|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | Gestión de analítica de datos |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501096. Desarrollar la solución de *software* de acuerdo con el diseño y metodologías de desarrollo. | RESULTADOS DE APRENDIZAJE | 220501096-02. Verificar el funcionamiento del *software*, según metodología y el modelo implementado. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 18 |
| NOMBRE DEL COMPONENTE FORMATIVO | Pruebas y calidad de *software* |
| BREVE DESCRIPCIÓN | Las pruebas de calidad de *software o testing,* son los procesos que proporcionan herramientas para constatar la calidad de una aplicación y con esto evitar que se presenten fallas en la implementación. Las pruebas hacen parte de las fases del desarrollo del *software,* y son fundamentales para descubrir los errores o *bugs,* antes que los usuarios finales. |
| PALABRAS CLAVE | Ágil, *bug*, calidad de *software,* funcionalidad, pruebas. |

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

**1 Introducción a pruebas de *software***

**2 Tipos de pruebas**

**3 Diseñar pruebas**

|  |
| --- |
| Cuadro de texto |
| Estimado aprendiz, bienvenido a este aprendizaje, orientado a las pruebas y calidad de *software*. Durante su desarrollo se abordarán los ejes temáticos de forma progresiva y secuencial, de la siguiente manera: pruebas de *software, testing* y calidad de *software*, tipos de pruebas, diseño y clasificación de las mismas. Al finalizar el desarrollo temático, estará en capacidad de determinar un plan de prueba, definir casos de prueba, validar el funcionamiento y tratar los errores encontrados.  En el siguiente video conocerá, de forma general, la temática que se estudiará a lo largo del componente formativo. |

**Guion de video introductorio**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de recurso** | Video motion | | | |
| **NOTA** | **La totalidad del texto locutado para el video no debe superar las 500 palabras aproximadamente** | | | |
| **Título** | **Pruebas y calidad de *software*** | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración** | **Texto** |
| 1 | Imagen:228131\_i1801 | NA | Las pruebas de *software* hacen parte de una de las etapas de su construcción | Construcción de *software* |
| **2** | Imagen:228131\_i1802 | NA | que usualmente se conoce como verificación y validación o también por la abreviación de *V&V*. | Verificación y validación |
| **3** | Imagen:228131\_i1803 | NA | Los procesos de verificación y validación tienen como propósito asegurar que las aplicaciones respeten las especificaciones | Asegurar requisitos |
| **4** | Imagen:228131\_i1804 | NA | y se ajusten a las necesidades de sus usuarios. | Necesidades |
| **5** | Imagen:228131\_i1805 | NA | En el desarrollo de un proyecto de *software* es frecuente que, por causa de distintos factores, se originen eventos o situaciones que amenazan los avances y el éxito de este. | Eventos de amenaza |
| **6** | Imagen:228131\_i1806 | NA | Generalmente, estos inconvenientes surgen por demoras en los tiempos de entregas, sobrecostos no contemplados y productos finales de mala calidad. | Amenazas imprevistas |
| **7** | Ilustración de concepto de objetivos de equipo  Imagen:228131\_i1807 | NA | La responsabilidad de evitar que un proyecto de *software* fracase, por las situaciones mencionadas, es de todo el equipo involucrado. | Colaboración de todo el equipo desde el jefe de proyecto hasta los especialistas de desarrollo |
| **8** | Proceso de certificación y normalización, negocio certificado por la iso, conformidad con las normas internacionales y concepto de garantía de calidad. Persona tocando el icono del certificado.  Imagen:228131\_i1808 | NA | Las pruebas de *software,* también conocidas como *testing* de *software* son utilizadas para evaluar la calidad de los productos | Las pruebas evalúan la calidad |
| **9** | Imagen:228131\_i1809 | NA | y, en consecuencia, perfeccionarlo con la identificación de errores, defectos e inconvenientes. | Identificar errores |
| **10** | Interfaz de usuario gráfica  Descripción generada automáticamente  Imagen:228131\_i1810 | NA | La verificación debe ser dinámica y constante a través de una serie de casos de pruebas limitadas y aplicadas a un conjunto de módulos o métodos que conforman el sistema. | Verificaciones dinámicas para los módulos del sistema |
| **11** | Imagen:228131\_i1811 | NA | El *testing* debe adaptarse a todos los enfoques de programación, ya sea de metodología ágil o en cascada; sin embargo, en todos ellos se deben garantizar los principios de verificación, validación y satisfacción del usuario. | *Testing* y enfoques de programación |
| **12** | Imagen:228131\_i1812 | NA | Lograr que el usuario perciba la menor cantidad de fallas posibles en una aplicación depende del diseño, planificación y ejecución de un buen plan de pruebas. | Minimizar la cantidad de errores |
| **13** | Imagen:228131\_i1813 | NA | Aplicaciones populares como *WhatsApp* presentan algún tipo de error en sus funcionalidades, lo que indica que es imposible crear el *software* perfecto, aun cuando se tengan todos los recursos disponibles. | Todas las aplicaciones contienen errores |
| **14** | Imagen:228131\_i1814 | NA | Las pruebas de *software* se pueden ejecutar de forma manual o automatizadas, ambas dependen del tipo de pruebas que requiere el producto. Las manuales son mayormente utilizadas para módulos del sistema que tienen un comportamiento indeterminado | Las pruebas pueden ser manuales o automatizadas |
| **15** | Imagen:228131\_i1815 | NA | y por lo tanto, necesitan de la supervisión de un experto para interpretarlas; por otro lado, las pruebas automatizadas son empleadas para aquellas funcionalidades que son de tipo sistemáticas, donde habitualmente se conocen todas las variables que intervienen en el proceso. | La elección depende de la complejidad de las pruebas |
| **16** | Imagen:228131\_i1816 | NA | La automatización de las pruebas de *software* libera a los probadores o *tester* de la realización de pruebas recurrentes y con ello permite el enfoque del tiempo y el esfuerzo en otros módulos del *software* que son más complejos y específicos. | Las pruebas automatizadas ahorran tiempo |
| **17** | Calificación de aplicaciones móviles, calificación en línea, marca de eficiencia. estableciendo estrellas para la aplicación, evaluación de funciones. personajes de dibujos animados de usuarios de teléfonos inteligentes. ilustración de metáfora de concepto aislado de vector.  Imagen:228131\_i1817 | NA | Existen numerosas herramientas para la automatización de pruebas como *Selenium, Gatling, Testim, postman*, entre otras. | Herramientas populares |
| **18** | Imagen:228131\_i1818 | NA | La gran mayoría de las herramientas de pruebas automatizadas emplean *machine learning* en sus procesos de detección de errores, por lo cual, a medida que más se utilizan son más precisas y efectivas para encontrar *bugs* en el código fuente. | El *machine learning* es útil para el *testing* |
| **19** | Ilustración de concepto de prueba de código de software  Imagen:228131\_i1819 | NA | En este componente formativo se conocerán los tipos de pruebas y algunas de las técnicas más utilizadas por los equipos de *testing* para lograr pruebas de *software* con eficiencia y calidad. | Realizar pruebas de *software* y calidad |
| **20** | Imagen:228131\_i1820 | NA | Igualmente, a elegir cuál es la metodología de pruebas que se ajusta a cada necesidad. | Elegir la metodología adecuada |
| **Nombre del archivo** | 228131\_v01 | | |  |

**Desarrollo de contenido**

# Introducción a pruebas de *software*

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| Las pruebas de *software* son una serie de procesos que buscan poner a prueba los sistemas o aplicaciones en distintos escenarios para determinar su correcto funcionamiento. Se aplican en cualquier fase del desarrollo del *software*, es decir, desde la etapa de análisis y diseño hasta la etapa de producción. La ejecución de las pruebas de *software* se puede realizar de forma automática para establecer la estabilidad de la aplicación; también facilita la identificación de cambios que afectan otras funcionalidades antes de que se prueben.  Imagen:228131\_i1822 | |

