**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

| PROGRAMA DE FORMACIÓN | Técnico en procesamiento de pruebas de *software* |
| --- | --- |

| COMPETENCIA | 220501099 - Probar la solución de acuerdo con parámetros técnicos y modelos de referencia. | RESULTADOS DE APRENDIZAJE | 220501099-02. Ejecutar las pruebas de *software* de acuerdo con los requisitos de operación y modelos de referencia. |
| --- | --- | --- | --- |

| NÚMERO DEL COMPONENTE FORMATIVO | 02 |
| --- | --- |
| NOMBRE DEL COMPONENTE FORMATIVO | Implementar prueba de *software* |
| BREVE DESCRIPCIÓN | En este módulo el aprendiz ejecutará y aplicará los artefactos de las pruebas de *software* sobre el producto, con lo cual tendrá la oportunidad de identificar activamente los diferentes tipos de prueba en acción, generando un informe sobre las pruebas exitosas y los errores o problemas encontrados. |
| PALABRAS CLAVE | CMII, informe, automatización, modelos de aplicación de pruebas, incidencias |

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

1. **TABLA DE CONTENIDOS:**

**Introducción**

**1. Pruebas de *software***

1.1. Causas que deterioran la calidad en el *software*

1.2. Modelo de desarrollo CMMI

1.3. Criterios del manejo de la confidencialidad

1.4. Herramientas tecnológicas para pruebas

1.5. Modelo de procesos de pruebas

1.6. Documentación de pruebas

1.7. *Continuous Integration* y *Continuous Delivery*

1.8. Depuración y técnicas de resolución de problemas

**2. Construcción, desarrollo y ejecución de pruebas de *software***

2.1. Aseguramiento de calidad de *software*

2.2. Implementación del plan de pruebas

2.3. Configuración del ambiente de pruebas

2.4. Ejecución de las pruebas

2.5. Creación de plan de pruebas e integración y entrega continua en una metodología ágil

**3. Gestión del desarrollo y ejecución de pruebas**

3.1. Mínimo producto viable

3.2. Aplicación de Scrum y Kanban

3.3. DevOps

3.4 Git, Gitflow y Githubflow

3.5. Gestión de pruebas

3.6. Gestión de incidencias

3.7. Incidencias funcionales

3.8. Incidencias no funcionales

1. **DESARROLLO DE CONTENIDOS:**

**Introducción**

Le invitamos a revisar el siguiente video para comprender algunas indicaciones iniciales del proceso de aprendizaje y los elementos de abordaje acerca de ejecución del plan de pruebas y sus generalidades:

Interfaz de usuario gráfica, Aplicación, PowerPoint

Descripción generada automáticamente

**1. Pruebas de *software***

Son un proceso para explorar, entender y conocer el producto que se está construyendo, en el cual se busca minimizar la cantidad de errores y evitar que de esta manera se repliquen en un ambiente productivo y se vean afectados los usuarios finales.

En este se definen diferentes técnicas para ser aplicadas en el producto, con el fin de garantizar el cumplimiento de los estándares definidos por la entidad desarrolladora y las funcionalidades especificadas por el cliente.

| Es importante identificar cuáles son las causas que pueden llevar a tener un *software* de mala calidad, entre los cuales podemos indicar malas prácticas de desarrollo, no entender el negocio (no identificar cuáles son las finalidades de las funciones del producto *software*), falta de personal calificado para realizar y ejecutar el plan de pruebas, falta de cultura orientada a la calidad de *software*, entre otras. |  |
| --- | --- |

Por ende, se recomienda tener en cuenta los siguientes puntos para establecer un plan de pruebas que permita elevar los estándares de calidad:

* Analizar los requisitos funcionales del producto *software*.
* Identificar en cada ciclo las nuevas funcionalidades a probar, o las que se deben volver probar (pruebas de regresión) dado que se ven afectadas por nuevas características.
* Establecer un inventario de todas las funcionalidades del sistema que se deben tener siempre en cuenta en las pruebas.
* Seleccionar estrategias para la ejecución y aplicación de pruebas.
* Seleccionar los criterios de inicio, aceptación y suspensión de pruebas.
* Seleccionar los ambientes y componentes requeridos.
* Seleccionar necesidades de personal y capacitaciones.
* Seleccionar la metodología y procedimientos para la aplicación de pruebas.
* Seleccionar el plan de pruebas.
* Tener en cuenta los riesgos y definir los planes de acción.

Estas pruebas buscan abarcar todos los aspectos involucrados desde personal, técnicas, metodologías y conocimientos, pero cada organización puede complementar con acciones de mejora continua adicionales.

**1.1. Causas que deterioran la calidad en el *software***

Las causas que afectan la calidad del *software* siempre están presentes, ya que son un atributo inevitable del proceso, con ello se tendrá el riesgo de tener un producto defectuoso con mala calidad.

Esto es reflejo de las malas prácticas desde la concepción del producto *software*, las acciones que pueden influenciar una mala calidad en el *software* o el deterioro de este pueden ser diversas, pero básicamente se podría indicar que se enfocan en los procesos de ejecución del desarrollo del producto de *software*, pues la mala ejecución genera un efecto en cascada donde se ven afectados el plan de desarrollo, el plan de pruebas, los tiempos de ejecución y los recursos invertidos.

Algunas de las prácticas para evitar construir un producto *software* con mala calidad pueden ser:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**1.2. Modelo de desarrollo CMMI**

Es un modelo de capacidad y madurez que posee dos vistas estructurales que permiten tener un enfoque distinto dependiendo de quién esté realizando la implementación; se centra en elementos esenciales de las prácticas y los procesos, el cual permite realizar una evaluación de la madurez del proceso de desarrollo de *software* en las organizaciones.

Los objetivos de CMMI:

* Producción de calidad de productos o servicios.
* Creación de valor para los accionistas.
* Mejora continua en la satisfacción de los clientes.
* Aumento en la cuota de mercado
* Obtener reconocimiento de excelencia en la industria

El modelo estructurado por nivel de madurez es un nivel de madurez bien definido, que entrega las bases necesarias para evolucionar hacia un proceso de *software* maduro, proporcionando en cada nivel aspecto de mejora continua que se clasifican en:

* Inicial.
* Gestionado.
* Definido.
* Cuantitativamente gestionado.
* Optimizado.

Los niveles de madurez consisten en un conjunto de áreas de proceso predefinidas que se miden por objetivos generales y específicos, aplicándolos a cada conjunto predefinidos de áreas de proceso.

1. **Nivel de madurez inicial:** los procesos suelen ser caóticos, en el cual se gestionan y crean productos o servicios que no funcionan, excediendo el presupuesto y el calendario de entregas. Las organizaciones en este nivel abandonan los proyectos ante una crisis y no tienen la capacidad de replicar los éxitos y buenas prácticas de procesos previos.
2. **Nivel de madurez administrativo: e**n este nivel los objetivos generales y específicos han sido alcanzados, los proyectos han asegurado que los requisitos sean gestionados y los procesos son planificados, ejecutados de manera controlada, conforme a los planes documentados y los productos o servicios satisfacen las necesidades del cliente en los requisitos especificados, también los objetivos y normas.
3. **Nivel de madurez definida:** en este nivel la organización ha logrado alcanzar todos los objetivos específicos de las áreas asignadas en los niveles 2 y 3. Los procesos están bien caracterizados, entendidos y donde se definen normas, procedimientos, herramientas y métodos para su aplicación.
4. **Nivel de madurez administrativo cuantitativamente:** en este nivel se seleccionan los procesos que aportan significativamente al rendimiento del proceso en general, concluyendo subprocesos que se controlan por medio de métricas, reportes y estadísticas que permiten medir la capacidad, permitiendo identificar casos que generen problemas para ser corregidos y no se repitan en el futuro, también para tomar decisiones de negocio que permitan el crecimiento de la organización.
5. **Nivel de madurez optimización:** este nivel está centrado en la mejora continua, la optimización y el rendimiento de los procesos de la organización, es el nivel ideal.

