**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | Aseguramiento de la calidad de *software* |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA |  | RESULTADOS DE APRENDIZAJE |  |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | CF11 |
| NOMBRE DEL COMPONENTE FORMATIVO | Fundamentos de la calidad del *software* |
| BREVE DESCRIPCIÓN | En este componente formativo, el aprendiz identificará que un *software* con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba, que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del *software.* |
| PALABRAS CLAVE | *Bug*, certificación, errores, incidencias, pruebas. |

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

1. **TABLA DE CONTENIDO:**

**Introducción**

**1. Descripción de la idea de negocio**

**2. Marco de referencia**

2.1. Proceso de desarrollo de *software*

2.2. Estándares ISO/IEC 25000 SQuaRE, ISO/IEC 15504, IEEE.

2.3. Modelo de desarrollo CMMI (*Capability Maturity Model Integration*)

2.4. Calidad en el proceso de desarrollo de *software*

2.5. Modelos de referencia para la calidad en el proceso

2.6. Calidad del *software* relacionada con el producto

2.7. Calidad relacionada con las personas

2.7.1. PSP/TSP

2.7.2. SCRUM

2.7.3. Calidad de software en metodologías ágiles

2.7.4. Metodología XP - Programación Extrema

**3. Documentar el proceso de calidad de *software***

3.1. Diseñar los instrumentos de calidad de *software*

3.2. Aplicar los instrumentos de calidad de *software*

3.3 Análisis y entrega de resultados

1. **DESARROLLO DE CONTENIDO:**

**Introducción**



Bienvenidos a este componente, en donde trataremos el tema relacionado con los Fundamentos de la calidad del *software.*

Recordemos que implementar buenas prácticas, definir adecuadamente los proceso, ser flexibles ante las diferentes metodologías y herramientas, y tener un personal capacitado en las fases de pruebas y calidad, se convierte en un desafío cada vez más grande, al cual se ven enfrentadas las empresas que construyen *software.*

Dadas las circunstancias actuales, la demanda de construcción de productos de *software* va en auge, teniendo un crecimiento exponencial año por año. Si agregamos las iniciativas de modernización de las tecnologías y la tendencia de sistematizar y facilitar los procesos, implica tener más células de desarrollo operando, y mejoras en las fases de calidad del *software.*

En este componente formativo, usted podrá abordar los fundamentos de la calidad del *software,* entendiendo los valores misionales de las empresas, y cómo estos repercuten en los diferentes procesos de control de la misma. La calidad es una característica que debe ser transversal en todos y cada uno de estos procesos, porque de esta forma podremos asegurar éxito y productividad. Un quipo capacitado, comprometido y creativo es un valor agregado en la industria del software, que cada vez va en aumento, pero que también exige compromiso de todos.

**1. Descripción de la idea de negocio**

La idea de negocio es el servicio o producto que se desea ofrecer al mercado, lo cual se requiere de un plan de negocio que debe incluir misión, visión, objetivos de la empresa y la estrategia para alcanzarlos, así como la estructura organizacional y la inversión para financiar el proyecto. No obstante, todo empieza cuando se tiene una idea de negocio para posteriormente materializarla por escrito en un documento teniendo en cuenta cinco elementos:

|  |
| --- |
| CF11\_1\_Descripcion\_idea\_negocio |

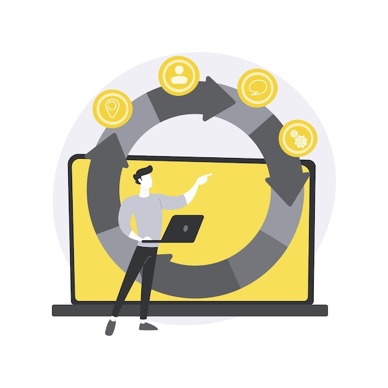
La estructura ideológica corresponde con los valores misionales de cualquier empresa, por lo tanto, es necesario describir en la idea de negocio, la misión, la visión y los objetivos que se intentan lograr. La presentación ante sus potenciales inversionistas y demás colaboradores, recae de alguna forma en esta claridad, desde lo que es actualmente la empresa y cómo se proyecta a un futuro.

**2. Marco de referencia**



Un marco de referencia aporta las directrices y saberes teóricos obtenidos con base en un estudio, una investigación y un análisis previo, que permite estimar en este caso, que un proceso de calidad fundamentado a través del estudio e investigación, estaría en la capacidad de plantear mejores prácticas y definir procedimientos que optimicen dicho proceso, para garantizar que este represente una mejora en el proceso de calidad y en la calidad del producto. Por lo anterior, este marco de referencia contiene los fundamentos de la calidad del *software* que permitirá construir un plan de pruebas, teniendo en cuenta el proceso de desarrollo de *software,* los modelos de desarrollo, metodologías de desarrollo tradicionales y ágiles, estándares de calidad, planificación de pruebas de *software* y la documentación que se debe realizar en el proceso de calidad de *software*, tal como se describen a continuación.

**2.1 Proceso de desarrollo de *software***



El proceso de desarrollo o ciclo de vida del *software*, requiere de una serie de etapas, con el objetivo de garantizar que el programa desarrollado es fiable, eficiente, seguro y cumple con los requerimientos de los usuarios finales.

Este proceso de desarrollo contiene una serie de fases que permiten validar el proceso de desarrollo del *software*, garantizando que se cumplan con los requerimientos para la verificación y aplicación de los procedimientos de desarrollo, comprobando las metodologías y técnicas usadas. Este proceso es muy aplicable en los casos en donde se utilicen metodologías tradicionales como la metodología en cascada.

**Metodología cascada**

El desarrollo en cascada es un proceso lineal por lo que se identifica en separar los procesos del desarrollo de *software* en diferentes fases que son sucesivas en el contexto de la ejecución del proyecto. A diferencia de los modelos iterativos, cada una de las fases se ejecuta una sola vez, y el resultado de cada fase es un insumo para la siguiente. Esta metodología es usada especialmente en el proceso de desarrollo de *software.*

|  |
| --- |
| CF11\_2\_1\_Metodologia\_cascada |

**2.2. Estándares ISO/IEC 25000 SQuaRE, ISO/IEC 15504, IEEE**

|  |  |
| --- | --- |
| Certificación ISO: la clave para garantizar la gestión de calidad |  Administración | Apuntes empresariales | ESAN | En un proceso de calidad de *software*, se debe identificar las normas ISO (Organización Mundial para la Estandarización), que proporcionan guías que a su vez aportarán especificaciones, de aspecto internacional, a los servicios, productos y sistemas de una empresa, para garantizar una óptima eficiencia y calidad en cuanto a un producto o servicio y a su funcionamiento y resultados. Por lo tanto, se deben tener en cuenta los estándares descritos en este apartado con el objetivo de seguir patrones que permitan asegurar la calidad a través del análisis de los estándares que se utilizan en la actualidad. |

La norma ISO/IEC 25000, también reconocida como *System and Software Quality Requirements and Evaluation (SQuaRE)*, tiene el objetivo de especificar en un documento el conjunto de guías y patrones que aseguran la óptima utilización del *software* de un negocio. No obstante, este procedimiento se realiza en compañías que realizan sus propias aplicaciones para la organización y gestión interna, en consecuencia, se refuerza la creación de forma correcta y usabilidad en los procesos internos.

