probar. La idea principal es cubrir caminos del programa a probar en los que aparezca una determinada variable o variables. Para ello, se atiende a diferentes criterios basados en la definición de una variable, su uso, y el camino desde que se define una variable hasta que se usa. | |  |
| **Basadas en errores**  Este tipo de técnicas de pruebas son basadas en el uso de casos de prueba especialmente diseñados para revelar fallos probables o predefinidos. Entre los que se encuentran:     * Predicción de error: los casos de prueba se diseñan con la intención de “averiguar” qué defectos podrían estar presentes en el componente a probar, basándose únicamente en la experiencia de la persona que esté realizando las pruebas, por ejemplo, como resultado de los errores cometidos anteriormente o la historia de fallos descubiertos en proyectos anteriores. * Pruebas de mutación: un mutante es una versión ligeramente modificada (en tiempo de compilación o en tiempo de ejecución) del programa a probar, que difiere en un pequeño cambio sintáctico del programa original. Cada caso de prueba se ejecuta tanto con el código original como con cada uno de los mutantes generados, de tal forma que, si un caso de prueba funciona con el programa original, pero no con un mutante, identificando, por tanto, la diferencia entre ellos, dicho mutante se dice que ha sido “matado”. * Pruebas de inyección de fallos: la inyección de fallos es una técnica de pruebas que simula fallos en ciertas partes del código de un programa, con el objetivo de determinar si el sistema, con dicho fallo, se comporta de la manera adecuada al ser ejecutado. * *Fuzzing*: son un tipo de técnicas de inyección de fallos, que consisten en usar datos inválidos, inesperados y aleatorios (llamados *fuzz*) como entrada a un sistema con el objetivo de forzar la ocurrencia de fallos. Hay dos aproximaciones principales para crear los casos de prueba: por mutaciones de muestras de datos existentes, o generando nuevos datos de entrada basados en modelos de la entrada del sistema. | |  |
| **Basadas en el uso**  Las técnicas de pruebas basadas en el uso tratan de evaluar la fiabilidad del *software* cuando es utilizado de la misma forma que lo usan los usuarios finales en el entorno de producción, con la finalidad de que los defectos puedan aparecer antes de que sea usado por usuarios reales. La fiabilidad del *software* se define como la probabilidad de que un sistema *software* funcione correctamente sin que se produzca ningún fallo durante un intervalo de tiempo, bajo una serie de condiciones. | |  |
| **Basadas en la naturaleza de la aplicación**  Las técnicas de pruebas explicadas anteriormente pueden ser aplicadas a cualquier tipo de *software*. Sin embargo, existen técnicas de pruebas más específicas para realizar las pruebas de forma más eficiente y eficaz para algunos tipos de aplicaciones. Por ejemplo, es posible encontrar técnicas de pruebas específicas para los siguientes tipos de aplicaciones:     * Aplicaciones orientadas a objetos: en las que se debe tratar con las características de este tipo de aplicaciones, como son la existencia de clases y objetos, abstracción, encapsulación, herencia o polimorfismo * Interfaces gráficas de usuario (GUI): ya sea aplicaciones de escritorio o interfaces web. En este caso, se debe evaluar si la interfaz funciona de la manera esperada, teniendo en cuenta los elementos gráficos que se muestran y cómo éstos reaccionan ante las interacciones con los usuarios. * Aplicaciones web: en las que se deben evaluar diferentes criterios como son la funcionalidad, usabilidad, interfaz, rendimiento, seguridad, o accesibilidad de estas. * Programas concurrentes: los cuales suelen ser más difíciles de probar que los programas secuenciales, puesto que, al igual que ellos, las pruebas también deben ser ejecutadas concurrentemente, causando en muchos casos que los fallos encontrados no puedan ser reproducidos de manera determinista. * Sistemas en tiempo real: los cuales están sujetos a restricciones estrictas de tiempo, es decir, las operaciones deben producir una respuesta en un tiempo determinado. * Sistemas de seguridad críticos: esto es, sistemas *software* cuyo fallo o mal funcionamiento puede perjudicar gravemente la vida, el medio ambiente o algún tipo de equipamiento. | |  |
| **Basadas en la finalidad de las mismas**  Las pruebas del *software* se llevan a cabo con la finalidad de realizar algún tipo de comprobación. Por un lado, los casos de prueba pueden ser diseñados para comprobar que las especificaciones funcionales se han implementado correctamente, pero también se pueden utilizar para comprobar otro tipo de aspectos más específicos, dando lugar a otros tipos de pruebas según su finalidad. Algunos ejemplos son los siguientes:     * Instalación: comprueban que el *software* puede ser instalado en el entorno objetivo. * Pruebas de compatibilidad: comprueban que el *software* sigue funcionando con otras aplicaciones, sistemas operativos o entornos diferente al original. * *Alpha* o *Beta Tester*: el objetivo es que un grupo representativo de usuarios utilice el sistema antes de realizar su paso a la etapa en producción. * De rendimiento: pruebas de carga, las pruebas de estrés o las pruebas de resistencia que evalúan la capacidad del sistema y los tiempos de respuesta ante una carga determinada o más allá de los límites para los que ha sido diseñado. * De recuperación: comprueban el comportamiento del sistema cuando se produce algún tipo de “desastre”. * Usabilidad: evalúan cómo de fácil es para los usuarios finales usar y aprender a usar el *software*. * Accesibilidad: comprueban que el *software* es accesible, por ejemplo, para personas con visión reducida, audición reducida o movilidad reducida, entre otros aspectos. * Seguridad: comprueban la seguridad del sistema, por ejemplo, para protegerlo contra el acceso de usuarios no autorizados o el abuso de usuarios autorizados. * Regresión: comprueban que las modificaciones realizadas en el sistema no causen comportamientos indeseados en otras partes del *software.* | |  |