El modelo está articulado con los siguientes componentes: objetivos, prácticas y subprácticas, cuyas definiciones se encuentran en las siguientes tarjetas:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

CMMI también define los niveles capacidad de la organización, en el cual se debe tener en cuenta las prácticas específicas y genéricas que pueden mejorar los procesos de la organización, cada nivel es una capa que permite estar en mejora continua con respecto a un proceso, clasificados así:

* **Nivel de capacidad 0: incompleta:** indica que un proceso se realiza parcialmente o no se realiza, dado que uno o más objetivos no se cumplen y no presenta objetivos genéricos.
* **Nivel de capacidad 1: realizar:** en este nivel se espera que se realicen todas las actividades específicas y genéricas del proceso, dado el nivel es posible que el rendimiento no sea estable y se llegue a no cumplir con objetivos como calidad, costo, tiempos, pero se puede ejecutar. Este solo es el inicio, lo cual significa que se hace algo, pero no se puede demostrar plenamente cómo funciona.
* **Nivel de capacidad 2: administración:** se denominan procesos administrados, ya que se puede contar con una planeación, una trazabilidad y un control en los proyectos individuales o grupales.
* **Nivel de capacidad 3: definir:** son procesos que están diseñados a partir de un conjunto de procesos estándar en la organización, con productos de trabajo, modelos, medidas y otras características relacionadas a herramientas que faciliten la identificación y ejecución del proceso.
* **Nivel de capacidad 4: administrada cuantitativamente:** en este nivel se realiza un seguimiento al proceso por medio de métricas estadísticas y otras técnicas cuantitativas, para poder establecer el rendimiento de los procesos, con esto se realiza una gestión en el proceso. En este nivel los atributos de calidad y rendimiento se entienden como términos estadísticos usados para la administración del producto a lo largo de la vida del proceso.
* **Nivel de capacidad 5: optimización:** este nivel se centra en la mejora continua del rendimiento de los procesos por medio de mejoras continuas que buscan la innovación.

**1.3. Criterios del manejo de la confidencialidad**

Cuando se habla de confidencialidad se refiere a los principios de seguridad de la información y en ese caso es importante tener en cuenta las siglas CIA que hacen referencia a: Confidencialidad, Integridad y Disponibilidad; sin estos principios fundamentales no podemos hacer referencia a que la información está segura, si uno de estos principios falla, existe la posibilidad de materializarse algún riesgo de seguridad de la información. La ISO/IEC 27001, es un estándar que especifica los requisitos necesarios para establecer, desarrollar, mantener y mejorar el sistema de gestión de seguridad la información (SGSI) y en esta norma se definen los controles necesarios para garantizar estos principios.

| Se debe tener en cuenta que ningún sistema es totalmente seguro, ni siquiera aquellos que tienen un nivel de madurez avanzado, por el contrario, se deben hacer evaluaciones periódicas frecuentes de los posibles riesgos y tomar las acciones pertinentes para minimizar dichas vulnerabilidades. Es necesario tener en cuenta cuales pueden llegar a ser las posibles amenazas de seguridad y cuando ya se tengan claras se pueden definir las acciones necesarias para minimizarlas. |  |
| --- | --- |

A continuación, una explicación de los términos CIA:

* **Confidencialidad**

Se conoce como el método que permite la prevención de la divulgación de la información a personas o sistemas que no tengan la autorización necesaria para acceder a ella.

* **Integridad**

Hace referencia a que la información se debe mantener intacta, libre de alteraciones y cambios por terceros, por eso se debe asegurar que la información sea modificada por la misma persona a la que le pertenece evitando de esta manera que se pierda la integridad.

* **Disponibilidad**

Es de suma importancia dado que de nada sirve que la información esté segura e íntegra si no podemos tener acceso a ella. La información debe estar disponible en cualquier momento que el usuario requiera accederla.

La seguridad de la información contempla la protección de la infraestructura, los dispositivos, las herramientas, la integridad física y moral de los usuarios, por lo cual se han optado por mecanismos como la encriptación, creación de *firewalls*, simuladores de ataques.

En la actualidad hay empresas que se encargan de gestionar la seguridad de otras empresas, las cuales pueden proveer servicios en las siguientes áreas:

* **Los servicios gestionados de seguridad**, desde los que se administran y monitorean los sistemas de seguridad las 24 horas del día.
* **Los servicios profesionales**, que apoyan la consultoría de análisis de vulnerabilidades, las pruebas de penetración, la implantación de normativas y estándares.
* El **área de tecnología e ingeniería**, para diseñar, integrar y soportar soluciones tecnológicas de seguridad, pues los clientes no siempre tienen la infraestructura necesaria que ellos le proporcionarán.

**1.4. Herramientas tecnológicas para pruebas**

Cuando se habla de herramientas para la aplicación de pruebas de *software*, por lo general se piensa en automatización, lo cual es un gran elemento potenciador de productividad, pero no siempre se aplica en todos los contextos, por lo que es importante reconocer la viabilidad de aplicar pruebas automatizadas dependiendo del campo y contexto a aplicar.

Se debe iniciar con la identificación de contextos del proyecto, además es necesario para un *tester* reconocer qué es automatizar una prueba de *software* y tener total conciencia de cuáles escenarios no son automatizables. Dentro de este proceso es importante que se identifique cómo se puede optimizar el proceso de automatización, trabajo en equipo con los compañeros de proyecto y activos para probar las herramientas de automatización.

A continuación, se presentan aspectos a considerar al momento de automatizar las pruebas de *software*:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

La automatización tiene sus ventajas y desventajas y todo depende del proyecto, el tiempo, el costo, la calidad y la metodología, por lo cual otro de los criterios para definir si automatizar o no, es el contexto.

Existen algunos mitos que puedes encontrar respecto a la automatización de pruebas que se presentan a continuación:

Interfaz de usuario gráfica, Aplicación, PowerPoint

Descripción generada automáticamente

Ahora bien, se expresan los siete principios que ayudan a comprender el objetivo del *testing*, independientemente de que sean pruebas manuales o automatizadas, según Rodríguez (2020):

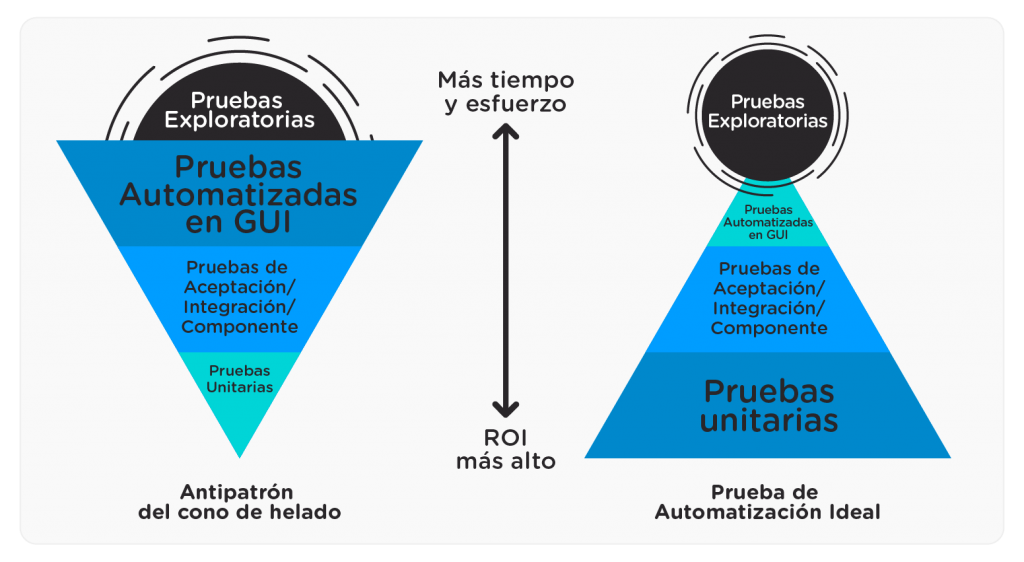
1. El valor de cualquier práctica depende de su contexto.
2. Hay buenas prácticas en contexto, pero no hay “buenas prácticas”.
3. Las personas, trabajando juntas, son la parte más importante del contexto de cualquier proyecto.
4. Los proyectos no son estáticos y a menudo toman caminos impredecibles.
5. El producto es una solución. Si el problema no se resuelve, el producto no funcionará.
6. El *testing* de *software* es un proceso intelectual desafiante.
7. Solo a través del juicio y la habilidad, practicados cooperativamente durante todo el proyecto, podremos hacer las cosas correctas en el momento adecuado para probar nuestros productos de manera efectiva.

| Al iniciar el proceso de automatización se debe tener en cuenta el costo de mantener los *scripts*, por esto es ideal garantizar que desde las pruebas unitarias se minimice el riesgo de errores en el código, siendo requerido contar una retroalimentación inmediata y continua a las diferentes capas, por esto las pruebas manuales y exploratorias son más valiosas a nivel de integración e interfaz de usuario, centrando esfuerzos en las pruebas que no son posibles de automatizar. |  |
| --- | --- |

En la siguiente figura, se puede observar dos modelos de aplicación de pruebas de *software*, a la izquierda un modelo que no está optimizado y donde los esfuerzos se reflejan en la capa superior, sin retroalimentación temprana; mientras que el de la derecha desde el inicio del desarrollo cada elemento como una unidad es verificado minimizando y optimizando los procesos de prueba de cara a un ambiente productivo:

**Figura 1**

*Modelo tipo cono y pirámide en ejecución de pruebas*



Nota. Tomada de Rodríguez (2020).

Aunque las pruebas manuales y las automatizadas son diferentes no son excluyentes entre ellas, al contrario, se consideran acciones complementarias que buscan mejorar la calidad del producto *software*.