## *Testing* y calidad de *software*

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Slider Presentación | |
| **Introducción** | El *testing* y el aseguramiento de la calidad, también llamadoQuality Assurance (QA)*,* es un mecanismo para la verificación y validación de las funcionalidades de los programas o aplicaciones de *software* con el fin de ofrecer un producto sin defectos o errores. | |
| El propósito final es hacer coincidir los requisitos iniciales con un *software* de calidad, lo cual incorpora la integración de herramientas asistidas o automatizadas que permiten la evaluación de las características de interés. | | Imagen:228131\_i1823 |
| El proceso de *testing* es paralelo a las etapas de construcción del *software* para que las actividades se realicen con calidad en cada una de las fases, se eviten reprocesos y fallas de funcionalidad, previo a la puesta en producción. | | Interfaz de usuario gráfica, Texto, Aplicación  Descripción generada automáticamente  Imagen:228131\_i1824 |
| El *testing* es una actividad que demanda tiempo y dinero, sin embargo, si se implementan técnicas efectivas para los procesos de control de calidad, se podrán anticipar a problemas como fallas de arquitectura, funcionalidades incorrectas o inválidas, fallos de seguridad, entre otros. | | Imagen:228131\_i1825 |
| Considera el riesgo e impacto que genera para una empresa el retiro de productos del mercado por defectos de *software* que ponen en peligro la seguridad de los usuarios. | | Imagen:228131\_i1826 |
| Por ejemplo, un fabricante de autos descubre una falla en los sensores de su sistema de bolsas de aire; esto implica el retiro de todas las unidades de automóviles que presentan este problema. Por lo tanto, los gastos económicos y efectos negativos en su reputación, serán de gran impacto. | | Imagen:228131\_i1827 |
| Cuando en el proceso de construcción del *software* se asignan suficientes recursos para la ejecución de pruebas de calidad, los resultados se pueden evidenciar con el aumento en la confiabilidad del mismoy la calidad de los productos finales. | | Imagen:228131\_i1828 |
| Dentro de los procesos de *testing,* se encuentran dos conceptos fundamentales: **validación** y **verificación**. Estos controlan que el *software* que se está desarrollando, cumpla con las especificaciones y expectativas requeridas por el cliente. | | IlustraciÃ³n del control de calidad  Imagen:228131\_i1829 |
| La validación y la verificación comparten propósito. La validación vela porque el *software* satisfaga los requerimientos, es decir, compara el que está en construcción con los modelos definidos en los requerimientos, a través de sus casos de usos o historias de usuario. | | Confirm ok sign in checkbox and positive thumbs up in tiny persons concept  Imagen:228131\_i1830 |
| La verificación se encarga de controlar que el producto en desarrollo haga lo que tiene que hacer. Un ejemplo cotidiano para ampliar el entendimiento de estos conceptos puede ser el proceso de construcción de una casa, donde antes de empezar a construir se crea un plano o modelo que el cliente revisa para corroborar que cumple con sus expectativas. | | Titans5landingkycconoce a tu clienteekyc3  Imagen:228131\_i1831 |
| Esto es la **validación**. Finalmente, cuando se construye la casa, se realizan constantes comprobaciones del plano para confirmar que está de acuerdo con lo planteado. Esto es la **verificación.** | | A set of concepts of people thinking on a laptop. Forgot your password and login for the account on the web page. Flat vector illustration.  Imagen:228131\_i1832 |
| Para mejorar la calidad del *software* o servicios desarrollados, es necesario entender y conocer al cliente y sus necesidades. La creación de productos con altos niveles de calidad, permite aumentar la fiabilidad como desarrollador, disminuir el mantenimiento y soporte, para incrementar la satisfacción del cliente. (Alfaro & Carrizo, 2021) | | Marca de verificación en la pantalla del portátil.  Imagen:228131\_i1833 |
| Existen muchas definiciones para *testing* y calidad de *software*, sin embargo, una forma general de definirlo es: hacer *testing* es verificar dinámicamente la adaptación del producto o *software* a los requerimientos. | | Ilustración de devops de diseño plano  Imagen:228131\_i1834 |
| En el proceso de *testing* se reportan los riesgos para que los desarrolladores los tengan en cuenta y se pueda minimizar la probabilidad de ocurrencia y disminución del impacto, de ahí que riesgos y *testing* estén relacionados. No hacer *testing* durante la construcción de un *software* genera vulnerabilidades para el alcance del proyecto. | | Imagen:228131\_i1835 |
| La depuración, o *debugging*, es otro concepto fundamental en el *testing,* que permite a los desarrolladores implantar puntos de interrupción o *breakpoint* para analizar el estado del *software* en una parte dada del código. | | Concepto de desarrollador de software idea de programación y desarrollo de sistemas de codificación tecnología digital empresa de desarrollo de software escribiendo código ilustración vectorial aislada  Imagen:228131\_i1836 |
| Con esto se busca encontrar las causas de las fallas y corregirlas sin necesidad de ejecutar la totalidad del *software*. La depuración es realizada por los programadores y el *testing* por los *testers* o probadores de *software*; además, *el testing* expone la existencia de *bugs* o errores, y la depuración encuentra las causas de las fallas para corregirlas. | | Imagen:228131\_i1837 |

|  |  |
| --- | --- |
| **Tipo de recurso** | Infografía estática |
| **Texto introductorio** | El proceso de calidad de *software* garantiza productos con niveles de calidad y eficiencia adecuados para satisfacer las necesidades de los usuarios y minimizar los riesgos y las amenazas que generen impacto negativo. |
| Nota. Adaptado de Tordecilla, D. (2022). | |
| **Código de la imagen** | Imagen:228131\_i1838 |

## Principios de las pruebas

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 1 |
| **Introducción** | Los principios de las pruebas sumados con métodos, estándares y herramientas, maximizan el *performance* en la labor del *testing*, debido a que permiten el diseño de casos de pruebas, la ejecución de pruebas exploratorias y automáticas, entre otras utilidades. A continuación, se describe una serie de principios que fortalecen la actividad de pruebas de *software:* |
| Imagen:228131\_i1839 | |
| **Ejecutar pruebas expone las fallas del *software***  Las pruebas permiten exponer la presencia de fallas en un *software*, pero no la falta de ellas. Después del reporte de una falla y su sucesiva subsanación, es posible minimizar la probabilidad de que las fallas que no han sido encontradas permanezcan en el *software*, sin embargo, no se podrá confirmar que un producto está libre de errores. | |
| **Realizar pruebas exhaustivas no es viable**  Con excepción de casos particulares, donde el riesgo de la materialización de una falla tiene un impacto vital, la realización de todas las posibilidades de los datos y funciones de un *software,* comúnmente no es práctico, debido a los altos costos en recursos y tiempo.  En razón a lo expuesto, es conveniente en la etapa de diseño y estructuración de las pruebas de *software,* tener en cuenta la relación de costo/beneficio, sin dejar de enfocarse en verificar las funcionalidades más importantes y de gran impacto para el proyecto. | |
| **Probar en etapas tempranas**  El proceso de *testing* debe ser paralelo a las fases del desarrollo del *software*, por lo tanto, realizar pruebas tempranas, preferiblemente en la etapa de análisis y diseño, resulta muy ventajoso porque permite la detección de errores cuando los costos y recursos para su rectificación, son bajos en comparación, de si se detectan en fases finales, como en la fase de despliegue. | |
| **Agrupación de fallas**  Con frecuencia, existen módulos o funcionalidades que tienden a presentar más fallas o *bugs* que otros. Estos módulos por lo general se agrupan, de acuerdo con la regla 80/20, donde se establece que el 80 % de los *bugs,* están relacionados con el 20 % de las mismas causas; es decir, los módulos de un *software* están caracterizados por un nivel de relevancia, que permite aplicar estrategias de pruebas para un determinado grupo y priorizar los módulos etiquetados con altas probabilidades de presentar *bugs*. | |
| **Paradoja del pesticida**  La ejecución de pruebas repetidas en una misma ubicación o módulo del sistema, impide la detección de nuevos *bugs;* por lo tanto, para aumentar la probabilidad de encontrar nuevos errores, se recomienda que el *tester* realice la exploración en distintas ubicaciones y actualice constantemente las estrategias y métodos de las pruebas. | |
| **Las pruebas dependen del escenario**  Las estrategias y tipos de pruebas son elegidas dependiendo del sistema y el escenario que se desea verificar. Es decir, una aplicación de tipo comercial no será testeada con los mismos métodos que una aplicación del sector salud. | |
| **Falacia de ausencia de fallas**  Después de corregir las fallas encontradas en un proceso de *testing* y de realizar un nuevo recorrido de pruebas, sin encontrar *bugs,* no es posible determinar que el producto sea 100 % útil, por cuanto este es medido a través de los parámetros de satisfacción y expectativas de los usuarios. | |