**3. Pruebas**

|  |
| --- |
| Cajón de texto de color |
| Para poder entregar productos de valor y calidad, es importante realizar las pruebas de nuestras aplicaciones, dado que hoy en día son bastante importantes y solicitadas. |

**~~3.1~~ Automatización de pruebas**

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Carrusel de tarjetas | |
| **Introducción** | Veamos ahora la prueba automatizada, que es un método que utiliza una herramienta de prueba de automatización para escribir y ejecutar un caso de prueba o conjunto de pruebas de automatización en la aplicación de *software*, comparando los resultados reales con el comportamiento esperado y creando informes de prueba: | |
| En pocas palabras, los *scripts* de prueba de automatización son los *scripts* preparados antes de ejecutar la prueba. Cuando sea necesario, estos ejecutarán los códigos de las aplicaciones web para comprobar si cumplen o no los requisitos. | |  |
| Cuando se trata de pruebas automatizadas, es difícil para los *testers* o personas con el rol de pruebas ser buenos sin tener conocimientos de programación. Pero con el tiempo la herramienta de prueba ha madurado para simplificar el proceso de prueba. Hay muchas herramientas sin código en el mercado que proporcionan el entorno gráfico para que los probadores puedan crear fácilmente casos de prueba. | |  |
| Una de estas herramientas bastante usada es Selenium. Selenium es un kit o conjunto de utilidades para facilitar el trabajo de obtener casos de pruebas en las aplicaciones web. Para ello, se pueden grabar, editar y depurar casos de prueba que podrán ser ejecutados de forma automática e iterativa posteriormente. | |  |
| Las pruebas automatizadas ayudan a eliminar tareas que consumen demasiado tiempo y son demasiado tediosas para realizarlas manualmente; además, las pruebas automatizadas se pueden ejecutar repetidamente una vez que se han creado. En otras palabras, las pruebas automatizadas ayudan a aumentar la eficacia y la eficiencia de las pruebas de *software*. | |  |

**~~3.2~~ Virtualización de servicios**

La virtualización es una salida para resolver este problema de restricción son el código interno escrito por el desarrollador.