Como se ha indicado previamente desde el contexto, no todas las pruebas son automatizables, por lo que se detallan los casos en los cuales se pueden automatizar:

Interfaz de usuario gráfica, Aplicación, PowerPoint

Descripción generada automáticamente

Cuando ya se sabe qué se va a automatizar, se debe seleccionar la herramienta pertinente para implementar el proceso, aunque la decisión de qué herramientas conformarán el repertorio de recursos disponibles dependerá de la experticia del equipo de desarrollo y de pruebas, así como del tipo de proyecto y los recursos asignados.

A continuación, se presentan un listado de herramientas que podrán utilizarse para automatizar diferentes tipos de pruebas:

| **Selenium:** herramienta de código abierto que se usa para realizar pruebas de interfaz en aplicaciones web. | Selenium WebDriver en un Ambiente de Pruebas Continuas | SG Buzz |
| --- | --- |
| **Appium:** herramienta de código abierto basada en Selenium que se puede automatizar pruebas para aplicaciones Android y iOS. | Cómo configurar Appium para la automatización de pruebas |
| **Cucumber:** parte del enfoque BDD (*Behavior Driven Development*). | Gherkin y el testing en desarrollo |
| **Ghosh inspector:** herramienta ideal para principiantes, dado que no se requiere saber programar para la definición de las pruebas. | Chequeos automáticos con Ghost Inspector - Federico Toledo |
| **Test Studio:** herramienta que sirve para probar funcionalidades de API y carga. |  |
| **Testim:** herramienta inteligente la cual emplea métodos de aprendizaje automático, pudiendo ejecutar las pruebas en diferentes entornos. | Software Testim - 2021: opiniones, precios y demos |
| **LoadRunner:** herramienta usada principalmente para probar rendimiento en condiciones de alta carga, simulando la concurrencia de miles de usuarios. |  |
| **WebLoad:** herramienta con capacidades de scripting puede probar casos de prueba complejos aplicando en aplicaciones web independientes y que están en la nube. | WebLOAD – TestMatick |
| **Blazemeter:** herramienta que se usa para medir el rendimiento de aplicaciones móviles, API o aplicaciones web, es utilizada en cualquier etapa del desarrollo con lo que apoya la entrega de *software* de alto rendimiento. Permite agregar una nueva configuración de prueba obteniendo nuevos datos de Google Analytics. | Cinco Herramientas para Pruebas de Rendimiento - HowToTesting |
| **jMeter:** herramienta de código abierto tipo apache la cual es aplicada para verificar el rendimiento de sitios web dinámicos, personas con pocos conocimientos puede interpretar fácilmente los resultados generados por la herramienta. | Label requests by reponse time in JMeter - DEV Community |

También existen herramientas de gestión de pruebas, a continuación, se presentan algunas de estas:

| **Xray:** herramienta de gestión de pruebas preferidas para pruebas manuales y automatizadas, la cual realiza la clasificación de las mismas y proporciona un resultado de evaluación eficiente en un corto tiempo. |  |
| --- | --- |
| **TestRail:** herramienta web que permite gestionar los casos de prueba, se puede personalizar y escalar funcionalidades. Es posible ver en tiempo real el avance de las pruebas y sirve como repositorio de la ejecución de pruebas manuales y automatizadas. | Pruebas en Aplicaciones Móviles - Abstracta Chile |
| **Testpad:** emplea planes de prueba para pruebas ágiles, exploratorias, gestión de casos de prueba, BDD, entre otros. |  |

Otro tipo de herramientas importantes son las de trazabilidad de incidencias, aquí se presentan algunas:

| **Jira:** popularmente conocida como una herramienta de gestión de proyectos además de apoyar el proceso de seguimiento y registro de bugs, está integrado con entornos de desarrollo para tener funcionalidades avanzadas en los ciclos de construcción, proporcionando reportes, estadísticas y alertas. | Un dibujo de un perro  Descripción generada automáticamente con confianza media |
| --- | --- |
| **Mantis:** herramienta de seguimiento de código abierto, contiene un panel de servicio donde se pueden ver los elementos pendientes, asignaciones a usuarios y filtros por equipo, proyecto o usuarios. | Un dibujo animado con letras  Descripción generada automáticamente con confianza media |
| **Natsparker:** herramienta que sirve para escanear automáticamente sitios web, aplicaciones o servicios web. Esta herramienta busca encontrar vulnerabilidades de seguridad, proporcionando pruebas de explotación sin falsos positivos. |  |
| **SoapUI:** herramienta funcional para servicios soap, rest, jms y amf, se usa en pruebas funcionales como en cargas de Apis, también puede realizar llamadas HTTP y JDBC. | SOAPUI – Interpolados |

Para finalizar, se listan algunas herramientas de prueba entre diferentes navegadores:

| **Lambda Test:** herramienta de prueba escalable basada en la nube que permite hacer pruebas entre diferentes navegadores. Permite la ejecución de casos de prueba en paralelo y realiza pruebas en tiempo real de forma eficiente. | Lambda-Test-logo - AwsQuality Technologies | Salesforce ISVPartner |  AppExchange Partner |
| --- | --- |
| **SauceLabs:** admite más de 650 navegadores, sistemas operativos y conjugaciones entre dispositivos, lo que brinda una alta gama de infraestructura necesaria para pruebas manuales o automatizadas. | Sauce Labs - Open Collective |
| **Espresso:** herramienta de pruebas de *software* de primera categoría para pruebas en Android desarrollada por Google. | Call view methods when testing by Espresso and Kotlin in Android | by  Alireza A. Ahmadi | AndroidPub | Medium |

**1.5. Modelo de procesos de pruebas**

El modelo de procesos de pruebas en V o modelo a cuatro niveles es empleado en muchos procesos de desarrollo, en los años 90 surgió su primera versión, adicional al modelo en V también lo acompañan los procesos de calidad y las fases individuales como tal, este modelo define las siguientes fases de proyecto individuales:

* Al inicio del proyecto se determina un análisis de las especificaciones del sistema.
* Luego se define la fase funcional con los requisitos funcionales y no funcionales.
* Posteriormente se continúa con la fase de diseño.
* Y, por último, se define la arquitectura del sistema y la codificación del mismo.

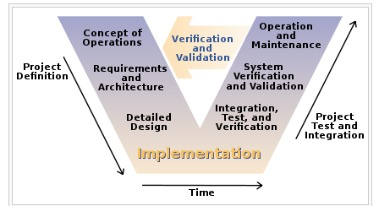
| De acuerdo con estos planes, se inicia el desarrollo y con ello se tendrá lugar a las fases de control de calidad, denominadas también verificación o validación, que por lo general están relacionadas a las fases del desarrollo. |  |
| --- | --- |

Es denominado modelo en V dado que compara cada fase de desarrollo con las fases de control de calidad, el brazo izquierdo contiene las tareas de diseño y desarrollo del sistema, el derecho las medidas de control de calidad definidas para cada fase, en medio de los dos se puede encontrar la implementación de producto, es decir el desarrollo y código fuente del producto *software*.

En la siguiente figura se detalla el modelo único de evaluación el cual se orienta a los productos de trabajo en lugar a los procesos que lo producen, además se pueden observar las fases mencionadas:

**Figura 2**

*Modelo V*



Nota. Tomada de Grhsoftware (2020).

**1.6. Documentación de pruebas**

En la fase de pruebas se lleva a cabo la ejecución de los diferentes planes de prueba definidos en la planeación de las fases anteriores, donde es necesario conseguir el objetivo y documentar la ejecución de cada prueba, en la medida de lo posible minimizando o evitando en su totalidad acarrear con errores que debieron solucionarse en iteraciones anteriores o en fases del desarrollo temprano.

En esta fase se deben ejecutar las siguientes acciones y generar estos entregables:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**1.7. *Continuous Integration* y *Continuous Delivery***

La integración continua o CI es el proceso de automatizar la integración de los cambios en las versiones de código y ajustes en el producto *software* de diferentes fuentes en un único proyecto. Esta es una de las prácticas fundamentales en DevOps, lo que permite integrar o fusionar con frecuencia los cambios de desarrollo en un repositorio central donde luego se realizan las compilaciones y las pruebas.

| Una herramienta de control de versiones es un elemento clave en el proceso de CI, dado que apoya el proceso de verificación con pruebas automatizadas, los estilos definidos por las reglas de desarrollo, verificación de código estático, sintaxis, entre otros. | Desarrolladores web dibujados a mano vector gratuito |
| --- | --- |

Esta forma de trabajo soluciona problemas de coordinación manual entre los desarrolladores, el equipo de operación y el resto del equipo encargado del correcto funcionamiento del producto *software*.

Este tipo de trabajo manual representa una tarea de sincronización compleja y a menudo confusa, que añade procesos y costos innecesarios a la ejecución del proyecto. Al no contar con procesos CI, las liberaciones son más inestables y lentas, dado que los riesgos crecen considerablemente a medida que aumentan las nuevas funcionalidades o el equipo de desarrollo que intervienen en él, también se puede concluir que sin la integración continua y al limitarse la comunicación el proceso de desarrollo se vuelve engorroso y se cae en el error de trabajar en una caja negra, donde solo llegan solicitudes de nuevas funcionalidades pero no se sabe en realidad con que se cuenta o con que no.