La ISO/IEC 25000, evoluciona de anteriores normas tales como las normas ISO/IEC 9126, que describe las características de un modelo de calidad del producto *software,* e ISO/IEC 14598, que involucraba el proceso de evaluación de productos *software.*

La familia de normas ISO/IEC 25000 está constituida como se aprecia a continuación.

|  |
| --- |
| CF11\_2\_2\_Normas\_de\_calidad |

**Ventajas y desventajas de la ISO 25000**

Cuando se implementa esta norma, es importante reconocer tanto las ventajas como las desventajas de la ISO 25000, que mostrará un panorama para entender su reconocimiento en el sector de la calidad.

Ventajas:

* Asegurar que la aplicación, servicios y productos que se vendan o utilicen tienen la confianza y calidad requerida.
* Minimiza la cantidad de errores (Bugs).
* Permite ahorrar tiempo en los procedimientos de entrega.
* Determina la constante calidad a partir de evaluaciones periódicas.
* Lo anterior genera en el cliente un aumento de la satisfacción.

Desventajas:

* No se definen desventajas específicamente en cuanto a estas normas puesto que se crea un marco de calidad a nivel internacional.
* Tener en cuenta estas normas es un proceso tedioso, no obstante, el resultado es óptimo y positivo.

 **2.3 Modelo de desarrollo CMMI (*Capability Maturity Model Integration*)**

Este modelo es uno de los que más utilizan las empresas de desarrollo de *software,* por lo tanto, se debe comprender que si se utiliza este modelo, su objetivo es auditar el cumplimiento de normas de calidad partiendo de la medición con niveles de madurez, además se enfoca en la mejora continua del producto en todo su ciclo de vida, impulsar la eficiencia y asegurar la calidad, de esta forma se reducen los riesgos en el proceso de desarrollo.

Cuando se desea mejorar algunos procesos de negocio, construir productos de calidad para satisfacer las necesidades del cliente y colaborar en el cumplimiento de los requerimientos de la norma ISO e instaurar prácticas de mejora continua, se debe utilizar este modelo puesto que es un modelo de capacidad y madurez, que además permite efectuar una apreciación de la madurez del proceso de desarrollo de *software* en las empresas.

Los objetivos de CMMI, son:

* 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.

CMMI presenta dos modelos, una está estructurada en cinco niveles de madurez y la otra está estructurada en seis niveles de capacidad. Las áreas de proceso están categorizadas en administración de procesos, administración de proyectos, ingeniería y soporte.

**Estructurada por nivel de madurez**

Un nivel de madurez bien definido con las bases necesarias para evolucionar hacia un proceso de *software* maduro.

Los niveles representados por etapas, consideran cinco niveles de madurez, los cuales contienen un conjunto de áreas de proceso predefinidas, los cuales se miden por objetivos generales y específicos, estos se aplican a cada conjunto predefinidos de áreas de proceso. Veamos su distribución:

|  |
| --- |
| CF11\_2\_3\_Modelo\_CMMI |

**Estructurada por nivel de capacidad**

|  |  |
| --- | --- |
| Vector gratuito ilustración de punto de referencia de diseño plano dibujado a mano | Un nivel de capacidad bien definida y establecida sobre una meseta evolutiva, permite describir la capacidad de la organización en relación con un área de proceso determinada, en donde deben tenerse en cuenta las prácticas específicas y genéricas, las mismas que puedan ser extendidas a otras áreas o a otros procesos de la organización relacionados. Cada nivel es una capa en la base que permite una mejora continua con respecto a un proceso particular.  En otras palabras, podemos decir que los niveles de capacidad son acumulativos, los cuales a medida que se está en las capas superiores de los niveles, también se impactan las características de los inferiores. |

En CMMI en una representación continua se cuenta con seis niveles de capacidad asignados por los rangos de números del 0 al 5. Esta representación es un conjunto predefinido de áreas de proceso, que se mide por el logro de los objetivos tanto genéricos como específicos, descritos de la siguiente manera:

* 0 – Incompleto: estar en el nivel de capacidad 0, 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.
* 1 - Se ha realizado.
* 2 – Administrativo: se denominan como procesos administrados, porque se puede contar con una planeación, una trazabilidad y un control en los proyectos individuales o grupales.
* 3 – Definido: 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.
* 4 - Cuantitativamente gestionado: 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.
* 5 – Optimizado: este nivel se centra en la mejora continua del rendimiento de los procesos por medio de mejoras continuas que buscan la innovación. Mientras que en el nivel 4 el enfoque es crear un marco de referencia y las mediciones de rendimiento, el nivel 5 se centra en buscar causas comunes para los problemas y arreglar esos problemas de proceso.

**2.4 Calidad en el proceso de desarrollo de *software***



Recordemos que la construcción de *software* está basada en unas fases que permiten incrementar el producto desde sus cimientos hasta obtener un producto base; esto es posible debido a 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, demorando posiblemente el tiempo, los recursos y el presupuesto asignado.

La calidad de *software* es un área importante en el proceso de desarrollo que busca estar en constante validación de la calidad del producto *software*, por lo general en el diseño de un plan de aseguramiento de calidad. Por ello es de importancia tener en cuenta los siguientes conceptos y su aplicación práctica en el ciclo de desarrollo de *software*:

Es un conglomerado de características de una entidad, que son esenciales y que además permiten calificar su valor.

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.

Calidad

Es 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.

Calidad de *software* (QA)

Es 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.

Aseguramiento de la calidad de *software* (SQA)

El propósito principal del SQA es detectar diversos problemas en las fases iniciales del desarrollo del *software,* estimando que es menos costosa su corrección.

Propósito del aseguramiento de la calidad de *software* (SQA)

La calidad del *software* puede medirse hasta la fase de implementación, pero de ser detectados allí los problemas de construcción del producto, implicaría costos extras, por esta razón se aconseja atender y solucionar los posibles errores en las fases iniciales, como en el diseño. 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 de éste y lo que debe garantizar para satisfacer las necesidades implícitas y explícitas. Si el diseño y los requerimientos no están correctamente definidos, es decir que son ambiguos, incompletos o son incoherentes, al momento de implementar el producto va a ser muy costosa su corrección.

**Funciones del aseguramiento de la calidad del *software* (SQA)**

El rol para SQA es brindar a la metodología de desarrollo de *software,* ya la administración, la seguridad de que procesos oficialmente establecidos están siendo implementados, teniendo en cuenta:

|  |
| --- |
| CF11\_2\_4\_Funciones\_SQA |

**Principios de aseguramiento de la calidad de *software* (SQA)**

Para poder realizar el aseguramiento de la calidad de *software (*SQA), se deben tener en cuenta diversos principios básicos:

|  |
| --- |
| CF11\_2\_4\_Principios\_SQA |

**Componentes del aseguramiento de la calidad de *software* (SQA)**

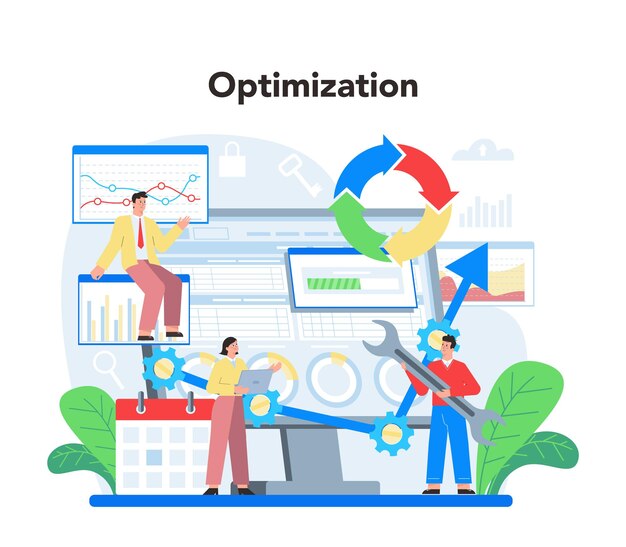
El aseguramiento de la calidad de *software*, debe incluir los siguientes componentes:

|  |
| --- |
| CF11\_ 2\_4\_Componentes\_SQA |

**Proceso del aseguramiento de la calidad de *software* (SQA)**

En el proceso de aseguramiento de la calidad de s*oftware,* se encuentran tres etapas:

|  |
| --- |
| CF11\_ 2\_4\_Proceso\_SQA |

 **2.5 Modelos de referencia para la calidad en el proceso**

En la calidad de *software* es importante involucrar el valor de los requerimientos explícitos e implícitos del producto, con el propósito de medir la calidad, los estándares y modelos de calidad existentes.