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Slider Imagen | |
| **Introducción** | El tiempo de comercialización, los costos de desarrollo y la calidad de la aplicación, son las principales prioridades para las organizaciones que desarrollan *software*. Veamos: | |
| Las restricciones siguen siendo un desafío fundamental para el desarrollo y la entrega de *software* oportuno, rentable y de alta calidad. Un área de enfoque para muchos equipos de desarrollo es la simulación y/o creación de apéndices del código de la aplicación para reducir las dependencias de los componentes del entorno de la aplicación. | |  |
| Las tres opciones que se utilizan tradicionalmente para resolver este problema de restricción son el código interno escrito por el desarrollador, las herramientas propietarias o las herramientas de código abierto, incluido Mockito. Sin embargo, existe una solución mucho más elegante: la virtualización de servicios. | |  |
| Cuando los ingenieros de *software*, los evaluadores y los administradores de sistemas escuchan por primera vez acerca de la virtualización de servicios, pueden confundirla con la virtualización de servidores o las máquinas virtuales. | |  |

**~~3.3~~ Enfoques Shift Left**

|  |
| --- |
| Cajón de texto de color |
| Es un enfoque para realizar pruebas de *software* en el que las pruebas se realizan antes en el ciclo de vida. Las prácticas de este enfoque lo que básicamente hacen es integrar sus pruebas en su proceso de desarrollo de *software* y, por lo tanto, descubrir errores antes cuando es más fácil y menos costoso repararlos. No realizar un buen diseño puede implicar costos a futuro en términos de tiempo y dinero. Con este enfoque se puede minimizar los errores o defectos de producción y acortar los ciclos de prueba que pueden alargarse. |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Slider pasos | |
| **Introducción** | Shift Left trata de descubrir tantos problemas lo antes posible, en el proceso de desarrollo de *software*, de modo que el costo de solucionarlos se encuentre bajo control y también contribuya a mejorar la calidad de los productos enviados. Se concentra en hallar errores en las fases de diseño y desarrollo mucho más que en el resto, y así no tener problemas en producción. Veamos los beneficios que aporta: | |
| Detección temprana: | Es importante, ya que a futuro puede ser un problema peor. |  |
| Ahorro de costes: | Mejorar la eficiencia en los equipos de desarrollo. |  |
| Fiabilidad en las pruebas: | Mejor comprensión del negocio, por lo que se diseñan pruebas más completas porque la participación en el control de calidad desde el principio. |  |
| Potencia el trabajo en equipo: | La calidad es responsabilidad de todos. |  |
| Acelera la entrega: | Permite probar lo antes posible en los *pipelines* de desarrollo. |  |