*Continuous Delivery* (entrega continua en español) es un concepto relativamente nuevo que cada vez es adoptado con más frecuencia en los procesos de desarrollo. Estas prácticas incluyen el desarrollo, control de calidad y entrega, dado que estas no son definitivas y antes al contrario se repiten de formas automatizadas, una y otra vez durante el proceso de desarrollo. El beneficio de este es que el producto se puede someter regularmente a procesos de calidad en cada una de sus fases de desarrollo, permitiendo realizar entregas, sin importar que el equipo de desarrollo siga trabajando en funcionalidades del producto final. Otro punto importante es que hay una realimentación o *feedback* constante que proviene del pipeline, lo que permite tomar acción inmediata ante una mejora.

**1.8. Depuración y técnicas de resolución de problemas**

La capacidad de encontrar y solucionar problemas rápidamente en un producto *software* pertenecen a las habilidades más importantes en los equipos de desarrollo o encontrar un problema en un ambiente productivo suele ser realmente desafiante, hay equipos completos dedicados al mantenimiento del sistema los cuales se encargan de corregir errores en los productos *software*. También hay personal que están disponibles de guardia de tal manera que si se llega a caer el sistema tienen la responsabilidad de que rápidamente este vuelva a estar en línea lo antes posible.

La resolución de problemas en el ámbito de depuración de código tiene como objetivo delinear una serie de recomendaciones o consejos que ayudarán a solucionar los problemas lo antes posible.

El contar con la capacidad necesaria para resolver los problemas en ambientes productivos lo antes posible es sumamente crítico, ya que la caída de un sistema puede generar pérdidas de miles o millones de dólares, por lo cual es ideal seguir las siguientes técnicas:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez se tengan en cuenta estas técnicas el crear nuevas funciones o al realizar mantenimiento, dar soporte o realizar la trazabilidad de un *bug*, lo más probable es que los tiempos se optimicen y la respuesta respecto a las acciones a realizar sean más rápidas y certeras.

**2. Construcción, desarrollo y ejecución de pruebas de *software***

La construcción de *software* está basada en fases que permiten incrementar el producto desde sus cimientos hasta obtener un producto base; esto es posible, ya que en el proceso se implementan diferentes estrategias y metodologías, el gran desafío está, en cómo construir un producto que se pueda validar rápidamente y que contenga los niveles adecuados de calidad. La naturaleza de los requisitos de *software* es cambiante, lo cual complica la precisión de los equipos, dilatando posiblemente el tiempo, los recursos y el presupuesto asignado.

**2.1. Aseguramiento de calidad de *software***

El aseguramiento de la calidad de *software* o SQA, es un área importante en el proceso que busca estar en constante validación de la calidad del producto *software*, por lo general en el establecimiento de un plan de aseguramiento de calidad.

Por esto, es importante tener en cuenta los siguientes conceptos y su aplicación práctica en el ciclo de desarrollo de *software*:

| Se define el concepto de “calidad” como un conglomerado de características de una entidad que son esenciales y que además permiten calificar su valor. Otro concepto a nivel de empresa está relacionado con el conjunto de estándares y normas que debe tener un producto o servicio para ser reconocido y percibido en el mercado satisfaciendo las necesidades de los clientes. |  |
| --- | --- |

Cuando se hace referencia a la “**Calidad de *Software* (QA)**” se entiende como el nivel que tienen los clientes para determinar según su percepción que el producto *software* satisface sus necesidades y expectativas. A nivel empresarial, la calidad de *software* hace referencia a las actividades que se establecen de manera común para asegurar que el desarrollo de *software* en todos los proyectos es de calidad. También se puede definir como la relación coherente del producto *software* con los requerimientos funcionales y no funcionales.

El “**Aseguramiento de la Calidad de *Software* (SQA)**” se conoce como el conjunto de actividades, métodos y técnicas planificadas y sistemáticas necesarias para asegurar que el producto *software* satisfará los requerimientos dados de calidad. Se encarga de garantizar que el producto haya sido construido siguiendo los procesos establecidos por la organización.

En el siguiente video se presente la breve historia del surgimiento del Aseguramiento de la Calidad de Software (SQA):

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Cabe aclarar que la calidad de *software* puede medirse una vez en la fase de implementación o terminado completamente o cuando el producto está en producción, no obstante, este hecho implicaría amplios costos para la compañía de *software* a diferencia de detectar y resolver los problemas en las fases iniciales como el diseño o en los requerimientos que generaría una reducción de los costos en cuanto a los cambios a realizar. Por lo tanto, es necesario tener en cuenta el proceso de SQA durante las etapas iniciales de la construcción del producto, puesto que es donde se definen las características que el producto debe garantizar para satisfacer las necesidades implícitas y explícitas. Por lo tanto, se debe tener en cuenta que si el diseño y los requerimientos no están correctamente definidos, es decir que están ambiguos, incompletos o son incoherentes, en este momento va a ser muy costosa su corrección.

En la siguiente infografía interactiva se presentan las funciones del Aseguramiento de la Calidad de Software (SQA):

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Además, para realizar el Aseguramiento de la Calidad de Software (SQA) se deben tener en cuenta diversos principios básicos como son:

* La calidad debe ser una constante a lo largo de todo el ciclo de vida del *software*. Por lo tanto, la calidad es una acción que se construye en la medida que el producto se va desarrollando, mas no cuando el producto está terminado, es por esta razón que todo el equipo o involucrados en el proceso deben también enfocarse en la calidad del producto.
* La calidad de *software* solo se logra con la contribución de todas las personas involucradas, es decir que la calidad no es un proceso solo del equipo de aseguramiento de la calidad de *software*, sino más bien una actividad sincrónica entre todas las personas que colaboran para el desarrollo del producto.
* La gestión y la planificación de la calidad debe ser eficaz y previamente definido y ejecutado; por lo tanto, se deben establecer procesos, estándares y una planeación claramente definida para poder asegurar la calidad del producto, donde contengan puntos de verificación y un proceso claro definido que incluya qué, quién y cuándo va a realizar la verificación o medición de la calidad.
* Se deben dirigir los esfuerzos a prevención de defectos, es decir que una vez que un defecto es detectado además de corregirlo se deben tomar medidas necesarias para que no vuelva a ocurrir. Es en este punto donde se tiene en cuenta lo que es el análisis de las causas raíces, una vez que se encuentra un defecto se debe analizar por qué se ocasionó y cuál fue la causa que lo originó; por ejemplo, si es un problema en el proceso o si es un problema con las herramientas que se están utilizando o si fue una mala definición de los requerimientos, lo que constata que un error no solamente se soluciona o resuelve, sino que hay que indagar más en la causa raíz; porque de no hacerlo en el futuro se va a presentar el mismo problema.
* Hay que reforzar los sistemas de detección y eliminación de defectos durante las etapas iniciales, lo que reafirma que eliminando los defectos en las etapas de requerimientos y diseño, estamos asegurando que esos defectos no van a ser replicados en las etapas subsiguientes, por ende es menos costoso detectar y corregir los defectos en las etapas iniciales, por ejemplo si se detecta un defecto en los requerimientos o en el diseño del *software*, lo más posible es que se tenga que corregir documentos o diagramas, pero si se espera a la etapa de desarrollo o cuando el *software* ya está en producción, tenemos que volver hacia atrás en todas las fases del ciclo de vida del desarrollo del *software*.
* La calidad, al igual que la productividad, costos y los plazos de entrega están al mismo nivel, lo que quiere decir que el cumplimiento de los plazos de entrega y mantenerse dentro del presupuesto también son variables que se evalúan dentro del proceso de aseguramiento de calidad de *software*, pues de que sirve un producto sin defectos cuando ha costado mucho más de lo presupuestado y se ha entregado cuando ya no será útil.
* Es esencial la participación de la dirección, que ha de propiciar la calidad, esto quiere decir que la dirección debe ser el primer interesado en asegurar la calidad y de proveer las directrices, herramientas y el apoyo necesarios para que se pueda lograr, por lo tanto si no hay apoyo de la dirección para implementar y ejecutar el proceso de aseguramiento de calidad del *software*, será difícil que se pueda llevar a cabo la calidad y por ende los productos no van a salir con la calidad requerida.