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| La identificación y aplicación de los principios de las pruebas son útiles para crear y organizar estrategias innovadoras que proporcionan visiones distintas a las implementadas por los desarrolladores al momento de crear las funcionalidades del producto.  Concepto de innovación colorido dibujado a mano  Imagen:228131\_i1840 | |

## Metodología de desarrollo de *software* ágil

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Pestañas o tabs horizontales | |
| **Introducción** | La metodología de desarrollo de *software* ágil emerge por el alto volumen de trabajo que tienen las áreas encargadas de codificar y el escaso tiempo para entregar a producción las aplicaciones. Teniendo en cuenta estos factores, la metodología ágil responde de mejor manera a las necesidades del mercado y supera a los métodos tradicionales como la metodología en cascada, donde las entregas son lentas debido a que se sigue un flujo de fase a fase para finalmente llevar un producto a producción. (Ramos, *et al*., 2022)  La implementación de metodologías ágiles, además de satisfacer al cliente con entregas tempranas, tiene las siguientes ventajas: | |
| **Aumentar la velocidad y eficiencia en las entregas del producto** | Con la implementación de metodologías ágiles las organizaciones pueden elevar la velocidad y la eficiencia de los grupos de desarrollo. Considerando que, al dedicarse a entregas parciales en plazos cortos, son menores los tiempos que cuando se tienen entregas totales de un producto. Además, permite mejoras en el versionado del *software*. | Ilustración del concepto de camino  Imagen:228131\_i1841 |
| **Incrementar la calidad del *software*** | Cuando los desarrolladores interactúan constantemente con el cliente se generan retroalimentaciones que son fundamentales en la calidad del *software*, de hecho, uno de los objetivos de esta metodología es ofrecer espacios de permanente intercambio. | Ilustración del concepto de calidad del producto  Imagen:228131\_i1842 |
| **Encontrar *bugs* y evitar fallas** | Con la aplicación de metodologías ágiles, los *testers* tienen la posibilidad de generar alertas sobre los *bugs* casi en tiempo real, y con esto se evitan fallas mayores cuando el producto esté más completo. | Ilustración del concepto de corrección de errores  Imagen:228131\_i1843 |
| **Aumentar la satisfacción del cliente** | La comunicación permanente entre los desarrolladores y el cliente es percibida como un factor de compromiso con la finalización del proyecto de *software*. En efecto, cuando se brinda la opción para que el cliente realice retroalimentación constante, en pro de las mejoras, el *software* estará respondiendo a sus necesidades. | Se ilustra el concepto de retroalimentación media  Imagen:228131\_i1844 |

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 2 |
| **Introducción** | Existen metodologías ágiles cada una con un enfoque particular, sin embargo, todas comparten las ventajas antes mencionadas. A continuación, se exponen algunas de las metodologías más populares y usadas por las organizaciones que se dedican al desarrollo de *software*. |
| Ilustración del concepto de método scrum  Imagen:228131\_i1845 | |
| **Extreme Programming o programación extrema**  Metodología también conocida como XP, consiste en la agrupación de reglas que generan agilidad y resiliencia a las organizaciones en la ejecución de los proyectos de *software,* en contextos que son muy variables. Por lo tanto, cuando se aplica una metodología ágil XP, se establece como prioridad la entrega del *software* final, de conformidad con los requerimientos exactos del cliente.  Para que la metodología XP sea exitosa debe existir un alto compromiso de las partes involucradas y mucho trabajo en equipo. Es necesario que los requisitos del *software* se establezcan al inicio del proceso, para que continuamente se dé tratamiento a las necesidades que vayan surgiendo.    Algunas de las propiedades de la metodología XP son:   * Impulsar el diálogo permanente entre los clientes y el equipo de desarrollo. * Garantizar la flexibilidad en el cronograma del proyecto. * Priorizar la reparación de los *bugs* de manera rápida y ofrecer respuestas precisas y claras. * Centrar la atención en la entrega del *software* sobre cualquier otro requisito como manuales o guías. * Trabajar con equipos bien estructurados para garantizar el correcto funcionamiento del *software*.   La metodología XP promueve los cambios constantes en el proyecto de *software*. Con ello se busca que al producto se le realicen continuas evaluaciones para mejorar y cumplir los requisitos establecidos por el cliente. | |

|  |
| --- |
| **Kanban**  Es una de las metodologías más sencillas y fáciles de implementar en las organizaciones. El método consiste en tener una pizarra visual para clasificar las actividades que se encuentran en estado de pendiente, en curso, o finalizadas.  Kanban emplea seis procedimientos básicos para lograr incrementar la agilidad en el desarrollo de las actividades. Estos procedimientos se deben aplicar y revisar periódicamente para la satisfacción del cliente.   * Exponer todos los objetivos del proyecto. * Minimizar el trabajo en proceso para reducir los ciclos. * Controlar los flujos de trabajo del grupo. * Diseñar e implantar directrices y procedimientos de atención al cliente. * Ejecutar reuniones de retroalimentación. * Maximizar la colaboración en los equipos de desarrollo. |

|  |
| --- |
| ***Scrum***  Es la metodología ágil más popular y empleada en el desarrollo de *software*. Está construida por procesos iterativos que se separan en fases regulares que se van incrementado de forma periódica.  El propósito principal de la metodología *Scrum* es la satisfacción de las necesidades de los clientes, basándose en los principios de transparencia, avances permanentes y responsabilidad de equipo. A diferencia de otras metodologías ágiles tiene varios roles para lograr entregas y resultados inmediatos. |
| **Roles del equipo *Scrum***   * **Dueño del producto o cliente:** este rol es el responsable de entregar los requisitos del sistema; además, tiene la función de priorizar los pendientes durante el desarrollo del producto y presentar comentarios, para mejorar la calidad del *software*. * **Scrum Master:** su rol se encarga de entrenar y ayudar al equipo de desarrollo, a permanecer centrado en las metas del proyecto. También elimina los impedimentos que surgen en el proceso para garantizar que los objetivos establecidos se lleven a cabo. * **Equipo de desarrollo:** es el grupo de especialistas que cuenta con las habilidades necesarias para ejecutar el proyecto y alcanzar las metas propuestas en cada uno de los *sprints*. |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Carrusel de tarjetas | |
| **Introducción** | Los roles descritos dentro de la metodología Scrum se pueden entender de mejor forma con una analogía de la vida cotidiana, donde se tiene un auto de carreras que es el equipo de desarrollo. El cliente es el conductor que acelera y determina la orientación y destino, y el Scrum Master es el mecánico que realiza los mantenimientos y ajustes para que funcione correctamente.  Fases de la metodología Scrum: | |
| Conjunto de banners planos scrum agile 2  Imagen:228131\_i1849 | | |
| **Sprint**: son lapsos de tiempo, por lo general de dos a cuatro semanas, que se imponen para cumplir con las metas propuestas. | | Gente de negocios que trabaja en la gestión de tareas  Imagen:228131\_i1850 |
| **Planificación del Sprint**: son sesiones donde se establecen los flujos de trabajo y metas que se asignarán a los equipos de desarrollo. | | Concepto de gestión del tiempo para landing page  Imagen:228131\_i1851 |
| **Reuniones diarias**: sesiones cortas de aproximadamente 20 minutos que se realizan para hacer seguimiento a los avances del Sprint y programar nuevas metas para el siguiente día. | | Equipo de negocios discutiendo ideas para iniciar  Imagen:228131\_i1852 |
| **Revisión del Sprint:** son sesiones que se realizan al final de cada Sprint para constatar si las metas propuestas se cumplieron o no. | | Ilustración del concepto de método scrum  Imagen:228131\_i1853 |
| **Retrospectiva del Sprint:** sesiones que realiza el equipo de desarrollo cuando termina un Sprint para analizar los errores cometidos y mejorar en el próximo Sprint. | | Personas que inician un proyecto empresarial  Imagen:228131\_i1854 |

|  |  |
| --- | --- |
| **Tipo de recurso** | Infografía estática |
| **Texto introductorio** | La constante evolución del desarrollo de *software* requiere metodologías que se ajusten a los altos estándares de rendimiento y competitividad del mercado. La metodología Scrum es una excelente alternativa para resolver situaciones donde son prioritarias las entregas en corto plazo. |
| **Imagen**  Diagrama  Descripción generada automáticamente con confianza baja | |
| **Código de la imagen** | Imagen:228131\_i1855 |