Estos modelos de calidad tienen un conjunto de factores que pueden ser medidos directamente como unidades de tiempo y errores, pero también indirectamente, como la facilidad de mantenimiento.

Los modelos de calidad son documentos que constituyen una parte importante de las mejores prácticas, plantean aspectos de administración en la cual las organizaciones deben hacer énfasis, además componen diversas prácticas que facilitan la medición de los avances en la calidad.

Los estándares de calidad además de ser una guía para alcanzar la calidad y la productividad, permiten especificar una serie de criterios de desarrollo para orientar la manera en que se emplea la ingeniería de *software.* A continuación, se nombran los modelos más importantes que evalúan la calidad del producto de *software.*

**Modelo Mc Call**

Este modelo fue diseñado por Jim Mc Call en 1977. Para el análisis de la calidad de *software*, este modelo determina 3 aspectos, describe 11 factores y menciona 23 criterios. Las métricas encontradas en este modelo son una serie de preguntas que ponderan un determinado atributo del producto de *software* numéricamente. En la siguiente tabla se establecen los criterios de calidad relacionados con los factores de calidad.

**Tabla 1**

*Modelo Mc Call, factores de calidad del software*

|  |  |  |
| --- | --- | --- |
| PERSPECTIVAS | FACTORES | CRITERIOS |
| Operatividad del producto: factores de calidad que influyen en el grado en que el *software* cumple con su especificación. | Usabilidad: facilidad de uso del *software.* | Operatividad.  Entrenamiento.  Comunicación. |
| Integridad: protección de programa del acceso no autorizado. | Control de acceso.  Auditoría de acceso. |
| Corrección: grado en que una funcionalidad coincide con su especificación. | Rastreabilidad.  Completitud.  Consistencia. |
| Fiabilidad – confiabilidad: capacidad de los sistemas de no fallar / la medida en que falla el sistema. | Consistencia.  Exactitud.  Tolerancia a fallos. |
| Revisión del producto: factores de calidad que influyen en la capacidad de cambiar el producto *software.* | Mantenibilidad: esfuerzo requerido para localizar y arreglar un fallo en el programa dentro de su entorno operativo. | Simplicidad.  Concreción. |
| Facilidad de prueba: facilidad del programa de realizar pruebas para asegurarse de que está libre de errores y cumple con su especificación. | Simplicidad.  Instrumentación.  Auto-descripción.  Modularidad. |
| Flexibilidad: facilidad de hacer los cambios necesarios según lo solicitado en el entorno operativo, | Auto-descripción.  Capacidad de expansión.  Generalidad.  Modularidad. |
| Transición del producto: factores de calidad que influyen en la capacidad de adaptar el *software* a los nuevos entornos. | Reusabilidad: facilidad de reutilización de *software* en un contexto diferente. | Auto-descripción.  Generalidad.  Modularidad. |
| Interoperabilidad: esfuerzo requerido para acoplar el sistema a otro sistema. | Modularidad.  Similitud de comunicación.  Similitud de dato.  Independencia del sistema.  Independencia de la máquina. |



El modelo de Mc Call es uno de los primeros desarrollados, no obstante, gran parte de los factores descritos son utilizados en la actualidad, además gran variedad de modelos de calidad, incluyendo la Norma ISO 9126 son una estandarización de este modelo.

**Modelo FURPS**

Este modelo lo desarrolló Hewlett-Packard en el año 1987. Incluye el desarrollo de una serie de factores de calidad de *software,* teniendo en cuenta el acrónimo de **FURPS**: funcionalidad (*Functionality*), usabilidad (*Usability*), confiabilidad (*Reliability*), desempeño (*Performance*) y capacidad de soporte (*Supportability*). En la siguiente tabla se establecen los factores y los criterios de calidad.

**Tabla 2**

*Modelo FURPS, factores de calidad del software*

|  |  |
| --- | --- |
| FACTORES | CRITERIOS |
| Funcionalidad | Características y capacidades del programa.  Generalidad de las funciones.  Seguridad del sistema. |
| Usabilidad | Factores humanos.  Factores estéticos.  Consistencia de la interfaz.  Documentación. |
| Confiabilidad | Frecuencia y severidad de fallos.  Exactitud de las salidas.  Tiempo medio de fallos.  Capacidad de recuperación ante fallos.  Capacidad de predicción. |
| Rendimiento | Velocidad de procesamiento.  Tiempo de respuesta.  Consumo de recursos.  Rendimiento efectivo total.  Eficacia. |
| Capacidad de soporte | Extensibilidad.  Adaptabilidad.  Capacidad de prueba.  Capacidad de configuración.  Compatibilidad.  Requisitos de instalación. |

**Modelo BOHEM**

Este modelo plantea una jerarquía en forma de árbol con tres ramas y con ciertos niveles, lo cual hace que el *software* sea de gran utilidad, enfocándose en tres aspectos: portabilidad, facilidad de uso y facilidad de mantenimiento. El modelo BOHEM se diseña en los siguientes niveles:

* Aplicaciones primarias.
* Construcciones Intermedias (factores).
* Construcciones primitivas.
* Métricas que establecen valores para los criterios (construcciones primitivas).

En la siguiente tabla se establecen los factores y criterios.

**Tabla 3**

*Modelo BOHEM, factores de calidad del software*

|  |  |
| --- | --- |
| FACTORES | CRITERIOS |
| Portabilidad | Independencia dispositivos.  Completitud. |
| Fiabilidad | Completitud.  Exactitud.  Consistencia. |
| Eficiencia | Eficiencia dispositivo.  Accesibilidad. |
| Ingeniería humana | Accesibilidad.  Comunicatividad.  Estructuración.  Auto-descripción. |
| Comprensibilidad | Consistencia.  Estructuración.  Auto-descripción.  Concisión.  Legibilidad.  Expansibilidad. |
| Modificabilidad | Estructuración. |

**ISO/IEC 9126**

Es un estándar internacional para la evaluación de la calidad del *software*, que se aplica a diversos tipos de *software*. No obstante, en el 2005 fue reemplazado por el conjunto de normas Square. Sus características se pueden observar en la siguiente tabla.