Es importante reconocer los factores que afectan o determinan la calidad del *software*, los cuales se definen a continuación:

* + **Corrección:** nivel en que un *software* satisface las especificaciones y logra los objetivos propuestos por el cliente, lo anterior responde a la pregunta: ¿el *software* hace lo que yo quiero?
  + **Fiabilidad:** nivel en que se puede esperar que un programa realice sus funciones esperadas con la precisión requerida, lo anterior responde a la pregunta: ¿el sistema está apto para usarse?
  + **Eficiencia:** cantidad de recursos de máquina y de código requeridos por un programa para efectuar sus funciones, este factor responde a la pregunta: ¿se ejecutará en mi *hardware* con eficiencia?
  + **Integridad:** nivel en el que puede controlarse el acceso al *software* o a los datos por personal que no es autorizado, este factor responde a la pregunta: ¿el *software* es seguro?
  + **Seguridad:** disponibilidad de instrumentos que permiten controlar y proteger los programas y los datos.
  + **Reusabilidad:** nivel de trabajo solicitado para que el *software* o alguna de sus partes pueda reutilizar en otras aplicaciones. Este factor responde a la pregunta: ¿podré reutilizar alguna parte del programa para construir otro proyecto?
  + **Exactitud:** nivel de precisión de los cálculos y del control que tiene el programa al ejecutar sus funciones.
  + **Completitud:** nivel en que se ha obtenido una total implementación de las funciones solicitadas.
  + **Flexibilidad:** trabajo necesario para modificar un programa que ya está en funcionamiento u operación. Este factor responde a la pregunta: ¿puede ser modificado de manera fácil?
  + **Portabilidad:** trabajo necesario para migrar el programa desde un hardware o un entorno de sistemas a otro ambiente u otro hardware, este factor responde a la pregunta: ¿podré usarlo en otra máquina o con otra configuración distinta a la máquina donde se desarrolló?
  + **Consistencia:** consiste en utilizar un diseño semejante y técnicas de documentación en el transcurso del proyecto de desarrollo de *software*.
  + **Facilidad de uso:** trabajo necesario para aprender a utilizar un programa y que además sabe preparar su entrada e interpretar su salida. ¿Está diseñado para ser utilizado de manera fácil?
  + **Facilidad de mantenimiento:** trabajo necesario para ubicar y arreglar un error en un programa, este factor responde a la pregunta: ¿permite ser corregido de manera fácil?
  + **Facilidad de prueba:** nivel de trabajo requerido para probar un programa de forma que asegure la ejecución de las funciones solicitadas, ¿permite ser probado de manera fácil?
  + **Facilidad de interoperabilidad:** nivel de trabajo necesario para acoplar un sistema a otro, lo anterior responde a la pregunta: ¿podrá interactuar con otros sistemas?
  + **Facilidad de auditoría:** se concentra en la facilidad en que se puede comprobar la conformidad con los estándares.
  + **Normalización de las comunicaciones:** nivel en que se utiliza el ancho de banda, las interfaces estándar y los protocolos.
  + **Tolerancia a los errores:** percance que se ocasiona cuando el programa encuentra un error.

Cuando se califica la calidad de un producto se debe elegir los factores que van a ser estimados como requerimientos; por lo tanto, para realizar esta elección se debe tener en cuenta las propiedades particulares de la aplicación a desarrollar o de su ambiente, como ejemplo podemos observar que si la aplicación se desarrolla para un ambiente en el que el *hardware* evoluciona rápidamente; entonces, se debe tener en cuenta el factor de portabilidad, por otro lado si se espera que las especificaciones del sistema cambien frecuentemente entonces el factor flexibilidad va a ser importante.

Otro punto que se debe tener en cuenta es el costo de los factores de calidad frente al beneficio que van a proporcionar (análisis de costo/beneficio), es decir que si agregamos todos los factores implicaría un incremento amplio de los costos, en lugar de lo anterior es prudente seleccionar los factores que logren contribuir el mayor valor posible al producto.

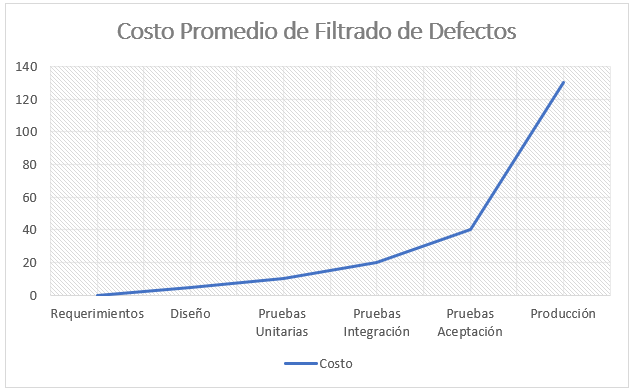
Los costos de calidad son importantes para conseguir productos de calidad, por lo que se establecen actividades para prevenir, mejorar o detectar la baja calidad que puede haber en los productos *software*.

| En la aplicación de la calidad de *software*, generalmente los ingenieros o empresas comienzan a realizar pruebas y a detectar defectos en el momento en que la aplicación ha sido totalmente construida lo cual no es una acción correcta, en la siguiente grafica se presenta el costo promedio de filtrado de defectos, se podrá percibir que mientras más avanzamos en la construcción del *software* más alto es el costo de las reparaciones y modificaciones. |  |
| --- | --- |

Si se detectan los errores en las primeras fases como de requerimientos o diseño los costos son menores debido a que lo más posible es que se deba modificar básicamente documentos; en cambio sí se detecta un defecto en las pruebas de aceptación o en producción, lo más seguro es que se deban involucrar prácticamente a todo el equipo porque posiblemente se deba regresar hasta la primera fase que es de requerimientos; por ende, el costo de corregir defectos se incrementa debido a que todo el trabajo que va a tener que realizar el equipo nuevamente seria cargado al costo del proyecto. Debido a lo anterior es muy importante que el Aseguramiento de la Calidad de Software (SQA) comience en las primeras etapas del desarrollo de *software*.

**Figura 3**

*Costo promedio de filtrado de defectos*



Para garantizar el Aseguramiento de la Calidad de Software (SQA), se deben incluir los siguientes componentes:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

La implementación del Aseguramiento de la Calidad de Software (SQA) trae consigo beneficios y problemas, los cuales se presentan a continuación:

**Beneficios**

* El *software* reducirá los defectos que son poco perceptibles o latentes, lo que implica la reducción del tiempo y el trabajo en el transcurso de las fases de prueba y mantenimiento.
* Se proporcionará una mayor fiabilidad, lo que proveerá al cliente una mayor satisfacción, objetivo común que todas las empresas buscan.
* Se lograrán disminuir los costos de mantenimiento (un porcentaje importante de los costos totales del *software*).
* El tiempo y el costo total del ciclo de vida del *software* disminuirá, si se realiza una eficiente implementación y ejecución del proceso de aseguramiento de calidad del *software*.

**Problemas**

* Es difícil institucionalizar en organizaciones pequeñas, en las que no están disponibles los recursos necesarios para realizar las actividades del Aseguramiento de la Calidad de *Software* (SQA).
* Representa un cambio cultural, y el cambio nunca es fácil de aceptar e implementar.
* Requiere un gasto que, de otro modo, nunca se hubiera destinado explícitamente a la ingeniería del *software* o al aseguramiento de la calidad.

Para que el proceso del Aseguramiento de la Calidad de *Software* (SQA) se aplique correctamente, se deben aplicar estas tres etapas:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En el Aseguramiento de la Calidad de *Software* usualmente se pueden trabajar con cuatro roles; no obstante, un analista puede ejercer el rol de diseñador y *tester*:

* **Gerente de pruebas:** tiene la responsabilidad de lograr el éxito del esfuerzo de las pruebas, es el responsable de planificar, gestionar los recursos y solucionar los problemas que se presentan dentro del equipo de trabajo.
* **Analista de pruebas:** inicialmente este rol identifica y define las pruebas necesarias, por lo tanto, va a indicar el alcance de lo que es necesario probar.
* **Diseñador de pruebas:** es el responsable de definir hacia dónde se enfocan las pruebas, así como la identificación de técnicas y recursos requeridos.
* ***Tester*:** es el responsable de realizar las pruebas requeridas, así como de registrar los resultados de las pruebas.

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

El Aseguramiento de la Calidad de Software se compone de artefactos específicos que se forman en las diversas etapas del proceso de SQA, como se muestra en el siguiente recurso de aprendizaje:

Interfaz de usuario gráfica, Aplicación, PowerPoint

Descripción generada automáticamente

**2.2. Implementación del plan de pruebas**

El plan de pruebas es un documento que describe un conjunto de procedimientos, técnicas y normas para probar un sistema.