**Implementación**

|  |
| --- |
| Cajón de texto de color |
| Estimado aprendiz, lo invitamos a ver el siguiente video en el que se observa el paso a paso en la implementación de las diferentes pruebas, para aplicar con éxito el enfoque Shift Left: |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de recurso** | Video spot animado | | | |
| **NOTA** | **La totalidad del texto locutado para el vídeo debe tener mínimo 490 y máximo 510 palabras** | | | |
| **Título** | A continuación, mencionamos el paso a paso de dicha implementación: | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración (voz en off)** | **Texto** |
| **Escena 1** |  | NA | Apreciado aprendiz, bienvenido a esta nueva experiencia de conocimiento, en el presente video conoceremos el paso a paso que debe seguirse a la hora de implementar el enfoque *Shift-left*:  1.Contar con una estrategia operativa definida.  Las organizaciones deben creer que las pruebas deben realizarse en las primeras etapas del ciclo de vida, por lo tanto, debe existir una estrategia operativa definida, deben definirse e implementarse puertas de calidad, prácticas de calidad y actividades de prueba a lo largo del ciclo de desarrollo. |  |
| **Escena 2** |  | NA | 2. Tener un nivel avanzado de madurez en Agile.  La metodología ágil incluye pruebas como parte integral del ciclo de desarrollo más corto. Por lo tanto, las pruebas de cambio a la izquierda encajan muy bien en el mundo ágil. Para implementar con éxito el enfoque *Shift-left* las organizaciones de TI ya deben traer consigo un cierto nivel de madurez en términos de seguimiento e implementación de la metodología ágil. |  |
| **Escena 3** |  | NA | 3. Adoptar la automatización de pruebas.  Dado que las pruebas de cambio a la izquierda implican pruebas frecuentes, el equipo de desarrollo debe adoptar herramientas de automatización de pruebas. En general, la automatización de pruebas acelerará el ciclo de vida del desarrollo y permitirá a las organizaciones de TI reducir el tiempo de comercialización. Además de eso, garantiza que se encontrarán menos errores más adelante en el ciclo de vida del desarrollo del *software*. |  |
| **Escena 4** |  | NA | 4. Implementar canalización de CI/CD.  Al automatizar la implementación de nuevas compilaciones y ejecutar pruebas para cada incremento de código, los desarrolladores podrán concentrarse en tareas esenciales como la codificación y el proceso de revisión. La calidad también se verá impulsada porque los defectos se pueden detectar y remediar temprano durante la integración del código antes de pasar a la siguiente etapa. |  |
| **Escena 5** |  | NA | 5. Realice pruebas tempranas y frecuentes (manual y automática).  Las pruebas de *software* deben comenzar temprano en el ciclo de vida del desarrollo de *software*. Entre desarrolladores y analistas de calidad, se pueden realizar varios tipos de pruebas para detectar defectos de manera temprana. Los tipos de pruebas pueden ser pruebas unitarias, pruebas estáticas de código, pruebas de servicios, integración de sistemas, pruebas funcionales, entre otras. Es importante automatizar tanto como sea posible estas pruebas para reducir el esfuerzo y la presión sobre las actividades de prueba y para proporcionar una retroalimentación más rápida sobre la estabilidad del código y el sistema. |  |
| **Escena 6** |  |  | 6. Lanzamiento e implementación automatizados.  Cuanto más automatizado sea el proceso de generación de lanzamiento y despliegue, menor será la probabilidad de que ocurra un error humano durante este proceso. Ya se ha demostrado que una canalización de CI/CD saludable mejorará la eficiencia y la eficacia del equipo, lo que permitirá aumentar la productividad al mismo tiempo que se entrega con el nivel de calidad esperado. |  |
|  |  |  | Estimado aprendiz, después de ver el anterior paso a paso para la implementación del enfoque *Shift-left*, le invitamos a seguir estudiando el siguiente tema: ***Rational Test***.  Sabemos que toda esta experiencia de aprendizaje será de gran provecho para su formación, muchos éxitos. |  |
| **Nombre del archivo** | 228125\_V2 | | | |

**4 *Rational Test***

|  |
| --- |
| Cajón de texto de color |
| En la web oficial de IBM (2022) se encuentran las características de estas tecnologías, como se resume a continuación. |

**~~4.1~~ Rational Test Virtualization Server**

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Tarjetas Avatar | |
| **Introducción** | Permite pruebas tempranas y frecuentes en el ciclo de vida del desarrollo. Elimina las dependencias al virtualizar parte o la totalidad de una aplicación o base de datos, por lo que los equipos de prueba de *software,* no tienen que esperar a que comience la disponibilidad de esos recursos.  Algunos beneficios y características son: | |
| Reduce el costo. Costo “reducido” para implementar y configurar entornos de aplicaciones complejos, según IBM. | |  |
| Gana velocidad y calidad. Ofrecen agilidad y previsibilidad con pruebas de integración continuas al principio del ciclo de vida del desarrollo. | |  |
| Reduce riesgos. Permite integrar con **IBM** ***Collaborative Lifecycle Management*,** que proporciona pruebas colaborativas. | |  |
| Virtualización de servicios, *software* y aplicaciones. | |  |
| Actualización, reutilización y compartir entornos virtualizados. | |  |
| Soporte para tecnologías de *middleware*. Admite un amplio conjunto de tecnologías de *middleware* y mensajería y proveedores de integración en varias industrias. | |  |
| Integración con otras herramientas. Ofrece todas las características de **IBM *Rational Performance Test Server*** (otra herramienta de IBM) para ayudar a validar el rendimiento y la escalabilidad de las aplicaciones. | |  |
| Precios e implementación flexibles. Según IBM, **IBM *Cloud DevOps for Hybrid Deployment*** ofrece un modelo de consumo basado en FlexPoints adquiribles. Estos se pueden asignar a través de aplicaciones en el paquete. | |  |