**Tabla 4**

*Modelo ISO/IEC 9126, criterios asociados a factores de calidad*

|  |  |
| --- | --- |
| FACTORES | CRITERIOS |
| Funcionalidad | Adaptabilidad.  Exactitud.  Interoperabilidad.  Seguridad. |
| Usabilidad | Comprensibilidad.  Aprendizaje.  Operatividad.  Atractivo. |
| Mantenibilidad | Análisis.  Cambio.  Estabilidad.  Prueba. |
| Fiabilidad | Madurez.  Tolerancia a fallos.  Recuperabilidad. |
| Eficiencia | Comportamiento del tiempo.  Uso de los recursos. |
| Portabilidad | Adaptabilidad.  Instalación.  Coexistencia.  Reemplazo. |

Nota. https://bit.ly/3kL4JWu

**2.6 Calidad del *software* relacionada con el producto**

Los clientes determinan según su percepción, que el producto *software* satisface sus necesidades y expectativas a partir de aplicar un proceso de calidad de *software*, puesto que se establecen actividades para asegurar que el desarrollo de *software* en todos los proyectos es de calidad. Por otra parte, el propósito de asegurar la calidad de un producto es detectar diversos problemas en las fases iniciales del desarrollo del *software,* estimando que es menos costoso su corrección. Cabe aclarar que la calidad de *software* de un producto 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.

|  |  |
| --- | --- |
| 6 Criterios de calidad de productos | Blog MBA Cámara de Comercio de Málaga | Una de varias funciones de la calidad de *software* es establecer estándares, procesos y planes, con el objetivo de satisfacer las políticas de la organización y se ajusten a las necesidades de cada proyecto en particular, para asegurar la calidad del producto *software*, por lo tanto, la calidad del *software* está dividida en:  ● La calidad del producto obtenido.  ● La calidad del proceso de desarrollo. |

Las anteriores dependen una de otra, puesto que para alcanzar calidad en un producto es pertinente que haya calidad en el proceso de desarrollo. El objetivo que se determine para la calidad del producto, establecerá los propósitos del proceso de desarrollo. Los requerimientos de calidad más significativos del proceso de *software* son:

* Que genere los resultados esperados.
* Los resultados deberán estar basados en definiciones correctas.
* Los resultados deberán ser mejorados teniendo en cuenta los objetivos del negocio.

**Factores que afectan o determinan la calidad del *software***

El objetivo general de la ingeniería del *software*, es la producción de *software* con capacidad para realizar con exactitud las tareas expresadas en su especificación, y esta calidad puede ser considerada desde dos perspectivas diferentes: la óptica del desarrollador y la del cliente o usuario final. Los factores que afectan al desarrollador se denominan externos e interno.

A continuación, se presenta una breve explicación de cada uno de ellos:

|  |
| --- |
| CF11\_2\_6\_Calidad\_producto |

**Costo de los defectos**

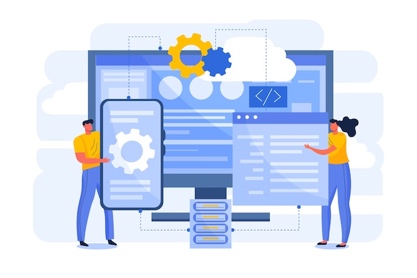
Los costos de calidad son importantes para conseguir productos de calidad. Por lo que se establecen actividades para prevenir, mejorar o detectar la baja calidad que puede haber en los productos *software.*

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.

**Conclusión de los factores que afectan o determinan la calidad del *software***

Cuando calificamos 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 debemos 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.

**2.7 Calidad relacionada con las personas**



La mejora de la calidad de *software* contiene procesos que ayudan a que los ingenieros de *software* manejen, controlen y mejoren el trabajo, teniendo también la oportunidad de construir equipos auto dirigidos y ser participantes efectivos del mismo, pero se debe tener en cuenta que primero deben tener el conocimiento para controlar y mejorar su trabajo, y después tener el conocimiento para trabajar en equipo. Por lo anterior el ingeniero podrá plantear mucho mejor el trabajo, medir la calidad del producto, mejorar las técnicas y determinar las medidas estándares para desempeño y calidad. En la calidad relacionada con las personas se establecen dos procesos PSP y TSP que proporcionan métodos detallados de planificación y estimación.

**2.7.1 PSP/TSP**

Personal *Software Process* (PSP) es un proceso diseñado para ayudar a los ingenieros de *software* en el control, manejo y mejora del trabajo que realizan, el cual está basado en la calidad del *software*; para el caso de *Team Software Process* (TSP) es un marco para el desarrollo de *software* que pone igual énfasis en el proceso, producto y trabajo en equipo.

El PSP tiene un proceso definido que ayuda a realizar mejor los trabajos, con el fin de reportar y conseguir datos completos y precisos del trabajo que se efectúa de manera individual, por lo tanto, mejora el desempeño personal, afectando de esta forma el trabajo del equipo. Por otro lado, contiene una serie de prácticas exactas para la mejora de la productividad personal y la gestión del tiempo de los ingenieros de *software* o desarrolladores, en actividades de mantenimiento de sistemas y desarrollo. Está planteado para ser utilizado en modelos de procesos CMMI o ISO 15504 y está orientado tanto a estudiantes como a ingenieros juniors.

El proceso de esta metodología se clasifica en tres fases:

|  |
| --- |
| CF11\_2\_7\_PSP |

En cada fase se realiza un criterio de entrada, en la cual se describe el problema, el plan del proyecto y registros de tiempo. En los criterios de salida se realizan los reportes y se tiene en cuenta la documentación que se realizó durante el proyecto.

A continuación, se nombran los formatos utilizados en **PSP** y que se fundamentan generalmente en dos medidas importantes:

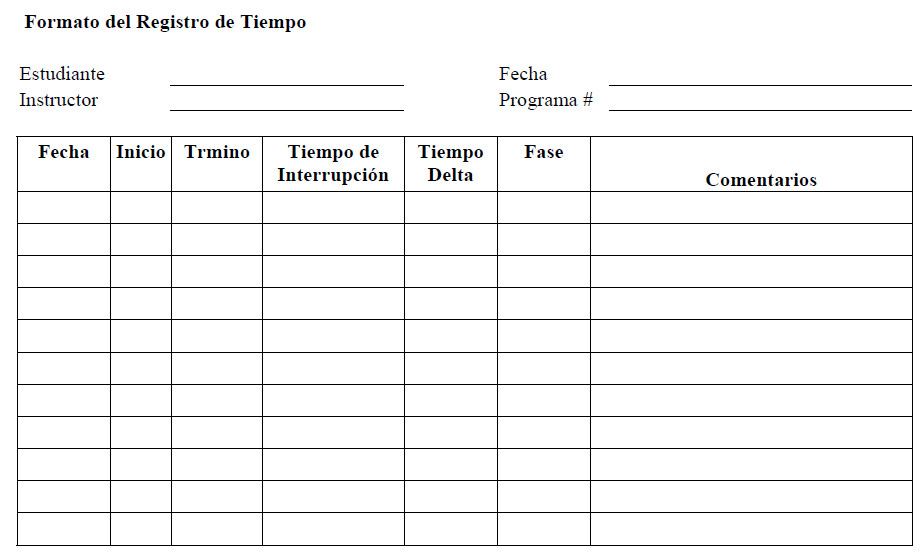
* El tiempo que se emplea en cada fase.
* Los defectos que se encuentran en cada fase.
  1. **Formato de registro de tiempo**

El objetivo de este formato es registrar el tiempo que se emplea en cada fase del proyecto. No obstante, los datos se utilizan para completar el resumen del plan del proyecto. El tiempo se registra en minutos y deben ser exactos.