En la siguiente tabla se describen los elementos que contiene un plan de pruebas:

**Tabla 1**

*Elementos de un plan de pruebas*

| Nombre y versión | Debe contener el nombre del sistema a probar, el nombre del documento y la versión. |
| --- | --- |
| Introducción | Especificar en qué consiste el sistema y cual sistema se va a probar. También se incluye y describe el alcance de las pruebas. |
| Pruebas planificadas | En esta parte se listan los tipos de prueba a realizar y los requerimientos a probar en el proceso. Especifica y describe las características a probar y las que no se van a probar. |
| Estrategias de prueba | Representa el enfoque recomendado para la comprobación de las aplicaciones. |
| Criterios de entrada y salida | En esta parte se especifica los criterios de entrada y salida de la aplicación al área de Aseguramiento de Calidad de *Software* (SQA). |
| Tareas de prueba | Se describen las actividades (plan, diseño, implementación y ejecución de pruebas, así como la evaluación de los resultados) que realiza el equipo de SQA. |
| Entregables | Se describe el informe de resultados de prueba y los defectos detectados durante el esfuerzo de pruebas. |
| Necesidades del ambiente | Describe los recursos del sistema. |
| Responsabilidades, personal y necesidades de entrenamiento | Describe las características del personal requerido para el esfuerzo de pruebas, así como las necesidades de entrenamiento. |
| Hitos principales | En este punto se plantean las fechas estimadas de inicio y fin de las principales actividades de prueba |
| Riesgos y contingencias | En este punto se plantean los riesgos identificados al momento de la planificación y su estrategia de mitigación. |

| **LLAMADO A LA ACCIÓN:**  Para un mejor entendimiento de cómo construir e implementar el plan de pruebas, ver el anexo que contiene un ejemplo práctico y cómo se relacionan los elementos requeridos. |
| --- |

La estructuración de un caso de prueba se convierte en una actividad sin dificultad, si tenemos la información necesaria para su proceso de elaboración, así mismo al momento de verificar un *software* es de mucho aporte, puesto que se transforma en una herramienta esencial en el proceso de registro, seguimiento y control.

A continuación, se nombran los elementos principales que debe contener un caso de prueba:

Tabla 2

*Elementos de caso de prueba*

|  | |
| --- | --- |
| Identificador | Puede ser alfanumérico o numérico. |
| Nombre | Nombre del caso de prueba de manera concisa. |
| descripción | Objetivo del caso de prueba, también describe que probara, en ciertas ocasiones se incluye el ambiente de pruebas. |
| Numero de orden  Ejecución | Orden en la cual se ejecuta el caso de prueba, en la situación de que se tengan múltiples casos de prueba. |
| Requerimiento asociado | Si se plantea un caso de prueba se debe saber a qué requerimiento va asociado para mantener la trazabilidad. |
| Precondición | Estado en la cual se debe encontrar el sistema antes de comenzar la prueba. |
| Postcondición | El estado en que debe encontrarse el sistema luego de ejecutar la prueba. |
| Resultado esperado | Objetivo que debe ser alcanzado posterior a la ejecución de la prueba. |

| **LLAMADO A LA ACCIÓN**  Para aprender cómo redactar y construir un caso de prueba vea el siguiente anexo: |
| --- |

Redactar el *script* de pruebas es un artefacto responsabilidad del *tester*, siendo este quien lo escribe y lo ejecuta. A continuación, se nombran los elementos principales que debe contener un *script* de prueba:

**Tabla 3**

*Elementos para el Script de prueba*

|  | |
| --- | --- |
| Nombre | Identificador único para el script. |
| descripción | Describe el objetivo del script de pruebas. |
| Pasos | Acciones que se van ejecutando en la aplicación. |
| Puntos de verificación | Se comprueban aquellos puntos donde se verifica si la aplicación se está comportando de la forma esperada.  Son las observaciones puntuales que se realizan durante la ejecución de las pruebas para verificar que el producto cumple con los requerimientos específicos. |
| Resultado esperado | Por cada punto de verificación puede haber un resultado esperado asociado. |

| **LLAMADO A LA ACCION**  Para aprender cómo redactar y construir un script de prueba ver el siguiente anexo: |
| --- |

**2.3. Configuración del ambiente de pruebas**

El ambiente de pruebas es una configuración de *software* y *hardware* en el que el equipo de pruebas va a realizar la prueba del producto de *software* que ha sido desarrollado. La gestión de los ambientes de pruebas es de gran importancia, puesto que asegura que el *software* sea llevado a producción con los niveles de calidad óptimos.

**Características de los ambientes de pruebas**

* El equipo de pruebas debe compartir un servidor en el cual va a residir el ambiente de pruebas, por lo general en una máquina distinta a la que utiliza el desarrollador.
* Es recomendable que no se usen direcciones IP, sino más bien nombres de dominio distintos a los de desarrollo y producción, por lo que podría crear desorden o confusión en el equipo de pruebas.
* Preferiblemente el ambiente de pruebas debe ser lo más parecido a uno de producción, que incluya herramientas de control de versiones, aplicaciones cliente/servidor, bases de datos, cuentas de usuario del sistema operativo, *clusters*, configuración de servidores, el mismo sistema gestor de bases de datos instalado tanto en desarrollo como en producción e instalar en un computador o servidor diferente al del ambiente de pruebas las herramientas de gestión de incidencias y casos de prueba.

**Restricciones en los ambientes de pruebas**

* Los desarrolladores deben tener desactivados los privilegios relacionados con el acceso de modificación en los ambientes de pruebas.
* El desarrollador no debe gestionar directamente el ambiente de pruebas, si no el administrador.
* Requiere de control de versiones.
* Instalar una versión en producción, una vez que ha sido probada en el ambiente de pruebas.

.

**Procedimientos para la gestión de ambientes de pruebas**

* Todas las etapas de pruebas funcionales y no funcionales requieren de ambientes de pruebas, por lo que es necesario definir en cada fase de los proyectos procedimientos específicos.
* Es recomendable que el gerente de cambios u operaciones sea diferente al del ambiente de producción, por lo que tiene el trabajo de planear y operar.
* Especificar los requerimientos de configuración, infraestructura y herramientas necesarias.
* La documentación de requerimientos de los proyectos debe incluir los requerimientos de ambientes.
* La utilización y disponibilidad de los ambientes de pruebas deben ser previamente planificados, del mismo modo el equipo de pruebas debe incluir en la planeación de los requerimientos las restricciones mencionadas.
* Se deben tener en cuenta procedimientos de gestión de soporte de ambientes.
* Se deben establecer procedimientos que se puedan auditar para la preparación de datos de prueba, accesos múltiples, actualizaciones, instalaciones, contar con herramientas de *debugging* y procesos de elaboración, asistencia y conectividad a los equipos de pruebas de todas las etapas**.**

**Figura 4**

*Ambientes de pruebas de software*



En la figura anterior se pueden observar los aspectos más importantes para tener en cuenta en la configuración de los ambientes, elementos como definiciones, características y restricciones.

**2.4. Ejecución de las pruebas**

Inicia creando los datos de prueba requeridos para ejecutar los casos de prueba establecidos. La ejecución de los casos de prueba se puede realizar de forma automatizada o manual, en cualquiera de las dos formas se debe registrar y documentar los fallos detectados en el sistema y su posterior corrección.

En el siguiente recurso de aprendizaje se puede observar el flujo de las actividades en relación con la ejecución de pruebas:



**2.5. Creación de plan de pruebas e integración y entrega continua en una metodología ágil**

En un proyecto ágil está compuesto por iteraciones o *sprints*, cada *sprint* tiene una duración que no es muy extensa, aproximadamente dos a cuatro semanas tal como se puede ver la siguiente figura, al final de cada iteración ya están definidas funcionalidades que pueden ser utilizadas por el cliente. En los proyectos agiles existen dos tipos de planificaciones reléase/entrega y sprint/iteración, no obstante, estas se pueden dar en un mismo momento; es decir, que puede incluir una sola planificación con actividades tanto del reléase como del sprint. No obstante, se debe tener en cuenta que en un proyecto ágil las pruebas de *software* se realizan en cada iteración.

**Figura 5**

*Desarrollo y testing en proyectos ágiles*

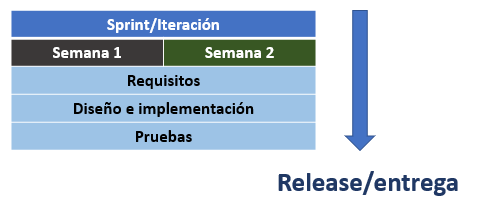


Nota. Tomada de Globetesting (s.f.)

Teniendo en cuenta las iteraciones de un *sprint*, al final de cada iteración ya están definidas funcionalidades que pueden ser utilizadas por el cliente. En los proyectos ágiles existen dos tipos de planificaciones reléase/entrega y sprint/iteración; no obstante, estas se pueden dar en un mismo momento, es decir que puede incluir una sola planificación con actividades tanto del reléase como del *sprint*. La siguiente figura lo representa.

**Figura 6**

*Actividades de los sprints y el release simultáneamente*

****

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

En el siguiente recurso de aprendizaje se presentan los principios de las pruebas ágiles o Agil Testing:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Cuando utilizamos el desarrollo guiado por pruebas (TDD) la planificación de las pruebas ágiles tiene en cuenta los siguientes elementos:

* La integración continúa.
* Pruebas unitarias con TDD.
* Automatización de pruebas.
* Pruebas no funcionales.
* Pruebas (ATDD).
* Los requerimientos deben estar correctamente definidos.

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

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

En el siguiente recurso se presentan los cuadrantes de las pruebas ágiles (*Agile Testing*):

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para empezar con las pruebas descritas en los cuadrantes, es necesario que haya en un ambiente de desarrollo con alguna funcionalidad instalada y desarrollada.