**~~4.2~~ Rational Test Workbench**

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 1 |
| **Introducción** | Esta solución proporciona pruebas de API, pruebas de interfaz de usuario funcional, pruebas de rendimiento y virtualización de servicios. Según IBM, presentan sus informes bastante completos, al igual que una buena interfaz gráfica. Proporciona herramientas para realizar pruebas de *software* que admiten un enfoque de DevOps, como pruebas de API, pruebas funcionales de interfaz de usuario, pruebas de rendimiento y virtualización de servicios. Automatiza y ejecuta pruebas antes y con más frecuencia para identificar errores cuando su corrección sea menos costosa. Incluso IBM, menciona algunos beneficios, que se pueden encontrar en su web oficial, como: |
|  | |
| Reducción de pruebas manuales. Se pueden automatizar todas las pruebas, como la regresión/funcional tradicional o la tecnología de integración y móvil, entre otras. | |
| Simplicidad de la experiencia de usuario. Se ofrece un entorno de creación completamente integrado para ofrecer una experiencia del usuario coherente en diversos dominios. | |
| Hallar defectos de las aplicaciones anticipadamente. Se puede realizar la creación de componentes virtuales e iniciar las pruebas. No es necesario esperar a que todo esté disponible para iniciar las pruebas. | |
| Trabajo con herramientas actuales. Permite realizar integración con otras herramientas para crear escenarios de pruebas, incluyendo aplicaciones de escritorio tradicionales, web y móviles. | |
| Mejoramiento en la calidad. Permite realizar la integración con otra herramienta de IBM, llamada IBM *Rational Quality Manager* para mejorar el rendimiento de las tareas de realización de pruebas. | |
|  | |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Carrusel de tarjetas | |
| **Introducción** | Mencionemos también algunas de sus características: | |
| Creación sin código. Permite utilizar pruebas de guion gráfico (*story-board*) para simplificar la creación de pruebas funcionales y de regresión que combinan texto para pruebas en lenguaje natural con edición visual. | | Colocar ícono relacionada con el texto |
| Modelos de carga de trabajo y pruebas de rendimiento visual sin *script*. Permite “acelerar” la entrega de conjuntos de pruebas de rendimiento a gran escala con la gestión automática de respuestas de servidor dinámico. | |  |
| Pruebas de integración continuas. Para desarrollar pruebas de nivel de servicio mediante la grabación del comportamiento del sistema actual, utilizando modelos de lenguaje de ejecución de procesos de negocio o el diseñador de pruebas visual. | |  |
| Emulación de cargas de trabajo precisa. Para “simplificar” el modelado de diferentes grupos de usuarios y condiciones de carga con el planificador gráfico de cargas de trabajo. | |  |
| Extensibilidad para estándares y protocolos. Adaptación de los servicios actuales o personalizados con funciones de extensibilidad, incluyendo la inserción de código Java o las transformaciones de datos personalizadas, entre otras. | |  |
| Establecimiento de precios y despliegue flexibles. Se da la opción de ganar IBM *FlexPoints* de agilidad, que se pueden adquirir y asignar de acuerdo con las necesidades exclusivas de su negocio. | |  |

|  |  |
| --- | --- |
| **Tipo de recurso** | **Infografía estática** |
| **Texto introductorio** | La virtualización de servicios suele confundirse con el Mockeo (Mocking o imitación) y la creación de *stubs* de simulación, pero no son lo mismo. Veamos: |
| Mockeo (Mocking) vs. Virtualización de servicios | |
| **Código de la imagen** |  |

|  |
| --- |
| Cuadro de texto |
| Recuerde explorar los demás recursos que se encuentran disponibles en este componente formativo; para ello, diríjase al menú principal donde encontrará la síntesis, una actividad didáctica para reforzar los conceptos estudiados, material complementario, entre otros. |

**SÍNTESIS**

|  |  |
| --- | --- |
| **Tipo de recurso** | Síntesis |
| ~~Desarrollo web~~ *~~full stack~~*  ~~Síntesis:~~ **~~Pruebas y métricas.~~** | |
| **Introducción** | El siguiente mapa integra los criterios y especificidades de los conocimientos expuestos en el presente componente formativo. |
|  | |