Un formato de registro de tiempo, debe contener los elementos mostrados en la siguiente figura:

**Figura 1**

*Ejemplo formato de registro de tiempo*



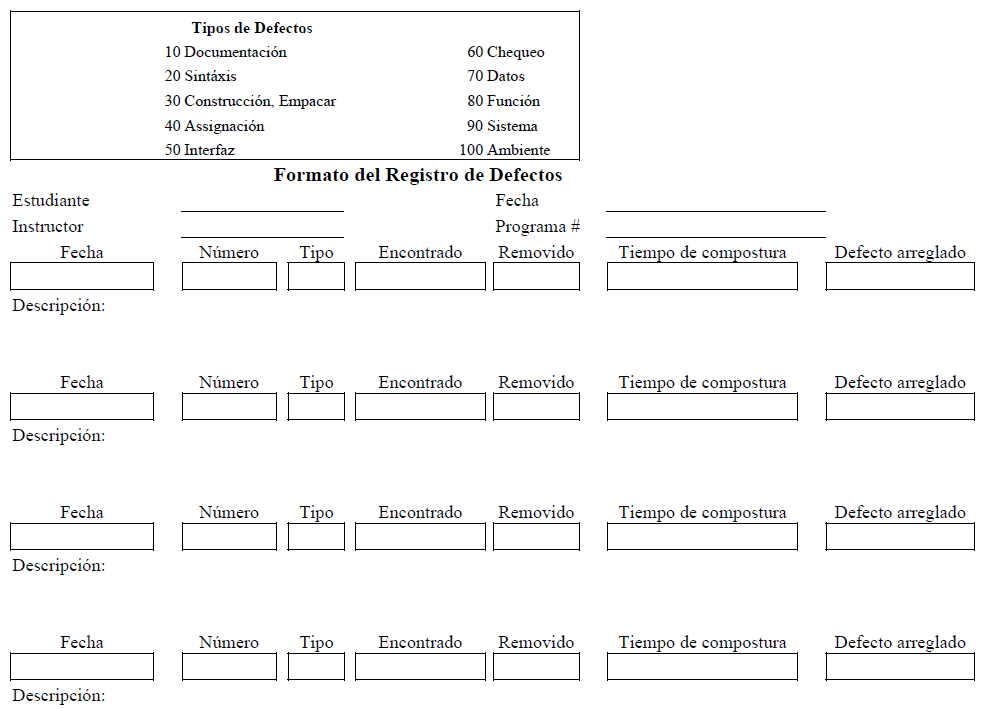
* 1. **Formato de registro de defectos**

El objetivo de realizar este registro de defectos está en fomentar la mejora continua en cada proyecto que se realice.

Un formato de registro de defectos, debe contener los elementos que se muestran a continuación:

**Figura 2**

*Ejemplo formato de registro de defectos*



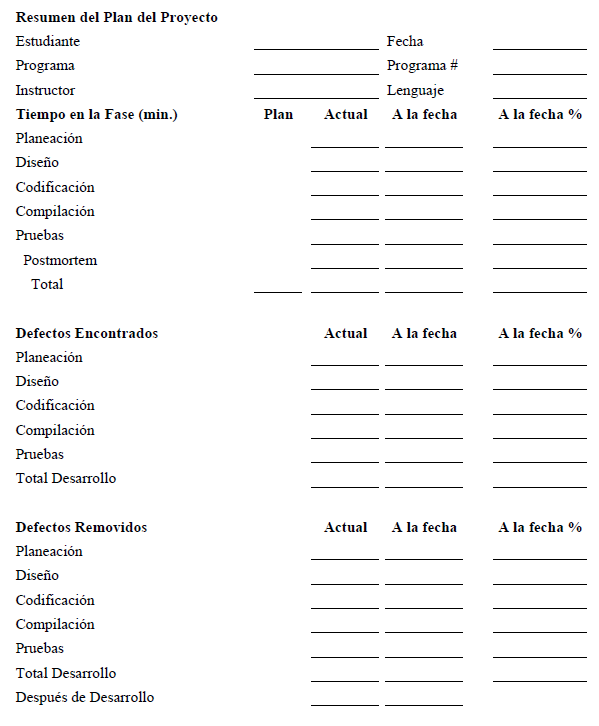
* 1. **Resumen del plan del proyecto**

Este formato define un histórico de todos los proyectos realizados, en él se encuentran los datos que serán útiles en el proyecto siguiente, por lo tanto, es importante que los datos se redacten con precisión y calidad para tener un margen de comparación con futuros proyectos.

Un formato de resumen del plan del proyecto debe contener los elementos mostrados en la siguiente figura:

**Figura 9**

*Ejemplo resumen del plan del proyecto*



**TSP**

|  |  |
| --- | --- |
| TSP (Team Software Process) timeline | Timetoast timelines | *Team Software Process* (TSP) es un proceso de desarrollo que está diseñado para orientar a equipos de personas en la planeación, diseño y desarrollo de sistemas de *software* de calidad. Esta metodología administra el desarrollo de los procesos de *software*, garantizando un entorno de trabajo natural y agradable para los equipos. |

El TSP contiene una serie de pasos estructurados con indicaciones para realizar las actividades en cada fase del desarrollo del proyecto, además es una herramienta útil referente a formación de equipos para el desarrollo de *software* de calidad, generando una planificación que permite determinar las responsabilidades y los roles en los equipos de trabajo. Para ello el TSP se basa en el PSP para formar profesionales con condiciones idóneas para la realización de proyectos demasiado grandes, además este marco contiene las características para generar planes detallados, utilizar datos de procesos, medir y gestionar la calidad del producto y detallar procesos operacionales.

Los objetivos del TSP son:

* Conformar equipos de tal forma que tengan la capacidad de registrar y planear su trabajo, constituir metas bien definidas y que tengan la aptitud para mejorar su propio trabajo a través de la medición del mismo.
* Establecer un marco en base a PSP.
* Establecer estándares para medir la calidad y el comportamiento.
* Suministrar métricas para equipos.
* Evaluar los equipos, las responsabilidades y los roles.
* Guías para solución de problemas que se generen en los equipos.
* Establecer una guía para que la mejora continua de procesos esté activa.

A continuación, se presentan las fases del ciclo de vida del TSP, donde se describe unas pautas para realizar un buen desarrollo de *software* en cabeza del equipo de trabajo.

|  |
| --- |
| CF11\_2\_7\_Ciclo\_de\_vida\_TSP |

**2.7.2 SCRUM**

SCRUM es un marco de trabajo ágil de muy amplio uso en la industria del *software* que se fundamenta en los valores y principios ágiles definidos en el Manifiesto Ágil (2001) y donde se definen tres pilares fundamentales (SCRUMstudy, 2013):

|  |
| --- |
| CF11\_2\_7\_2\_Scrum |

Adicionalmente, este marco de trabajo ágil, está estructurado por un conjunto de roles, eventos y artefactos como se observa en la siguiente figura.

|  |  |
| --- | --- |
| Dentro del equipo, hay roles que se dividen en dos categorías fundamentales: los roles centrales que hacen referencia a los requeridos obligatoriamente para la creación de un producto, están altamente comprometidos y de los cuales depende el éxito o no de un proyecto, y también están los roles no centrales que hace referencia a todos el personal interesado en el proyecto, pueden interactuar con el equipo pero no son los responsables del éxito del mismo, dentro de esta categoría entran los *stakeholders*, directivos, gerentes, marketing, asesores, etc. |  |

Existen tres roles centrales dentro del marco de trabajo de Scrum:

|  |
| --- |
| CF11\_2\_7\_2\_Roles |

Además de los roles, Scrum define un conjunto de eventos con participantes y objetivos claros que se desarrollan en momentos particulares del flujo general de Scrum, a continuación, se detalla cada uno de estos eventos:

* ***Sprint*:** es el corazón de Scrum y se refiere a una iteración que está acotada generalmente por un lapso entre 2 y 4 semanas, donde se realiza un ciclo completo de actividades de análisis, diseño, construcción y pruebas, para desarrollar una versión del producto potencialmente entregable al cliente.
* **Planeación de *Sprint*:** reunión realizada justo antes del inicio de un *Sprint,* donde se definen el subconjunto de requerimientos (*Sprint Backlog*) a ser desarrollados en el siguiente *Sprint* y cómo será el proceso requerido para hacer la entrega del siguiente *Sprint*, lo cual incluye detallar los requerimientos en tareas concretas, estimación de tiempos/esfuerzo y distribución inicial de responsabilidades. Dependiendo de la duración del *Sprint* este tiempo de planificación puede variar, pero la métrica establecida para *Sprint* de 4 semanas corresponde a una planeación de *Sprint* de 8 horas.
* **Reunión diaria (*daily meeting*):** reunión realizada generalmente al inicio de cada día donde el equipo informa en que ha venido trabajando, qué cosas realizaré en el día y qué problemas se le han presentado. Es una reunión corta que se realiza de pie y que debe tener una duración alrededor de los 15 minutos. Esta reunión se alinea con los pilares de transparencia e inspección.
* **Revisión del *Sprint*:** reunión realizada al finalizar el *Sprint*, donde el equipo de desarrollo muestra los resultados del *Sprint.* Para *Sprint* de 4 semanas se usa reunión de revisiones 4 horas.
* **Revisión de retrospectiva:** última realizada luego de la revisión del *Sprint* y tiene como objetivo la autoevaluación personal y del grupo sobre el desempeño del *Sprint* que acaba de finalizar. En esta reunión se identifican y documentan los aprendizajes por medio de diferentes técnicas en las que generalmente se busca dar respuesta a las siguientes preguntas: ¿qué funcionó bien y se debe seguir haciendo?, ¿que no funcionó bien y se debe dejar de realizar? y ¿qué debemos empezar a mejorar? Para un *Sprint* de 4 semanas se utilizan 3 horas para esta reunión.

Finalmente, el marco de trabajo Scrum define un conjunto de artefactos que permiten registrar y gestionar información clave, para asegurar los tres pilares fundamentales y proveen información valiosa durante todo el proceso de desarrollo de *software*. Entre los artefactos representativos de Scrum encontramos:

|  |
| --- |
| CF11\_2\_7\_2\_Artefectos |

Entre los principales beneficios del marco de trabajo Scrum, encontramos:

* Es posible gestionar las expectativas del cliente de manera regular ya que este puede y debe participar en las reuniones de revisión por lo que está enterado todo el tiempo del estado actual del proyecto.
* El cliente puede obtener resultados importantes y utilizables desde las primeras iteraciones ya que la lista de producto está priorizada para ofrecer mayor valor en el menor tiempo posible y porque cada finalización de *Sprint* debe tener como resultado una versión totalmente funcional.
* El proyecto puede iniciar con requerimientos de muy alto nivel y es fácil administrar los cambios.
* La participación constante del cliente permite mitigar riesgos del proyecto desde sus primeras etapas.
* Los procesos de retrospectiva permiten establecer actividades permanentes de mejora continua en función de las experiencias del equipo.

**2.7.3 Calidad de *software* en metodologías ágiles**

La vertiente ágil en el desarrollo de *software* pretende distribuir de forma permanente y continua el proceso de desarrollo distribuido en iteraciones rápidas.

Sin embargo, el término de “metodología ágil” es engañoso dado que implica que el enfoque ágil es la única manera de abordar el desarrollo de *software*. En esta vertiente no se define una serie de pasos o de indicaciones sobre el qué hacer en el proceso de desarrollo de *software*, por el contrario, trata de una forma de pensar en colaboración y los flujos de trabajo, en el cual se definen una serie de valores que orientan las decisiones respecto a lo que se hace y la forma en que se hace.

Las metodologías ágiles buscan generar una serie de piezas que agregan valor y satisfacción al cliente. En estas metodologías se implementan enfoques flexibles y de trabajo en equipo para mejorar el performance del proceso, con la finalidad de ofrecer mejoras constantes.

**Valores de la metodología ágil**

La metodología ágil nació aproximadamente en el año 2001, dando respuesta a los proyectos gestionados por medio de metodologías en cascada, por lo cual una serie de desarrolladores redactó el manifiesto ágil. En este manifiesto se describieron cuatro características principales las cuales se deberían priorizar por encima de cualquier otra cosa. Por lo cual, los equipos que trabajan bajo esta vertiente deben valorar lo siguiente:

* Las personas y las interacciones antes que los procesos y las herramientas.
* El *software* en funcionamiento antes que la documentación exhaustiva.
* La colaboración con el cliente antes que la negociación contractual.
* La respuesta ante el cambio antes que el apego a un plan.

En un proyecto ágil vamos a tener iteraciones o *sprints*, cada *sprint* tiene una duración que no es muy extensa aproximadamente dos a cuatro semanas; al final de cada iteración ya estándefinidas funcionalidades que pueden ser utilizadas por el cliente. En los proyectos ágiles existen dos tipos de planificaciones: *release*/entrega y *sprint/*iteració*n*; no obstante, estas se pueden dar en un mismo momento, es decir, que puede incluir una sola planificación con actividades tanto del *release* como del *sprint*. Se debe tener en cuenta que en un proyecto ágil las pruebas de *software* se realizan en cada iteración.



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 agiles, 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.

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

Según los principios del *Agile Testing*, las personas que representan la parte de negocio del producto están involucradas en cada iteración del mismo, y el flujo de retroalimentación continua acorta el tiempo de respuesta y la toma de decisiones para integrar en la mejora del producto, estos principios son:

|  |
| --- |
| CF11\_2\_7\_3\_Principios |

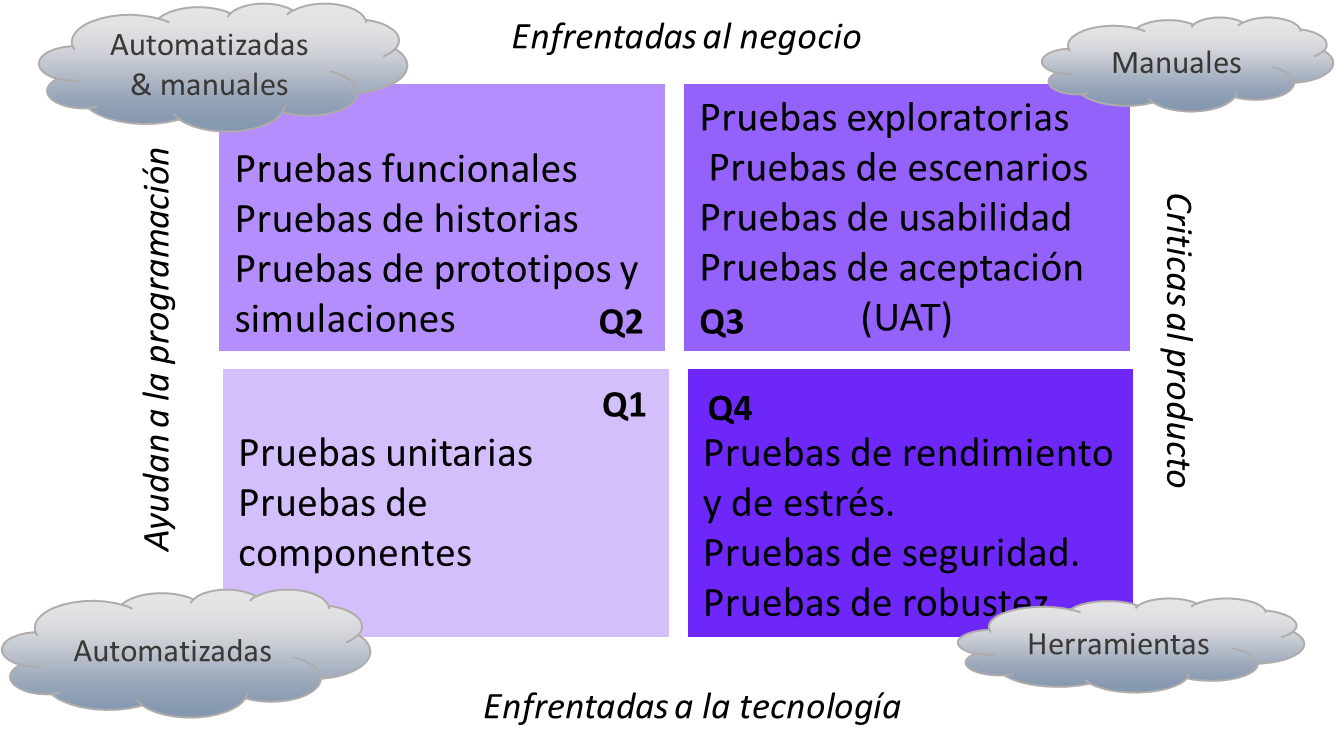
**Cuadrantes de las pruebas en metodologías ágiles**