A continuación, se presentan los pasos en el proceso para trabajar con las pruebas especificadas en los cuadrantes:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En el siguiente recurso se describen algunas recomendaciones para tener en cuenta al momento de trabajar con pruebas de *software* ágiles (*Agile Testing*):

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

La creación del plan de pruebas para proyectos ágiles es muy importante, pues en este se definen las funcionalidades que ya están disponibles para el cliente. Por lo tanto, una planificación de un reléase tiene los siguientes elementos:

* **Alcance**: se incluye el propósito del reléase y las historias de usuario.
* **Suposiciones**: se incluyen los supuestos del proyecto.
* **Análisis de riesgos**: define qué se incluye en la automatización de pruebas.
* **Automatización de pruebas:** indica qué se incluirá en la automatización.
* **Ambiente de pruebas:** establece qué se necesita en relación con el ambiente de pruebas.
* **Datos de las pruebas.**
* **Resultados de las pruebas.**

Estos se describen en un plan de pruebas, sin embargo, es posible que en los equipos se tome la decisión de no diseñarlo, en ese sentido se recomienda que los *tester* tomen notas o documenten los factores importantes relacionados con las pruebas en cada *release*.

Para realizar pruebas de *software* en una metodología ágil, se comienza por redactar el plan de pruebas teniendo en cuenta que se puede actualizar en cada *sprint*.

Un plan de pruebas ágiles debe incluir:

* + - 1. Introducción.
      2. Alcance.
      3. Recursos en este caso el nombre de los *tester*.
      4. Descripción de funcionalidades a probar.
      5. Los tipos de pruebas (pruebas de rendimiento, pruebas de aceptación (UAT) que se van a realizar.
      6. Infraestructura lista.
      7. Suposiciones, plan de riesgos y los entregables que se producirán al final.

| **LLAMADO A LA ACCION**  Para aprender cómo realizar el plan de pruebas para proyectos ágiles vea el siguiente anexo donde se entrega un ejemplo: |
| --- |

**3. Gestión del desarrollo y ejecución de pruebas**

La gestión del desarrollo la comprenden la planeación, ejecución, administración, control, seguimiento, medición y evaluación de la ejecución del proyecto, y para esto es necesario apoyarse en una metodología o marco de trabajo.

Todas las fases pueden ser ejecutadas desde diferentes perspectivas, implementando metodologías tradicionales o realizando una implementación de metodologías ágiles, incluso tomar lo mejor de cada aspecto y unificarlo en una metodología híbrida, hoy en día es probable que la mayoría de empresas tengan una implementación propia o híbrida que implique usar las mejores prácticas de los diferentes marcos de trabajo, entre los cuales podemos mencionar algunos de desarrollo de *software* y otros de gestión de proyectos, tales como RUP, XP, Scrum o Kanban.

**3.1 Mínimo producto viable**

MVP (*Minimum Viable Product*) o Producto Mínimo Viable es una versión de prueba de un proyecto, por lo general contempla las funcionalidades base con las cuales deberá contar el elemento a construir. Es una metodología muy usada en la era digital en lo relacionado a la construcción de nuevos productos, que busca evitar perder tiempo y dinero en proyectos que no tendrán la respuesta esperada en el mercado.

La idea de un MVP es poder ofrecer un producto base con las mismas características de valor del producto terminado, dado que un MVP por lo general inicia con un presupuesto base bajo y se busca poner a prueba el mismo para poder fortalecer el músculo financiero ya sea por cuenta propia o por inversión, se debe construir rápido y poniendo los esfuerzos en los puntos correctos y que generan valor.

Para proponer un MVP es importante tener en cuenta el siguiente paso a paso:

Texto

Descripción generada automáticamente con confianza media

**3.2 Aplicación de Scrum y Kanban**

Scrum es un marco de trabajo que cuenta con varios eventos (Sprint, Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective), roles (Scrum Master, Product Owner and Development Team) y artefactos (Product Backlog, Sprint Backlog); y establece los lineamientos para conformar equipos de trabajo de alto rendimiento, auto organizados, impulsados por el trabajo en equipo, la transparencia en la comunicación y sinergia entre la interdisciplinariedad de los equipos de trabajo en el proyecto.

| Este enfoque proporciona una serie de reglas y tareas que se deben ejecutar en cada una de las iteraciones de un proyecto de *software* con la finalidad de asegurar su adecuada implementación. Esto genera como resultado un enfoque muy sencillo, que sus pasos son claros y no dan pie a malas interpretaciones, esto es ideal dado que da menos espacios a suponer cosas y la curva de aprendizaje es mayor al ser aplicada. |  |
| --- | --- |

Por otro lado, Kanban cumple un rol de marco de trabajo ágil como herramienta ya que su misión es aplicar y cumplir una serie de reglas definidas, tales como visualizar el flujo de trabajo, limitar el WIP, definir acciones y políticas explícitas, medir el flujo de trabajo. Al tener reglas tan claras como las de una herramienta, suele ser bastante intuitiva y fácil de implementar, solo es requerido entender su funcionamiento y que cada miembro del equipo lo aplique correctamente para que se ponga en marcha.

Es importante que gracias a sus características de herramienta se puede integrar fácilmente a otros enfoques como por ejemplo Scrum. Uno de los principales problemas que nos puede generar Kanban está relacionado con el WIP, dado que puede llegar a generar cuellos de botella.

**3.3. DevOps**

DevOps, hace referencia a la combinación de los términos en inglés de desarrollo y operaciones (*Development and Operations* - DevOps), lo que indica la sinergia de tecnología, procesos y personas para ofrecer entregas de valor constantes a los clientes. Este permite que haya una interacción entre los roles, teniendo en cuenta el desarrollo, operaciones, calidad y seguridad, permitiendo que se coordinen y colaboren para el desarrollo de los productos, aumentando la calidad y la fiabilidad. Al adoptar una cultura DevOps los equipos podrán responder de mejor manera a las necesidades del cliente, alcanzando los objetivos de negocio en menor tiempo.

Algunos objetivos que se buscan cumplir con la implementación de la cultura y herramientas DevOps son los siguientes:

* Minimizar el tiempo de comercialización.
* Adaptación a la competencia y al mercado.
* Mantenimiento la confiabilidad del sistema y la escalabilidad.
* Mejora el tiempo medio de recuperación.

DevOps puede estar presente en las fases de planeación, desarrollo, implementación, despliegue y aplicación, estas fases no son específicas de un rol, y en la cultura DevOps se busca que todos los roles estén implicados de una u otra forma en todas las fases, como se describen en el siguiente recurso de aprendizaje:

Interfaz de usuario gráfica, Aplicación, PowerPoint

Descripción generada automáticamente

**3.4 Git, Gitflow y Githubflow**

Git es una herramienta de control de versiones diseñada por Linus Torvalds, es un *software* libre, la cual está pensada para mejorar la eficiencia, confiabilidad y compatibilidad entre las diferentes versiones de código fuente. Uno de los propósitos es llevar el registro de los cambios realizados, basada en *branches* (ramas).

La estrategia de *branching* – **GitFlow** es separar el trabajo en diferentes ramas (*branches*), lo que permite adaptarse rápidamente al proceso de trabajo colaborativo que se requiere en un equipo de desarrollo, se basa en dos ramas principales que tienen vida infinita y paralela, denominada así:

* ***Master*:** contiene el código de producción.
* ***Develop*:** contiene el código que ha finalizado desarrollo.

Durante el proceso de desarrollo se crean otras ramas que son de soporte que tienen vida finita, y caducan cuando se realiza el proceso de extracción hacia la rama de desarrollo o *master* según sea el flujo:

* ***Feature*:** se crea a partir de *develop* cuando es requerido el desarrollo de una nueva funcionalidad. Al finalizar el desarrollo se hace la fusión (*merge*) o extracción a la rama *develop*.
* ***Release*:** se crea a partir de *develop* para preparar una nueva versión del código que debe ser liberada en producción. Al finalizar el desarrollo se hace *merge* a *develop* y a *master*.
* Hotfix**:** se crea a partir de *master* cuando es necesario corregir un error detectado en producción de manera urgente, por decirlo así es una solución en caliente. Al finalizar el desarrollo se hace fusión a *develop* y a *master*.

La estrategia de *branching* - **GitHub Flow** es creada por GitHub y es conocida en la comunidad de desarrolladores como una alternativa simple y ligera a GitFlow. GitHub Flow se basa en un flujo de trabajo en ramas que se enfocan principalmente en la entrega continua. En Git Flow, no existen las ramas de *releases*, ya que en esta estrategia se tiene pensado que cada implementación ocurra con frecuencia de ser posible hasta varias veces al día.

En esta estrategia de ramas, en el repositorio tenemos dos tipos de *branches*:

* ***Main* (o *master*):** el *branch* de código principal, es el que contiene el código que está listo para producción.
* ***Features*:** las ramas de funcionalidades que permiten el desarrollo en paralelo.