## Prueba ágil

|  |  |
| --- | --- |
| **Tipo de recurso** | Tarjetas Conectadas |
| **Introducción** | Una prueba ágil es una práctica del *testing* de *software* donde se siguen los fundamentos del desarrollo ágil y se busca que las pruebas se adapten a la organización. En este enfoque se integran los roles del equipo de trabajo y se adiciona el rol de *tester*, el cual tiene como propósito la ejecución de las pruebas de *software* para la verificación de los requisitos que solicita el cliente.  Las pruebas ágiles se basan en los siguientes principios: |
| Ilustración de concepto de prueba de código de software  Imagen:228131\_i1856 | |
| Soporte técnico, programación y codificación  Imagen:228131\_i1857 | **Testing continuo**  Se deben realizar pruebas constantes para garantizar la calidad del *software* a través de cada uno de los ciclos de entrega. |
| Retroalimentación comentario encuesta soporte respuesta barra palabra  Imagen:228131\_i1858 | **Retroalimentaciones periódicas**  Contribuye para que el equipo de trabajo logre las metas establecidas en el proyecto de *software*. |
| Imagen:228131\_i1859 | **Compromiso de equipo**  En un *testing* ágil los desarrolladores, diseñadores, el cliente y en general todos los que intervienen en el proyecto son responsables por las pruebas del *software,* a diferencia de la metodología tradicional donde solo es el equipo de *testing*. |
| Ilustración del concepto de prueba de velocidad  Imagen:228131\_i1860 | **Rapidez en las respuestas**  Los integrantes del proyectoson involucrados en cada ciclo de este y aportan con la retroalimentación en la disminución de los tiempos de respuesta. |
| Ilustración del concepto de taller de codificación  Imagen:228131\_i1861 | **Codificación limpia y simple**  La codificación sencilla permite que en las próximas iteraciones se encuentren los *bugs* más fácilmente y sus respectivas correcciones demanden menos tiempo. |
| Ilustración de concepto de archivos de texto  Imagen:228131\_i1862 | **Reducir la documentación**  Minimizar la documentación del producto es útil para ganar tiempo y enfocarse en las pruebas. |
| Ilustración del concepto de codificación manual  Imagen:228131\_i1863 | **Pruebas en caliente**  Las pruebas se realizan durante la implementación, a diferencia de la metodología tradicional donde se realizan luego de finalizar la codificación del producto. |

# Tipos de pruebas

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| Las pruebas de *software* son elementos que permiten medir indicadores como la funcionalidad, el *performance* y la experiencia del usuario sobre un producto. Estas se realizan de forma manual o automatizada en el ciclo de vida del desarrollo del *software*. Si bien existe una gran variedad de pruebas, a continuación se describen algunos de los tipos más utilizados en el *testing* de *software*: Laptop and software assisting in testing process, tiny people testers. Automated testing, automotive executed test, software auto tester concept. flat vector modern illustration  Imagen:228131\_i1864 | |

## Pruebas según conocimiento del *software*

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | | Pestañas o tabs Verticales |
| **Introducción** | | Criterio de clasificación que se basa en el conocimiento interno que tiene el *tester* sobre la aplicación, por lo tanto, es necesario que tenga habilidades de programación y conozca a detalle la estructura del diseño del *software*. Este tipo de pruebas se dividen en: de caja negra, de caja blanca y de caja gris. |
| Testing software concept. Application or website code test process.  Imagen:228131\_i1865 | | |
| **Pruebas de caja negra** | Las pruebas de caja negra implican desconocimiento de la estructura interna, la tecnología y arquitectura del *software* y, sin embargo, se conoce cuál es el comportamiento dadas las entradas y salidas de este. Las pruebas que se diseñen bajo este criterio, no necesitan información sobre lo que sucede dentro del sistema para que funcione, sino que se desarrollan con base en los modelos funcionales, los casos de usos, los requerimientos o historias de usuario. | |
| **Pruebas de caja blanca** | Se diseñan a partir de la estructura del programa y por ende el *tester* debe tener total conocimiento del código fuente, los componentes, los flujos de control, la base de datos, entre otros elementos de la aplicación. | |
| **Pruebas de caja gris** | Son las que se ejecutan con información parcial de la estructura del programa, ya que habrá ocasiones donde no se cuenta con acceso al código fuente, pero sí a la base de datos y viceversa. | |

## Pruebas según su finalidad

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 2 |
| **Introducción** | Esta clasificación del *testing* se centra fundamentalmente en el propósito de las pruebas, las cuales pueden ser de progresión, regresión, exploratorias, humo o sanidad. |
| Imagen:228131\_i1866 | |
| **Pruebas de regresión**  Se ejecutan de forma periódica dependiendo de algunas condiciones, como el lanzamiento de una nueva versión del *software,* donde tienen como objetivo verificar que las funcionalidades que estaban operando correctamente lo sigan haciendo en la nueva versión y, con ello, evitar el *“downgrade” o* devolución del *software* a una antigua versión. Este tipo de pruebas son candidatas para la automatización de los casos de pruebas, por ser periódicos y, por lo general, son los mismos que se han probado en el pasado. | |
| **Pruebas de progresión**  Las pruebas de progresiónson las que buscan probar nuevas funcionalidades, es decir, se centran en lo que no tenía el *software* en las versiones anteriores y garantizan la calidad del desarrollo del *software* que se está construyendo o también llamado *release*. | |
| **Testing exploratorio**  Ofrece mayor flexibilidad al *tester*, debido a que permite libertad de probar y explorar nuevos escenarios que surgen de su innovación. Este estilo o enfoque promueve el aprendizaje para la optimización continua de las pruebas. Por lo general, se generan nuevas pruebas a medida que se va probando el *software.* | |
| **Testing de humo**  El *testing* de humo o sanidad, son pruebas rápidas que se realizan en la etapa inicial del plan de pruebas para verificar, con prontitud, que la aplicación sea pertinente en sus salidas y asegurar que los requerimientos prioritarios se cumplan. | |

## 

## Pruebas según nivel

|  |  |
| --- | --- |
| **Tipo de recurso** | Rutas / Pasos. Verticales 1 |
| **Introducción** | Las pruebas según el nivel están clasificadas de acuerdo con las etapas del diseño del desarrollo del *software*, es decir, desde el análisis hasta el despliegue. En este tipo de pruebas las etapas están asociadas y colaboran entre sí para garantizar el éxito de estas.  Dentro de las pruebas por nivel se encuentran: componente, integración y sistema. A continuación, se detallan cada una de ellas. |
| Imagen:228131\_i1867 | |
| **Pruebas por componente** | También llamadas pruebas unitarias, son las que aseguran el correcto funcionamiento por separado de un módulo, una función, un método o una clase del sistema. Este tipo de pruebas son ejecutadas por el desarrollador que termina de construir esa parte pequeña de la aplicación y se caracterizan por la rapidez en la corrección de los errores, porque siendo el mismo desarrollador quien las ejecuta, se corrigen en los tiempos de este proceso. |
| ***Testing* de integración** | Comprueba que las interfaces entre los módulos se comuniquen correctamente. Este tipo de pruebas las realizan tanto desarrolladores como los *testers* y son de gran importancia para lograr un sistema funcional y completo. |
| **Pruebas de sistema** | Son las más generales porque se centran en verificar que la aplicación funcione en su totalidad y cumpla con los requisitos establecidos por el cliente. Este tipo de pruebas son realizadas mayormente por el *tester* y en menor medida por el usuario final. Se pueden incluir dentro del *testing* de caja negra. |