**ACTIVIDAD DIDÁCTICA**

|  |  |
| --- | --- |
| Tipo de recurso | Actividad didáctica. Opción múltiple |
| Apreciado aprendiz, a continuación, encontrará una serie de preguntas que deberá resolver con el objetivo de evaluar la aprehensión de los conocimientos expuestos en este componente formativo: |  |
| Pregunta 1  De las siguientes es una fase en el ciclo de desarrollo de *software:* |  |
| *Shift left.* | Pruebas (Correcta). |
| *Rational Test.* | Selenium. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo, la temática Trazar la ruta de desarrollo de *software.* |
| Pregunta 2  Según la norma ISO 9126, NO se podría considerar como atributo de calidad general: |  |
| Pruebas unitarias (correcta). | Usabilidad. |
| Funcionalidad. | Portabilidad. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Trazar la ruta de desarrollo de *software.* |
| Pregunta 3  Una de las siguientes es una técnica de prueba mencionada en el contenido: | Especificar la imagen que acompañará el texto |
| Basada en el código (correcta). | Basada en métricas. |
| Basada en Java. | De eficiencia. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Técnicas de prueba. |
| Pregunta 4  Otro nombre que pueden recibir las pruebas basadas en especificación: |  |
| Basadas en código. | Caja negra (correcta). |
| Basadas en la naturaleza de la aplicación. | Ninguna de las anteriores. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Técnicas de prueba. |
| Pregunta 5  Otro nombre que pueden recibir las pruebas basadas en especificación: |  |
| Basadas en código. | Caja negra (correcta). |
| Basadas en la naturaleza de la aplicación. | Ninguna de las anteriores. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Técnicas de prueba. |
| Pregunta 6  Una técnica destacada de las pruebas basadas en especificación, es: | Especificar la imagen que acompañará el texto    Imagen 5: Construya aquí el código de la imagen |
| Basadas en grafos (correcta). | *Fuzzing.* |
| *Alfa* o *Beta* Tester. | Ninguna de las anteriores. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Técnicas de prueba. |
| Pregunta 7  Son ejemplos de pruebas basadas en la finalidad de las mismas: | Especificar la imagen que acompañará el texto    Imagen 5: Construya aquí el código de la imagen |
| De instalación y usabilidad (correcto). | *Alfa* o *Beta* y *Fuzzing.* |
| Accesibilidad y orientada a objetos. | Ninguna de las anteriores. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Técnicas de prueba. |
| Pregunta 8  Entre los beneficios de *Shit-left* mencionados están: | Especificar la imagen que acompañará el texto    Imagen 5: Construya aquí el código de la imagen |
| Detección temprana de errores y potencia el trabajo en equipo (correcto). | Accesibilidad y usabilidad. |
| Mutación e inyección de fallos. | Ninguna de las anteriores. |
| Retroalimentación positiva: Muy bien ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Pruebas. |
| Pregunta 10  Para la implementación de *Shift-left* se pueden tener en cuenta los siguientes: | Especificar la imagen que acompañará el texto    Imagen 5: Construya aquí el código de la imagen |
| Estrategia operativa definida, adoptar la automatización de pruebas (correcta). | Implementar canalización de CI/CD y crear un *script* en Java con JS. |
| Realizar la creación de *script* con NodeJS, pruebas unitarias. | Ninguna de las anteriores. |
| Retroalimentación positiva: Muy bien, ha seleccionado la respuesta correcta. | Retroalimentación negativa: Lastimosamente no has acertado, te invitamos a repasar nuestro componente formativo CF19, la temática Pruebas. |