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

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

**Figura 4**

*Cuadrantes de las pruebas ágiles*



Nota. Tomado de <https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/>

Cada cuadrante involucra pruebas específicas, las cuales se distribuyen de acuerdo con los roles ejercidos, es así como se tienen dos tipos de pruebas:

* + - * 1. **Pruebas de críticas al producto *(Critique the Product)***

Estas pruebas se enfocan en apoyar al equipo desarrollador a través de la construcción (desarrollo) del producto, puesto que le orientan sobre su funcionalidad, posteriormente contribuyen a la introducción de un nuevo código y la refactorización sin que genere resultados imprevistos en el comportamiento del sistema, se encuentran en:

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

* Pruebas unitarias.
* Pruebas de componente

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

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

Las pruebas y las funcionalidades de este cuadrante detallan cada historia de usuario, además también pueden ser automatizadas, no obstante, es necesario que algunas de estas pruebas sean ejecutadas desde la interfaz de usuario o lo que es lo mismo desde el punto de vista del cliente.

En las pruebas de este cuadrante, los diseñadores de la interfaz de usuario crean *wireframes* para realizar en primera instancia, validaciones con el cliente, y posteriormente iniciar con el desarrollo.

* + - * 1. **Pruebas de apoyo al equipo (*Supporting the Team)***

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

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

● Pruebas exploratorias.

● Pruebas de usabilidad.

● Pruebas de escenarios.

● Pruebas de aceptación de usuario.

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

● Pruebas de rendimiento y de estrés.

● Pruebas de seguridad.

● Pruebas de robustez.

**2.7.4 Metodología XP - Programación Extrema**

**XP** es la abreviación comúnmente utilizada para referirse a *Extreme Programming,* el cual es un marco de desarrollo de *software* ágil que busca producir *software* de alta calidad en contextos con requisitos altamente cambiantes, riesgos que involucran tiempos fijos con tecnologías nuevas y equipos de trabajo pequeños ubicados en un mismo sitio.

**XP** define cinco valores (Beck & Andres, 2004), tal como podemos ver a continuación:

|  |
| --- |
| CF11\_2\_7\_4\_Metodologia\_XP |

Adicional a los valores, XP se caracteriza por la definición de un conjunto de 12 prácticas de desarrollo de *software* que, aunque pueden ser adoptadas de forma aislada, tiene mayor relevancia cuando son desarrolladas en conjunto (Jeffries, 2011):

|  |
| --- |
| CF11\_2\_7\_4\_Practicas\_XP |

**3. Documentar el proceso de calidad de *software***

La documentación de productos de *software* son artefactos importantes, dado que estos permiten transferir y comunicar aspectos que al revisar o inspeccionar un sistema no se pueden entender rápidamente y suele ser complicado, por tal razón es que se documenta para comprender, compartir y mostrar el comportamiento y la estructura de un sistema o de sus componentes, controlar y visualizar las arquitectura del sistema y controlar el riesgo, además la documentación debe utilizarse para el desarrollo del producto *software* y su mantenimiento en el futuro resaltando que no solo se documenta un proceso respecto a la calidad, sino, también las fases del desarrollo del *software,* requerimientos, análisis, diseño, construcción y pruebas. Sin embargo, cuando se requiere documentar un proceso de calidad de *software* es pertinente tener en cuenta tanto el plan de pruebas como los resultados arrojados que se muestran en los apartados siguientes.

**3.1 Diseñar los instrumentos de calidad de *software***

Los instrumentos de calidad de *software,* al ser combinados con otras partes, son útiles al momento de realizar mediciones, observar y almacenar datos que sean verificables y que se puedan utilizar para mejoras continuas en los procesos de calidad de un producto. Entonces para documentar pruebas de *software,* es importante ver el plan de pruebas como un instrumento clave en el diseño y creación de pruebas de *software.*

**Plan de pruebas**

Un instrumento de gran utilidad para estructurar el proceso de pruebas y la documentación es el plan de pruebas. Por lo tanto, la finalidad es suministrar la información requerida para planear y controlar las actividades relacionadas con las pruebas en el proceso de desarrollo de un producto *software*. Presenta el enfoque para realizar la verificación de los componentes del producto. Todo proceso de desarrollo de *software* debe contar con la implementación de la fase de pruebas.



El personal que se dedica a ejecutar pruebas de *software* requiere de un plan de pruebas de *software*, cuyo objetivo es comunicar a todos los involucrados del proyecto: los entregables, ítems a ser certificados, criterios de aprobación y fallos, criterios de suspensión y reanudación, las necesidades de ambiente, las capacitaciones necesarias para los integrales del equipo, riesgos y el laboratorio de usabilidad.

El plan se puede aplicar a todo el proyecto y se ajusta a las necesidades de cada empresa, teniendo en cuenta el tamaño del proyecto, el tiempo, el costo, el ciclo de vida del *software*, los involucrados.

Cada entidad puede definir su propio plan de pruebas, basados en buenas prácticas, de acuerdo con las siguientes características:

|  |
| --- |
| CF11\_3\_1\_Plan\_de\_pruebas |

**Formatos**

Recordemos que el estándar ISO/IEC/IEEE 29119-3:2013 es un instrumento que suministra una serie de artefactos validados a nivel internacional por las organizaciones a la vanguardia del proceso de desarrollo de *software*, y a su vez al margen de la ejecución de pruebas de *software*. Los de mayor reconocimiento son:

* Documentación de proceso de pruebas (a escala) organizacional.
* Documentación del proceso de gestión de pruebas.
* Documentación del proceso de pruebas dinámicas.

**Artefactos de prueba**

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.

La calidad de *softwar*e se compone de artefactos específicos que se forman en las diversas etapas del proceso de SQA, como se observa a continuación.

**Plan de pruebas:** documento que describe un enfoque sistemático para probar un sistema.

**Informe de resultados de pruebas:** organiza y presenta un análisis resumido de los resultados de las pruebas para su auditoría y evaluación. Este informe debe incluir un listado de defectos detectados, la cantidad y tipos de pruebas elaboradas, así como su complejidad y los tipos de pruebas realizadas.

PLANIFICACIÓN

**Caso de prueba:** conjunto de condiciones con las cuales un *tester* debe determinar si un sistema funciona según lo requerido.

***Script* de prueba:** recopilación de instrucciones que se realizan al *software* sujeto a prueba para verificar que el sistema funciona de acuerdo con lo requerido. Hay *scripts* de pruebas automatizadas y manuales.