## Pruebas según funcionalidad

|  |  |
| --- | --- |
| **Tipo de recurso** | Tarjetas animadas |
| **Introducción** | Las pruebas según su funcionalidad se pueden clasificar en las categorías de funcionales y no funcionales. |
| IlustraciÃ³n de prueba de software  Imagen:228131\_i1868 | **Pruebas funcionales**  Se realizan cuando se requiere verificar que los elementos críticos del negocio se ajustan con las funcionalidades del *software* y operan de acuerdo con lo esperado, sin presentar errores. Por lo tanto, las pruebas funcionales se definen a partir de lo que el sistema debe hacer, por ejemplo, el *software* debe imprimir una factura al final de la venta.  Con este tipo de pruebas se pueden revisar los elementos con los que el usuario tiene algún tipo de interacción. Dentro de las pruebas funcionales se encuentran: las unitarias, las de integración, las de regresión, las de sistema, las de sanidad, entre otras. |
| Concepto de evaluaciÃ³n comparativa idea de mejora del desarrollo empresarial comparar empresas competidoras de calidad  Imagen:228131\_i1869 | **Pruebas no funcionales**  Son atributos de la calidad que se encargan de verificar requisitos cuando el *software* está en operación y no sus funcionalidades. Con estas pruebas se mide cuáles son los límites o capacidad que soporta la aplicación, el rendimiento que tiene con altas cargas de trabajo, su escalabilidad y portabilidad entre otras. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 1 |
| **Introducción** | A continuación, se describen las clases de pruebas no funcionales más utilizadas en el *testing* de *software.* |
| Imagen:228131\_i1870 | |
| **Pruebas de portabilidad**  Se encargan de verificar cómo se comporta la aplicación en diferentes plataformas con distintas configuraciones de *hardware y software*. Por ejemplo, las pruebas para aplicaciones de escritorio y móviles se realizan en arquitecturas diferentes de sistemas operativos. El objetivo de estas es garantizar que el *software* pueda funcionar en todas las plataformas. | |
| **Pruebas de seguridad**  Con esta clase de pruebas se localizan las vulnerabilidades que al ser explotadas permiten el acceso con altos privilegios a funcionalidades no autorizadas de la aplicación. También se conocen como Pentesting o Ethical Hacking*.* Su objetivo es el de garantizar la confidencialidad, disponibilidad e integridad del *software*. | |
| **Pruebas de rendimiento**  Simulan diferentes niveles de carga para evaluar el *performance* del sistema y su comportamiento cuando se presentan picos y cuellos de botella en sus flujos de información. Los niveles de carga son simulados con base en la cantidad de usuarios que posiblemente hagan uso de la aplicación, es decir, se puede realizar una prueba con un total de 5 usuarios y otra con 1000 usuarios concurrentes. Cabe resaltar que esto depende de los requisitos del producto. | |
| **Localización**  Estas pruebas garantizan que la aplicación funcione para una región en particular, por lo tanto, es importante que pueda adaptarse a distintas zonas geográficas y verificar variables como el tipo de moneda, fechas, traducción de lenguaje, entre otras. | |
| **Usabilidad**  Se centran en la facilidad de la aplicación para su aprendizaje y uso. Estas clases de pruebas son muy complejas debido a la subjetividad, ya que dependen de la percepción de cada usuario del sistema y, por lo tanto, no hay criterio unificado en los resultados. | |
| **Accesibilidad**  Estas pruebas pretenden que el sistema sea usado por todos los usuarios independientemente de sus habilidades técnicas, físicas o cognitivas. Por ejemplo, en el diseño *software* se debe tener especial cuidado y pensar en los posibles usuarios daltónicos para que las interfaces no incluyan colores que sean confusos para ellos. | |

|  |
| --- |
| Cuadro de texto |
| Para reforzar el aprendizaje de este tema, se recomienda ver el siguiente video:  Video temático 228131\_v02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de recurso** | **VIDEO TEMÁTICO** | | | |
| **NOTA** | **La totalidad del texto locutado para el video no debe superar las 500 palabras aproximadamente** | | | |
| **Título** | **Pruebas automatizadas de *software*** | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración (voz en off)** | **Texto** |
| **1** | Interfaz de usuario gráfica  Descripción generada automáticamente  228131\_i1890  Imagen de referencia: <https://as2.ftcdn.net/v2/jpg/04/90/33/39/1000_F_490333979_EQdf0tNa0W7MdstKjrGwnjQjoxfYP2zO.jpg> | NA | Las pruebas automatizadas son construidas a través de *scripts* y *software* especializado, simulando las acciones del usuario final. | Pruebas por Script |
| **2** | 228131\_i1891  Imagen de referencia:  <https://www.freepik.es/vector-premium/concepto-diseno-ilustracion-plana-actualizacion-software-ilustracion-sitios-web-paginas-destino-aplicaciones-moviles-carteles-pancartas_19163382.htm> | NA | Las pruebas se ejecutan automáticamente en un servidor de pruebas, portátil o un sistema de integración. | Se ejecutan automáticamente |
| **3** | Interfaz de usuario gráfica  Descripción generada automáticamente  228131\_i1892  Imagen de Referencia:  <https://www.shutterstock.com/image-photo/fast-internet-connection-metaverse-technology-concept-2142756045> | NA | En las pruebas automatizadas la ejecución de casos es rápida y eficiente, brindando un mayor grado de confiablidad y precisión. | Ejecución rápida |
| **4** | Seguimiento de proyectos, finalización de tareas o lista de comprobación para recordar el concepto de progreso del proyecto, administrador de proyectos de negocios con un lápiz grande para comprobar las tareas completadas en el calendario de administración de proyectos. vector plano  228131\_i1893  Imagen de referencia:  <https://www.shutterstock.com/image-vector/project-tracking-task-completion-checklist-remind-2117356745> | NA | Estas pruebas son adecuadas para proyectos grandes porque tienen un mayor retorno de la inversión o ROI y, por lo tanto, ayudan a reducir costos en pruebas y detección de *bugs.* | Aplican para grandes proyectos |
| **5** | 228131\_i1894  Imagen de referencia:  <https://www.shutterstock.com/image-illustration/application-release-orchestration-aro-automation-ara-2122358591> | NA | Gracias a sistemas orquestadores como *Jenkins*, *Bamboo*, *Microsoft* *Azure*, entre otros, estas pruebas se pueden ejecutar en cualquier momento de forma automática sin intervención humana. | Sistemas orquestadores |
| **6** | 228131\_i1895  Imagen de referencia:  <https://as2.ftcdn.net/v2/jpg/04/10/34/39/1000_F_410343906_ypWFxtTDG2eP02NUBgL27Z0XPRnq6irv.jpg> | NA | Las pruebas automatizadas se pueden reusar y ejecutar una y otra vez con altos niveles de cobertura o también se pueden programar para que se disparen cuando detecten un cambio en el código de la aplicación que se está desarrollando. | Pruebas reutilizables |
| **7** | 228131\_i1896  Imagen de referencia:  <https://www.freepik.es/foto-gratis/primer-plano-empresario-trabajando-casa-haciendo-calculos_10499865.htm#query=Pruebas%20software%20Alto%20Costo&position=39&from_view=search&track=ais> | NA | Sin embargo, estas generalmente tienen un alto costo de implementación y requieren conocimiento técnico en la manipulación de herramientas especializadas. | Alto costo |
| **8** | Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico  Descripción generada automáticamente  228131\_i1897  Imagen (Elaboración Propia):  <https://1drv.ms/u/s!AmN5mLAAnCdygcNZdT0LyaBfoOftGA?e=xX92XY> | NA | *Selenium* IDE es una de las herramientas más famosas para automatizar pruebas en las aplicaciones web. Su uso es sencillo porquese instala como una extensión de *Google Chrome* o *Mozilla Firefox*. | Selenium IDE |
| **9** | Interfaz de usuario gráfica, Aplicación  Descripción generada automáticamente  228131\_i1898  Imagen (Elaboración Propia).  Sugerencia: Hacer **Zoom** en la parte superior derecha donde está ubicado el icono “REC”.  <https://1drv.ms/u/s!AmN5mLAAnCdygcNbup_VYIQTPA86BQ?e=n7PpoN> | NA | Para empezar a hacer pruebas con *Selenium* IDE, basta con dar clic en el icono que tiene la etiqueta “REC” e ingresar la URL de la página web que se quiere probar. | Ingresar URL |
| **10** | 228131\_i1899  Imagen (Elaboración Propia)  <https://1drv.ms/u/s!AmN5mLAAnCdygcNclSWD-VrWc8Uqlw?e=7n8EPG> | NA | Por ejemplo, al seleccionar un sitio web diseñado para pruebas como *OpenCar*t, se pueden grabar a través de Selenium IDE todos los procesos e interacciones que se realizan en la página web. | Grabar interacciones de la página Web. |
| **11** | 228131\_i18100  Imagen (Elaboración Propia)  <https://1drv.ms/u/s!AmN5mLAAnCdygcNdpiHpeh311n2qTw?e=voUiqk> | NA | Luego de realizar las pruebas en el sitio web se detiene la grabación y se guarda el caso de prueba. | Detener y guardar prueba |
| **12** | Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico  Descripción generada automáticamente  228131\_i18101  Imagen (Elaboración Propia)  Recomendación: Hacer **énfasis** en el centro de la imagen dónde encuentra la columna: command, Target y Value.  <https://1drv.ms/u/s!AmN5mLAAnCdygcNew9bPNuF3v7oBDg?e=39eJJo> | NA | Con el caso de prueba almacenado en la herramienta se pueden aplicar los principios de las pruebas automatizadas como la reutilización de las interacciones grabadas para variar los parámetros ingresados sin tener que entrar nuevamente al sitio web. | Reutilizar prueba grabada |
| **13** | 228131\_i18102  Imagen de referencia:  <https://www.freepik.es/vector-gratis/aumento-velocidad-carga-navegador-internet-rapido-tecnologia-linea-moderna-tiempo-descarga-acelerado-optimizacion-rendimiento-telefono-inteligente-ilustracion-concepto-mejora_11668409.htm#query=Aumento%20de%20velocidad&position=39&from_view=search&track=sph> | NA | Aplicando esta técnica se aumenta la velocidad de las pruebas por cuanto se pueden guardar los formularios y solo variar el campo que se considera crítico. | Aumento de velocidad |
| **14** | 228131\_i18103  Imagen de referencia:  <https://www.shutterstock.com/image-vector/tiny-programmers-programming-website-internet-platform-1821632519> | NA | *Selenium* IDE es una herramienta *open source* que no tiene ningún costo al utilizarse, además se puede integrar con los lenguajes de programación potentes como *Java, Python, Ruby, C#*, entre otros. Al ser una aplicación web, es fácil de instalar. | Herramienta web open source |
| **15** | 228131\_i18104  Imagen de referencia:  <https://www.shutterstock.com/image-vector/white-background-colorful-circular-frame-tech-694584718> | NA | Para entregas continuas es fundamental la automatización de pruebas, porque permite publicar nuevas versiones de código de forma inmediata para los usuarios. La publicación automática no es posible si existe un paso manual que interfiera en el proceso. | Publicar versiones inmediatas |
| **16** | 228131\_i18105  Imagen de referencia:  <https://www.shutterstock.com/image-vector/sustainable-communities-environmental-house-building-tiny-2186021769> | NA | La ejecución en diferentes contextos y escenarios automatizados de pruebas facilita un mayor rango y precisión en los diferentes flujos que tiene la aplicación. | Diferentes escenarios |
| **17** | 228131\_i18106  Imagen de Referencia:  <https://www.shutterstock.com/image-vector/document-management-form-check-list-concept-2079696961> | NA | Las pruebas automatizadas retornan evidencias del funcionamiento del *software*, dependiendo de cada una de las opciones que los usuarios utilicen. | Retornan evidencias de las opciones |
| **18** | 228131\_i18107  Imagen de referencia:  <https://www.shutterstock.com/image-photo/mobile-phone-personal-data-cyber-security-1452139694> | NA | La implantación de pruebas automatizadas en las organizaciones que desarrollan *software* otorga ahorros importantes en el mediano plazo, porque logra crear productos seguros y confiables para sus usuarios. | Ahorro, productos seguros y confiables |
| **Nombre del archivo** | 228131\_v02 | | | |