**MATERIAL COMPLEMENTARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de recurso** | **Material complementario** | | |
| Tema | Referencia APA del material | tipo | Enlace |
| ~~Trazar la ruta de desarrollo de~~ *~~software~~* | ~~Artículo de noticias en línea·2012·Mascheroni~~ | ~~Artículo~~ | ~~http://sedici.unlp.edu.ar/handle/10915/19202~~ |
| Técnicas de prueba | Mascheroni, M. A. (2012). *Calidad de software e ingeniería de usabilidad.*  https://utncomunicacionprofesional.files.wordpress.com/2012/04/ingenieria-de-sitios-web.pdf. | Texto | https://utncomunicacionprofesional.files.wordpress.com/2012/04/ingenieria-de-sitios-web.pdf |
| Pruebas | İnan, E. (2022). *What is Unit Testing? - HardwareAndro*. Medium. https://medium.com/hardwareandro/what-is-unit-testing-435e8134b16e | Texto | https://medium.com/hardwareandro/what-is-unit-testing-435e8134b16e |
| *Rational Test* | Galani, L. (2022). *Best practices for test automation - assert(QA)*. Medium. https://medium.com/assertqualityassurance/best-practices-for-test-automation-536e7a94ded4 | Artículo | https://medium.com/assertqualityassurance/best-practices-for-test-automation-536e7a94ded4 |
| *Rational Test* | Weidner, F. A. G. (2022, 5). *Unit Testing with JUnit and Mockito - Ana Flávia G Weidner*. Medium. https://medium.com/@anaflaviagw/unit-testing-with-junit-and-mockito-55330d93b8a0 | Artículo | https://medium.com/@anaflaviagw/unit-testing-with-junit-and-mockito-55330d93b8a0 |

**GLOSARIO**

|  |  |
| --- | --- |
| **Tipo de recurso** | Glosario |
| Accesibilidad web: | tiene como objetivo lograr que las páginas web sean utilizables por el máximo número de personas, independientemente de sus conocimientos o capacidades personales e independientemente de las características técnicas del equipo utilizado para acceder a la web. |
| **Adaptabilidad:** | la capacidad del producto *software* para adaptarse a diferentes entornos especificados sin aplicar acciones o medios distintos a los previstos para este fin para el *software* considerado. [ISO 9126] Consulte también pruebas de portabilidad. |
| **Criterios de aceptación:** | los criterios de salida que debe satisfacer un componente o sistema para ser aceptado por un usuario, cliente u otra entidad autorizada. |
| Ingeniería de *software*: | disciplina formada por un conjunto de métodos, herramientas y técnicas que se utilizan en el desarrollo de programas informáticos, más conocidos como *softwares*. |
| Ingeniería web: | uso de métodos sistemáticos, disciplinados y cuantificables al desempeño eficaz, operatividad y crecimiento de programas de muy buena calidad en la World Wide Web. |
| **Pruebas de accesibilidad:** | pruebas para determinar la facilidad con la que los usuarios con discapacidades pueden usar un componente o sistema. |
| Pruebas de integración: | las pruebas de integración están destinadas a validar el funcionamiento de múltiples componentes de aplicaciones a medida que interactúan entre sí o se integran entre sí. |
| Pruebas unitarias: | una prueba unitaria solo valida la unidad más pequeña de un proceso computacional. Eso podría significar una prueba de un solo método o componente en una aplicación. |
| ***Software* a medida:** | *software* desarrollado específicamente para un conjunto de usuarios o clientes. Lo opuesto es el *software* estándar. |
| ***Software* de prueba automatizado:** | *testware* utilizado en pruebas automatizadas, como *scripts* de herramientas. |

**REFERENCIAS BIBLIOGRÁFICAS:**

|  |  |
| --- | --- |
| **Tipo de recurso** | **Bibliografía** |
| IBM. (2022). https://www.ibm.com/ | |
| López Echeverry, *et al*. (2008). Introducción a la calidad de software. *Scientia Et Technica, 2*(39). <https://doi.org/10.22517/23447214.3241> | |
| Mascheroni, M. A. (2012). *Calidad de software e ingeniería de usabilidad.*  https://utncomunicacionprofesional.files.wordpress.com/2012/04/ingenieria-de-sitios-web.pdf. | |
| ~~Significados (2022). Recuperado de: https://www.significados.com/calidad/~~ | |
| Pressman. (2006). *Ingeniería de software, un enfoque práctico*, 5ª. Edición. Editorial Pressman. | |
| Tsui .(2022). *Essential of Software Engineering*, 5a. edición. JB Learning. | |
| Torcivia, M. (2020). *Shift-left-Testing.* Medium. https://medium.com/globant/shift-left-testing-c838521617b0 | |