Cuando se utiliza GitHub Flow, existen seis pasos, los cuales se deben seguir para asegurar la calidad en el código y que este está listo para ser liberado a producción en todo momento:

Interfaz de usuario gráfica, Aplicación, PowerPoint

Descripción generada automáticamente

**3.5. Gestión de pruebas**

En el siguiente recurso de aprendizaje se explica en qué consiste la gestión de pruebas:

Interfaz de usuario gráfica

Descripción generada automáticamente

**3.6. Gestión de incidencias**

Para hablar sobre la gestión de incidencias es necesario tener en cuenta estos conceptos:

Primero identificar que se considera una incidencia, la cual es referida como “una interrupción no planificada de un servicio, o reducción en la calidad de un servicio”. El objetivo de tener un proceso de gestión de incidencias es minimizar el impacto negativo que estas conllevan, y llevar el sistema a su funcionamiento normal, brindando el servicio rápidamente y lo antes posible.

Los objetivos de la gestión de incidencias, se encuentran dentro de la cadena de valor de servicio y tiene como objetivo prevenir y restaurar en el menor tiempo posible cualquier interrupción o retraso en el servicio que afecte al cliente, de tal manera que se minimicen los impactos desde las operaciones comerciales.

La velocidad de recuperación es la base fundamental de este proceso, por lo cual a menudo las incidencias pueden ser abordadas con soluciones temporales, que posteriormente son refactorización para que sean permanentes.

| Una de las herramientas que pueden ayudar a gestionar considerablemente este tipo de problemas es la Mesa de Servicio o *Service Desk*, que ha pasado de ser parte de los procesos estratégicos dentro de las organizaciones, al ser el único punto de contacto con los usuarios. Sin este contacto es posible que se pasen por alto limitaciones en la estructura o en la priorización de incidencias. |  |
| --- | --- |

**3.7. Incidencias funcionales**

Son aquellos problemas que se generan a partir de los requerimientos entregados para la funcionalidad del producto *software*, estas se detectan mientras se ejecutan los casos de prueba en los cuales se pretende verificar que el *software* cumple con el nivel de calidad deseado para ser enviado a producción.

Estas incidencias deben ser reportadas en el informe de pruebas para que sean revisadas y parchadas lo antes posible, cuando una incidencia funcional es detectada en un ambiente previo a producción, este es tomado como un *bug* que deberá ser tratado desde todo el ciclo de desarrollo del *software*, mientras que si es detectado en un ambiente productivo este deberá tratarse directamente como un *hotfix*, en el cual lo más probable es que se cree un parche temporal para luego definir una mejora y enviar un ajuste definitivo.

**3.8. Incidencias no funcionales**

Son aquellos problemas o errores que no se desprende explícitamente de las funciones del *software*, corresponden a los problemas o errores que encontramos en el contexto en el cual el producto *software* se encuentra desplegado. Problemas de carga, problemas de velocidad, problemas de conexión entre componentes, certificados de seguridad caducados, enrutamientos.

Para determinar si hay algún tipo de incidencia no funcional, recordemos que las siguientes son tipos de pruebas que nos ayudan a determinar o encontrar esos estados fallidos:

* **Pruebas de carga:** permiten validar y determinar la respuesta del producto *software* cuando este está sometido a una carga de concurrencia a miles de usuarios o peticiones.
* **Pruebas de rendimiento:** en este tipo de pruebas se busca realizar un cálculo de la respuesta de la aplicación con distintas peticiones, por ejemplo, procesar el ingreso de 10, 100 o 100 usuarios de forma concurrente, y determinar cuál es la curva de rendimiento y con qué valores el servidor se comienza a saturar.
* **Pruebas de estrés:** este tipo de pruebas busca determinar cuáles son los valores máximos que puede resistir el sistema antes de que se caiga, por ejemplo, determinar cuál es la cantidad de usuario que soporta de manera concurrente hasta que llegue un *timeout*.

Este tipo de incidencias son trabajadas por el equipo de infraestructura, el cual, con diferentes herramientas de monitoreo, puede lanzar alertas o mejorar las capacidades de los servidores para tener más recursos y solucionar las fallas no funcionales. Actualmente con las herramientas alojadas en la nube (*cloud*) es más sencillo conocer los estados de las instancias y mejorar la infraestructura.

1. **ACTIVIDADES DIDÁCTICAS**

| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| --- | --- |
| Nombre de la Actividad | Cuestionario de aseguramiento de la calidad de *software*. |
| Objetivo de la actividad | El objetivo de esta actividad es que el aprendiz pueda reconocer los diferentes conceptos del aseguramiento de la calidad de *software* y sus características. |
| Tipo de actividad sugerida | Preguntas de opción múltiple con única respuesta. |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | Anexo\_Actividad\_Didactica1\_CF02\_M6 |

1. **MATERIAL COMPLEMENTARIO:**

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

| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| --- | --- | --- | --- |
| 1. Pruebas de *software* | Escobar, S., M. E., y Fuertes, D., W. M. (2015). Modelo formal de pruebas funcionales de *software* para alcanzar el Nivel de Madurez Integrado 2. *Revista Facultad de Ingeniería UPTC, 24*(39)*.* | Artículo | <https://revistas.uptc.edu.co/index.php/ingenieria/article/view/3549> |
| 2. Construcción, desarrollo y ejecución de pruebas de *software* | Múnera, Á., , J. y Uribe, A., C. (2020). *Desarrollo de aplicaciones web y pruebas de software*.Repositorio digital tdea. | Artículo | <https://dspace.tdea.edu.co/handle/tdea/1081> |
| 2.1. Aseguramiento de calidad de *software* | Pressman, R. S. (1993). *Ingeniería del Software: un enfoque práctico*. Mc Graw-Hill. | Libro | <https://github.com/deberestes01/GeneralTES/blob/master/Ingenieria_del_Software._Un_Enfoque_Practico.pdf> |
| 3.3. DevOps | Guijarro, O., J., Caparrós, R., J., y Cubero, L., L. (2020). *DevOps y seguridad cloud.* (Cap. II – Fundamentos avanzados de DevOps). eLibro. | Libro | <https://elibro-net.bdigital.sena.edu.co/es/ereader/senavirtual/128889?page=27> |

1. **GLOSARIO:**

| TÉRMINO | SIGNIFICADO |
| --- | --- |
| Entrega continua | proceso en el cual se utilizan herramientas tecnológicas casi sin la necesidad de la intervención de personas. |
| DevOps | desarrollo y operaciones. |
| Integración continua | práctica de desarrollo de *software* mediante la que los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. |
| Kanban | marco de trabajo ágil que opera como herramienta y permite visibilizar los estados de una tarea. |
| MVP | Mínimo Producto Viable. |
| QA | calidad de *software*. |
| Scrum | marco de trabajo ágil que estipula una serie de tareas y eventos para realizar iteración que den valor a la entrega continua de un producto. |
| SQA | Aseguramiento de la Calidad de *software*. |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Globetesting. (s. f.). *Automatización de pruebas*. Globetesting. <https://www.globetesting.com/automatizacion-pruebas/>

Grh*software*. (2020). *¿Qué es el modelo V en las pruebas de software?* Grhsoftware.com. [https://grh*software*.com/que-es-el-modelo-v-en-las-pruebas-de-*software*/](https://grhsoftware.com/que-es-el-modelo-v-en-las-pruebas-de-software/)

Rodríguez, C. (2020). *Automatizar pruebas de software: ¿cuándo y por qué?* cl.abstracta.us. [https://cl.abstracta.us/blog/automatizar-pruebas-de-*software*/](https://cl.abstracta.us/blog/automatizar-pruebas-de-software/)

1. **CONTROL DEL DOCUMENTO**

|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| --- | --- | --- | --- | --- |
| Autor (es) | Gustavo Rodríguez | Experto Temático | Regional Atlántico - Centro Colomboalemán | Octubre 2021 |
| Ervin Andrade | Experto Temático | Regional Cauca - Centro de Teleinformática y Producción Industrial | Octubre 2021 |
| Peter Pinchao | Experto Temático | Regional Cauca - Centro de Teleinformática y Producción Industrial | Octubre 2021 |
| Giovanna Escobar | Diseñadora Instruccional | Regional Antioquia - Centro de Servicios de Salud | Octubre 2021 |
| Andrés Felipe Velandia Espitia | Revisor Metodológico y Pedagógico | Regional Distrito Capital – Centro de Diseño y Metrología | Octubre de 2021 |
| Rafael Neftalí Lizcano Reyes | Asesor Pedagógico | Regional Santander - Centro Industrial del Diseño y la Manufactura | Octubre de 2021 |
|  | José Gabriel Ortiz Abella | Corrector de estilo | Regional Distrito Capital – Centro para la Industria de la Comunicación Gráfica. | Octubre del 2021. |

1. **CONTROL DE CAMBIOS**

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

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