## Estrategias de pruebas

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| Una estrategia de pruebas de *software,* agrupa los métodos de elaboración de los casos de pruebas en una sucesión de pasos bien definidos, para construir una aplicación correctamente. Los procesos de verificación y validación del producto, tienen como hoja de ruta, las estrategias de pruebas. (Solarte, 2019)  Imagen:228131\_i1871 | |
| En las estrategias de pruebas se tiene en cuenta que:   * Las pruebas inician desde el interior hacia afuera, comenzando con la verificación de los módulos o métodos y finalizando con la validación de los requisitos y funcionalidades del *software.* * Es pertinente que en los puntos críticos del *software* se realicen distintos casos de pruebas que permitan garantizar la calidad de este. * El *testing* se debe realizar por los especialistas que construyen el *software* o por los equipos de *tester* que integran el proyecto. * El *testing* y la depuración son procesos para encontrar *bugs* y corregirlos; sin embargo, la depuración se ejecuta por el desarrollador en tiempo de codificación del *software* y el *testing* y, además de contener estas pruebas, abarca todas las verificaciones y validaciones del producto. * Están formadas por pruebas de bajo nivel y alto nivel, donde las de bajo nivel se relacionan con el código fuente y las de alto nivel con las funcionalidades del *software*. | |

# Diseñar pruebas

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| Teniendo en cuenta que las pruebas de *software* no pueden probar todo y que las pruebas exhaustivas no son posibles, es pertinente que las pruebas aplicadas se acerquen a los estándares de calidad y sean eficientes, de modo que las técnicas que se ejecuten en el *testing de software,* maximicen la posibilidad de encontrar *bugs* y minimicen la cantidad de pruebas aplicadas. Algunas de las técnicas más usadas son: particiones de equivalencia, valores límites y análisis de robustez, diagrama de transición de estados y tablas de decisión. Pruebas comparativas. software de evaluación comparativa, indicador de rendimiento del producto, pruebas de carga, características de rendimiento, prueba de productos competitivos.  Imagen:228131\_i1872 | |

## Particiones de equivalencia

|  |  |
| --- | --- |
| **Tipo de recurso** | Rutas / Pasos. Verticales 1 |
| **Introducción** | Esta técnica parte del concepto matemático de conjuntos, donde los conjuntos contienen varios subconjuntos que cumplen con dos condiciones: la primera, que la intercesión de dos subconjuntos sea nula, es decir, los criterios de uno no se repiten en el otro y, la segunda, que la sumatoria de todos los subconjuntos o particiones den como resultado el conjunto total.  Las ventajas que supone esta técnica de pruebas son las siguientes: |
| Imagen:228131\_i1873 | |
| **Botón 1** | **Disminución de casos de pruebas**  Como las pruebas están agrupadas en particiones, permiten minimizar los casos de pruebas, ya que basta con testear un elemento o representante de la partición para obtener resultados que identifican a todos los elementos. |
| **Botón 2** | **Detección de inconsistencia**  Cuando seclasifican las pruebas en los subconjuntos, se pueden identificar pruebas que no encajan en ninguna de las particiones, por tanto, ayuda a detectar errores en las especificaciones o requisitos. |
| **Botón 3** | **Funcionales en el plan de pruebas**  Son útiles para identificar la terminación de una prueba, es decir, dentro del plan de pruebas se puede implementar un criterio de finalización cuando se ejecute al menos un elemento de la partición de equivalencia. |

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| La desventaja principal de esta técnica es que no permite combinar particiones entre sí, lo que genera limitantes a la hora de cruzar elementos que guardan relación con otros, pero se encuentran en distintas particiones. | |

## Valores límites y análisis de robustez

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Pestañas o tabs horizontales | |
| **Introducción** | La técnica de valores límites y análisis de robustez, centra su enfoque en los límites de las particiones e implica testear las clases de equivalencia en los bordes de las particiones, porque, estadísticamente, en estos bordes es donde se presenta la más alta la probabilidad de encontrar la mayoría de los *bugs*. | |
| **Errores en la definición de**  **valores límites** | Los desarrolladores tienden a pasar por alto errores en la definición de valores límites al establecer condiciones como mayor que, menor que, mayor o igual que y menor o igual que. | Imagen:228131\_i1874 |
| **Ejemplo de aplicación** | Un ejemplo de la aplicación de esta técnica puede ser en el *testing* de un módulo donde se verifica que la edad de una persona está entre el rango de 18 a 25 años, para asignar una promoción y con ello un descuento. Si el requerimiento especifica que el rango de edad es entre 18 y 25 años, el código fuente de la aplicación, deberá contener las operaciones de mayor o igual que, y menor o igual que. En caso de no contener dichas restricciones, se generarán *bugs* que tendrán que ser corregidos. | Imagen:228131\_i1875 |
| **Análisis de robustez** | Se fija en los extremos de las clases de equivalencias, pero llevando las pruebas al extremo. Esto es de gran utilidad para aquellos módulos donde es importante el análisis de valores numéricos con decimales, ya sean valores muy grandes o pequeños. En el ejemplo de la edad visto anteriormente, se puede aplicar un análisis de robustez si en las especificaciones se establece que las promociones se darán si el usuario está muy cerca a cumplir el rango de edad establecido, por ejemplo, si tiene más de 17 años, 9 meses y hasta 25 años, 1 mes. | Imagen:228131\_i1876 |

## Diagrama de transición de estados

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| Esta técnica representa los diferentes estados por los que un objeto cambia en una aplicación, bien sea por un proceso o por un producto. Los diagramas que se crean bajo esta técnica, están formados por estados, eventos y transiciones. Por ejemplo, en un sistema que hace seguimiento a los pedidos de los clientes, se aplica un diseño de pruebas con la técnica de transición de estados, debido a que el pedido pasará por varias condiciones a lo largo del proceso, como: ingresado, en preparación, despachado y entregado. | |
| **Figura 1** *Ejemplo de transición de estados*    Imagen:228131\_i1877  Los diagramas de transición de estados son fundamentales en el *testing* de *software* porque permiten entender cuáles son las transiciones válidas e inválidas en el sistema. Por ejemplo, si una aplicación que gestiona pedidos pasa de un estado “ingresado” a “despachado”, se identifica un *bug*, porque previamente deberá pasar por el estado “en preparación”. | |

## Tablas de decisión

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| La técnica de tablas de decisión resuelve el inconveniente que tienen las técnicas de particiones de equivalencia y análisis de robustez, donde no se permite el *testing* entre las interacciones de los subconjuntos del sistema. En términos generales, una tabla de decisión es una combinación de condiciones, reglas y acciones, las cuales se pueden representar en una matriz, para su mayor compresión.  **Tabla 1** *Tabla de decisión*   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Condiciones** | **Regla 1** | **Regla 2** | **Regla 3** | **Regla 4** | | Condición 1 | Valor 1.1 | Valor 2.1 | Valor 3.1 | Valor 4.1 | | Condición 2 | Valor 1.2 | Valor 2.2 | Valor 3.2 | Valor 4.2 | | Condición 3 | Valor 1.3 | Valor 2.3 | Valor 3.3 | Valor 4.3 | | **Acciones** |  |  |  |  | | Acción 1 | x | x |  | x | | Acción 2 |  | x | x | x | | |
| Para mayor comprensión sobre la implementación de la técnica de tablas de decisión, se expone el siguiente ejemplo: una empresa donde todos los empleados reciben algún tipo de incentivo; sin embargo, aquellos que tienen más de 5 años de antigüedad acceden a la devolución de los gastos de viáticos que tengan durante el mes, por realizar su labor. Además, los empleados que poseen un cargo directivo, accederán también al pago del estacionamiento durante la jornada de trabajo, aún si tienen o no, los 5 años de antigüedad.  **Tabla 2** *Ejemplo de una tabla de decisión caso práctico*   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Condiciones del empleado** | **Regla 1** | **Regla 2** | **Regla 3** | **Regla 4** | | Antigüedad > 5 años | Sí | No | No | Sí | | Cargo directivo | No | No | Sí | Sí | | **Acciones de la empresa** |  |  |  |  | | Devolución de viáticos | x |  |  | x | | Estacionamiento gratis |  |  | x | x |   Como se evidencia, la complejidad estaba en el enunciado, pero luego de sistematizarlo como una tabla de decisión, las cuatro reglas se convierten en casos de prueba del sistema, que garantizarán la validación de los requisitos expuestos en el problema. | |

|  |
| --- |
| 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. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Síntesis |
|  | |
| **Introducción** | El siguiente mapa integra los criterios y especificidades de los conocimientos expuestos en el presente componente formativo. |
| Imagen:228131\_i1878  Enlace para ver maximizado: [Pruebas y Calidad de Software - MindMeister Mapa Mental](https://www.mindmeister.com/es/map/2359636500?t=HDpmq8adaR) | |

**Actividad didáctica**

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Actividad didáctica. Opción múltiple | |
| Apreciado aprendiz, a continuación se encuentra una actividad para afianzar los conceptos y aprendizajes del componente de formación sobre Pruebas y Calidad de *Software*.  Antes de su realización, se recomienda la lectura de este componente formativo. Esta actividad no es calificable y puede realizarse todas las veces que se requiera para afianzar el conocimiento.  Lea atentamente las preguntas y seleccione, para cada una de ellas, la respuesta que considere correcta, entre las opciones que se presentan. | | Imagen:228131\_i1879 |
| 1. ¿Cuál de las siguientes afirmaciones es verdadera sobre el proceso de *debugging*? | | Imagen:228131\_i1880 |
| *Debugging* es una de las actividades que se realiza en el *testing*. | | *Debugging* es el proceso de identificación y corrección de errores. (Correcta) |
| El *debugging* permite que el probador de software sea el más técnico del equipo. | | *Debugging* es una metodología para construir software |
| **Retroalimentación opción correcta**: ¡Excelente! *Debuggin*g es el proceso de identificación y corrección de errores que realiza el desarrollador en la codificación de la aplicación. | | **Retroalimentación opción incorrecta**: se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 2. ¿Cuál de las siguientes afirmaciones pertenece a uno de los principios del *testing*? | | Imagen:228131\_i1881 |
| El *testing* aporta a la calidad del *software*, pero no contribuye en la satisfacción del usuario. | | Las pruebas de *software* deben realizarse lo más tarde posible. |
| Las pruebas de *software* son para verificar y validar de forma dinámica el cumplimiento de los requisitos del sistema. (Correcta) | | Las pruebas de *software* son generales y pueden aplicarse en cualquier escenario o contexto. |
| **Retroalimentación opción correcta**: ¡Excelente! La verificación y validación son principios fundamentales del *testing* | | **Retroalimentación opción incorrecta**: se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 3. ¿Cuál de las siguientes afirmaciones no es una ventaja de la implementación de metodologías ágiles en la construcción del *software*? | | Imagen:228131\_i1882 |
| Elevar la velocidad y la eficiencia de los grupos de desarrollo de las organizaciones. | | Alertar a los *testers* sobre las fallas de forma oportuna y evitar retrocesos en la codificación. |
| Propiciar interacción constante con el usuario para generar retroalimentaciones y mejorar la calidad del *software*. | | Entregas al final del desarrollo del *software* que permiten poca interacción con el usuario. (Correcta) |
| **Retroalimentación opción correcta**: ¡Excelente! Las entregas al final son propias de metodologías tradicionales como la metodología en cascada. | | **Retroalimentación opción incorrecta**: se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 4. ¿Cuál de las siguientes fases o etapas no se encuentra definida dentro de la construcción del *software*? | | Imagen:228131\_i1883 |
| implementación | | Organización (Correcta) |
| Diseño | | *Testing* |
| **Retroalimentación opción correcta**: ¡Excelente! La organización no es una fase del desarrollo del *software*. | | **Retroalimentación opción incorrecta**: se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 5. ¿Cuál de las siguientes metodologías no es una metodología ágil? | | Imagen:228131\_i1884 |
| *Kanban* | | *Scrum* |
| Programación extrema | | Programación estructurada (Correcta) |
| **Retroalimentación opción correcta:** ¡Excelente! La metodología estructurada o tradicional no hace parte de las metodologías ágiles. | | **Retroalimentación opción incorrecta:** se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 6. ¿Cuál de las siguientes afirmaciones no es un principio del *testing* ágil? | | Imagen:228131\_i1885 |
| Los integrantes del proyecto son involucrados en cada ciclo y aportan con la retroalimentación en la disminución de los tiempos de respuesta. | | Las pruebas se realizan durante la implementación, a diferencia de la metodología tradicional donde se realizan luego de finalizar la codificación del producto. |
| Se realizan pruebas constantes para garantizar la calidad del *software* a través de cada uno de los ciclos de entrega | | La retroalimentación se hace una vez, cuando finaliza la codificación del producto. (Correcta) |
| **Retroalimentación opción correcta:** ¡Excelente! Las retroalimentaciones en la metodología ágil, se realiza en todas las etapas de la construcción del *software*. | | **Retroalimentación opción incorrecta:** se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 7. ¿Cuál de los siguientes principios no es un objetivo de las pruebas de seguridad? | | Imagen:228131\_i1886 |
| Integridad | | Disponibilidad |
| Confidencialidad | | Publicidad (correcta) |
| **Retroalimentación opción correcta:** ¡Excelente! La publicidad no es un principio de las pruebas de seguridad. | | **Retroalimentación opción incorrecta:** se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 8. ¿Cuál es el objetivo de las pruebas de portabilidad? | | Imagen:228131\_i1887 |
| Garantizar la confidencialidad, disponibilidad e integridad del *software*. | | Evaluar el performance del sistema y su comportamiento cuando se presentan picos y cuellos de botella en sus flujos de información. |
| Garantizar que el software pueda funcionar en todas las plataformas. (Verdadero) | | Aumentar la facilidad de la aplicación para su aprendizaje y uso. |
| **Retroalimentación opción correcta:** ¡Excelente! Las pruebas de portabilidad garantizan que el *software* sea multiplataforma. | | **Retroalimentación opción incorrecta:** se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 9. ¿Cuál es una ventaja de la técnica de particiones de equivalencia? | | Imagen:228131\_i1888 |
| Permiten detectar inconsistencias (Verdadero) | | Permiten entender cuáles son las transiciones válidas e inválidas en el sistema |
| Permiten combinar particiones entre sí. | | Permiten el análisis de valores numéricos con decimales, ya sean valores muy grandes o pequeños. |
| **Retroalimentación opción correcta:** ¡Excelente! La técnica de particiones de equivalencia permite encontrar inconsistencias en las aplicaciones. | | **Retroalimentación opción incorrecta:** se recomienda volver a revisar el componente formativo e intentar nuevamente. |
| 10. ¿Cuál es el inconveniente de las particiones de equivalencia que resuelven las tablas de decisión? | | Imagen:228131\_i1889 |
| El *testing* entre las interacciones de los subconjuntos del sistema. (Verdadero) | | La seguridad en los subconjuntos del sistema. |
| Detección de inconsistencias. | | El análisis de valores numéricos con decimales. |
| **Retroalimentación opción correcta:** ¡Excelente! Las tablas de decisión permiten el *testing* entre subconjuntos del sistema. | | **Retroalimentación opción incorrecta:** se recomienda volver a revisar el componente formativo e intentar nuevamente. |

**Material complementario**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de recurso** | Material complementario | | |
| **Tema** | **Referencia APA del material** | **Tipo** | **Enlace** |
| Introducción a las pruebas de software | Verona, S., Delgado, M., Güemes, A. & Castillo, M. (2022). Good practices for software test automation. Case study in an electronic store. *Revista Cubana de Ciencias Informáticas*, *16*(2), p. 101–117. | Artículo | <https://search-ebscohost-com.bdigital.sena.edu.co/login.aspx?direct=true&db=fap&AN=158595501&lang=es&site=ehost-live> |
| Introducción a las pruebas de software | Piattini M., García, F., Pino, F. & García, I. (2020). *Calidad de sistemas de información.* Editorial Rama. | Libro | <https://www.alphaeditorialcloud.com/library/publication/calidad-de-sistemas-de-informacion-5a-edicion-ampliada-y-actualizada> |
| Introducción a las pruebas de software | Subra, J. (2020). *Scrum: un método ágil para sus proyectos*(2ª ed.). Ediciones ENI. | Libro | <https://sena-primo.hosted.exlibrisgroup.com/permalink/f/q6j6k0/sena_biblioteca_eniDPT3SCRU> |
| Introducción a las pruebas de software | Casanovas, I., Rozo, M. & Pollo, C. (2020). Ontología para transferir conocimiento en la etapa de pruebas de software. *Revista de las Personas y Tecnología*, *13*(39), p. 69-92. | Artículo | <https://sena-primo.hosted.exlibrisgroup.com/permalink/f/1i756fj/TN_cdi_dialnet_primary_oai_dialnet_unirioja_es_ART0001444710> |
| Tipos de pruebas | Páez, J., Cortes, J., Simanca, F. & Blanco, F. (2021). Aplicación de UML y SCRUM al desarrollo del software sobre control de acceso*. Información Tecnológica*, *32*(5), p. 57–66. | Artículo | <https://doi-org.bdigital.sena.edu.co/10.4067/S0718-07642021000500057> |

**Glosario**

|  |  |
| --- | --- |
| **Tipo de recurso** | Glosario |
| *Debugging:* | proceso que implanta puntos de interrupción o *breakpoint,* para analizar el estado del *software* en una parte dada del código. |
| Programación extrema: | agrupación de reglas que generan agilidad y resiliencia a las organizaciones en la ejecución de los proyectos de *software* en contextos que son muy variables. |
| Pruebas funcionales: | verificación de los elementos críticos del negocio, se ajustan con las funcionalidades del *software* y operan de acuerdo con lo esperado sin presentar errores. |
| Pruebas no funcionales: | atributos de la calidad que se encargan de verificar requisitos cuando el *software* está en operación y no sus funcionalidades |
| Sprint: | lapsos de tiempo, por lo general de dos a cuatro semanas, que se imponen para cumplir con las metas propuestas. |
| Tabla de decisión: | combinación de condiciones, reglas y acciones que se pueden representar en una matriz para mayor comprensión. |
| *Testing:* | mecanismo para la verificación y validación de las funcionalidades de los programas o aplicaciones de *software* con el fin de ofrecer un producto sin defectos o errores. |
| *Testing* de humo: | pruebas rápidas que se realizan en la etapa inicial del plan de pruebas para verificar con prontitud que la aplicación sea pertinente en sus salidas y asegurar que los requerimientos prioritarios se cumplan. |
| validación: | proceso que compara el *software* que está en construcción con los modelos definidos en los requerimientos a través de sus casos de usos o historias de usuario. |
| verificación: | proceso para controlar que el producto en desarrollo haga lo que tiene que hacer. |

**Referencias bibliográficas**

|  |  |
| --- | --- |
| **Tipo de recurso** | Bibliografía |
| Alfaro, A. y Carrizo, D. (2021). *Método de aseguramiento de la calidad en una metodología de desarrollo de software: un enfoque práctico*. Universidad de Tarapacá. <https://www-virtualpro-co.bdigital.sena.edu.co/descarga/metodo-de-aseguramiento-de-la-calidad-en-una-metodologia-de-desarrollo-de-software-un-enfoque-practico> | |
| IBM. (2022). *¿Qué es la prueba de software?* <https://www.ibm.com/es-es/topics/software-testing> | |
| Pérez, Y., Gallegos, J., Zapata, S., Ccama, D. & Choque. R. (2020). Design Thinking en la planificación de pruebas de software. *Revista Innovación y Software*, *1*(2), p. 40-51. <https://sena-primo.hosted.exlibrisgroup.com/permalink/f/1i756fj/TN_cdi_doaj_primary_oai_doaj_org_article_49cd1beda5b84284be236b081b7bd5cf> | |
| Ramos, A., Pérez, H., Nava, S. & Martínez, F. (2022). Mic-agile: metodología ágil para micro-equipos de desarrollo de software. *Revista Ingenio*, *19*(1), p. 1-8. <https://doi.org/10.22463/2011642X.2779> | |
| Solarte, G., Castro, Y. & Muñoz, L. (2019). Planificación, gestión y control de la calidad del software. *Scientia Et Technica*, *24*(4), p. 611-617. <https://sena-primo.hosted.exlibrisgroup.com/permalink/f/1i756fj/TN_cdi_dialnet_primary_oai_dialnet_unirioja_es_ART0001364569> | |