CONSTRUCCIÓN

**Resultado de la ejecución de pruebas:** recopilación de la información relacionada con los resultados de la ejecución de cada *script* de prueba, determinando si tuvieron una salida exitosa o un problema, respecto al comportamiento que debería tener.

**Defectos:** un error produce un defecto y un defecto produce un fallo.

EJECUCIÓN

**3.2 Aplicar los instrumentos de calidad de *software***

En la elaboración de un plan de pruebas se deben tener ¡claros los requerimientos de usuario que forman la iteración o proyecto, por lo tanto, se debe analizar la información de la especificación de requisitos, la matriz de trazabilidad, especificaciones y diseño funcional, casos de uso, requisitos no funcionales, prototipado e historias de usuario en el caso de metodologías ágiles.

**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 como podemos ver a continuación:

1. Debe contener el nombre del sistema a probar, el nombre del documento y la versión.
2. Especificar en qué consiste el sistema y cuál sistema se va a probar. También se incluye y describe el alcance de las pruebas.
3. En esta parte se listan los tipos de prueba a realizar y los requerimientos a probar en el proceso. Se específica y describe las características a probar y las que no se van a probar.
4. Representa el enfoque recomendado para la comprobación de las aplicaciones.
5. En esta parte se especifican los criterios de entrada y salida de la aplicación al área de Aseguramiento de Calidad de *Software* (SQA).
6. Se describen las actividades (plan, diseño, implementación y ejecución de pruebas, así como la evaluación de los resultados) que realizarán el equipo de SQA.
7. Se describe el informe de resultados de prueba y los defectos detectados durante el esfuerzo de pruebas.
8. Describe los recursos del sistema.
9. Describe las características del personal requerido para el esfuerzo de pruebas, así como las necesidades de entrenamiento.
10. En este punto se plantean las fechas estimadas de inicio y fin de las principales actividades de prueba.
11. En este punto se plantean los riesgos identificados al momento de la planificación y su estrategia de mitigación.

**Principales elementos para redactar un caso de prueba**

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, en la siguiente tabla, se describen los elementos principales que debe contener un caso de prueba.

**Tabla 5**

*Principales elementos de una casos de prueba*

|  |  |
| --- | --- |
| Elemento | Descripción |
| Identificador | Puede ser alfanumérico o numérico. |
| Nombre | Nombre del caso de prueba de manera concisa. |
| Descripción | Objetivo del caso de prueba, también describe qué probará; en ciertas ocasiones se incluye el ambiente de pruebas. |
| 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 prueba. |

**Principales elementos para redactar un *script* de prueba**

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

|  |
| --- |
| CF11\_3\_2\_Elementos\_script\_prueba |

**Creación del plan de pruebas para proyectos ágiles (*Agile Testing*)**

Un *release* define funcionalidades que ya están disponibles para el cliente. Por lo tanto, una planificación de un *release* tiene ciertos elementos:

|  |
| --- |
| CF11\_3\_2\_Agile\_Testing |

Estos elementos 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 documente los factores más importantes relacionados con las pruebas en cada *release*.

Para realizar pruebas de *software* en una metodología ágil, comenzamos por redactar el plan de pruebas, teniendo en cuenta qué 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.

**3.3 Análisis y entrega de resultados**

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

|  |
| --- |
| CF11\_3\_3\_Analisis\_entrega |

**Incidencias detectadas**

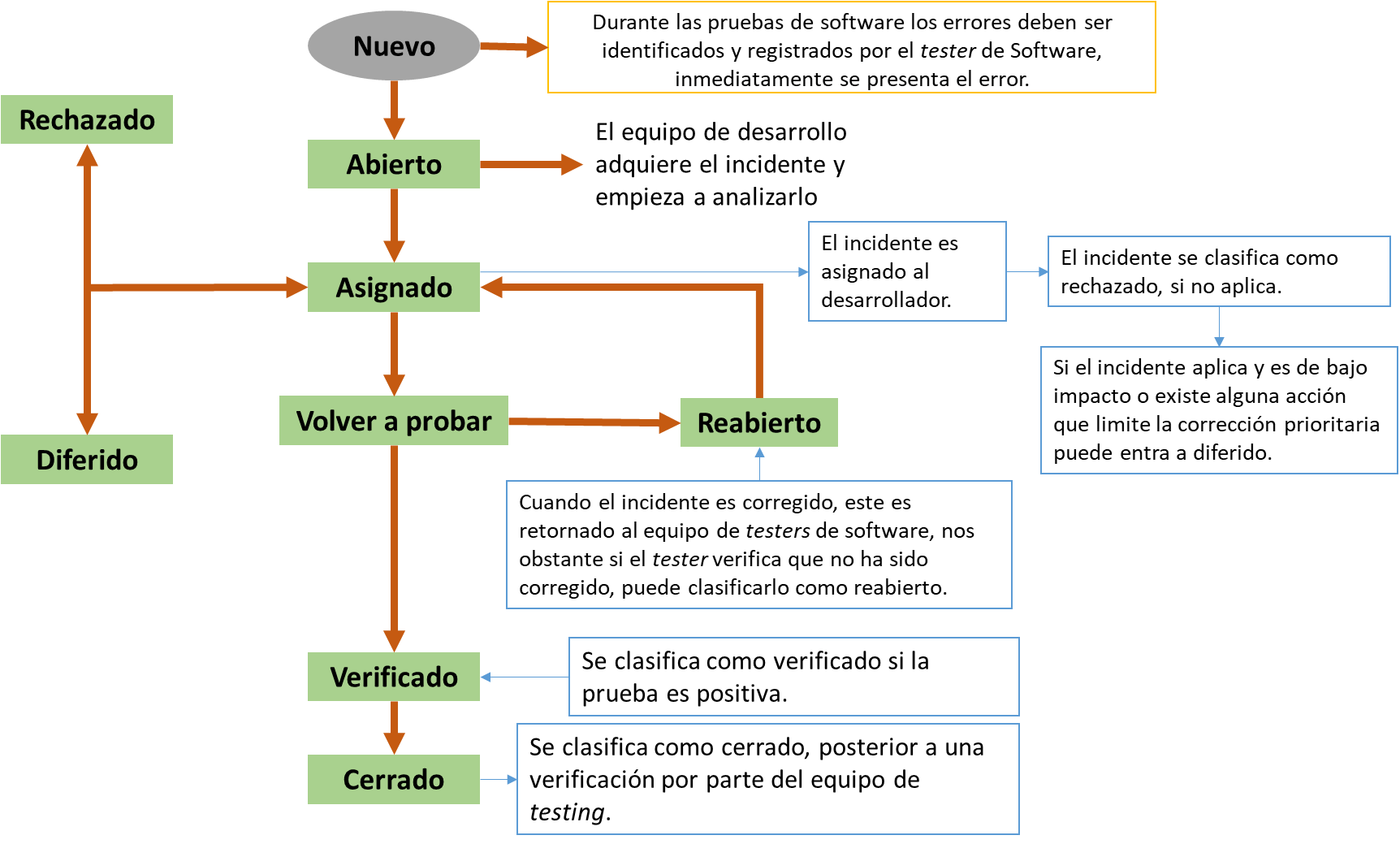
La gestión de incidencias es un elemento central e importante en el proceso de calidad de *software,* puesto que es en este punto donde se tienen en cuenta los errores (*bugs*). Por lo tanto, el propósito de las incidencias detectadas es precisamente generar su corrección, de tal manera que sea mínimamente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

**Ciclo de vida de una incidencia**

Un sistema de gestión de incidentes debe manejar los errores de *software* teniendo en cuenta el proceso. Este se adapta a proyectos de *software* en fase de desarrollo, más no en ambiente de producción.

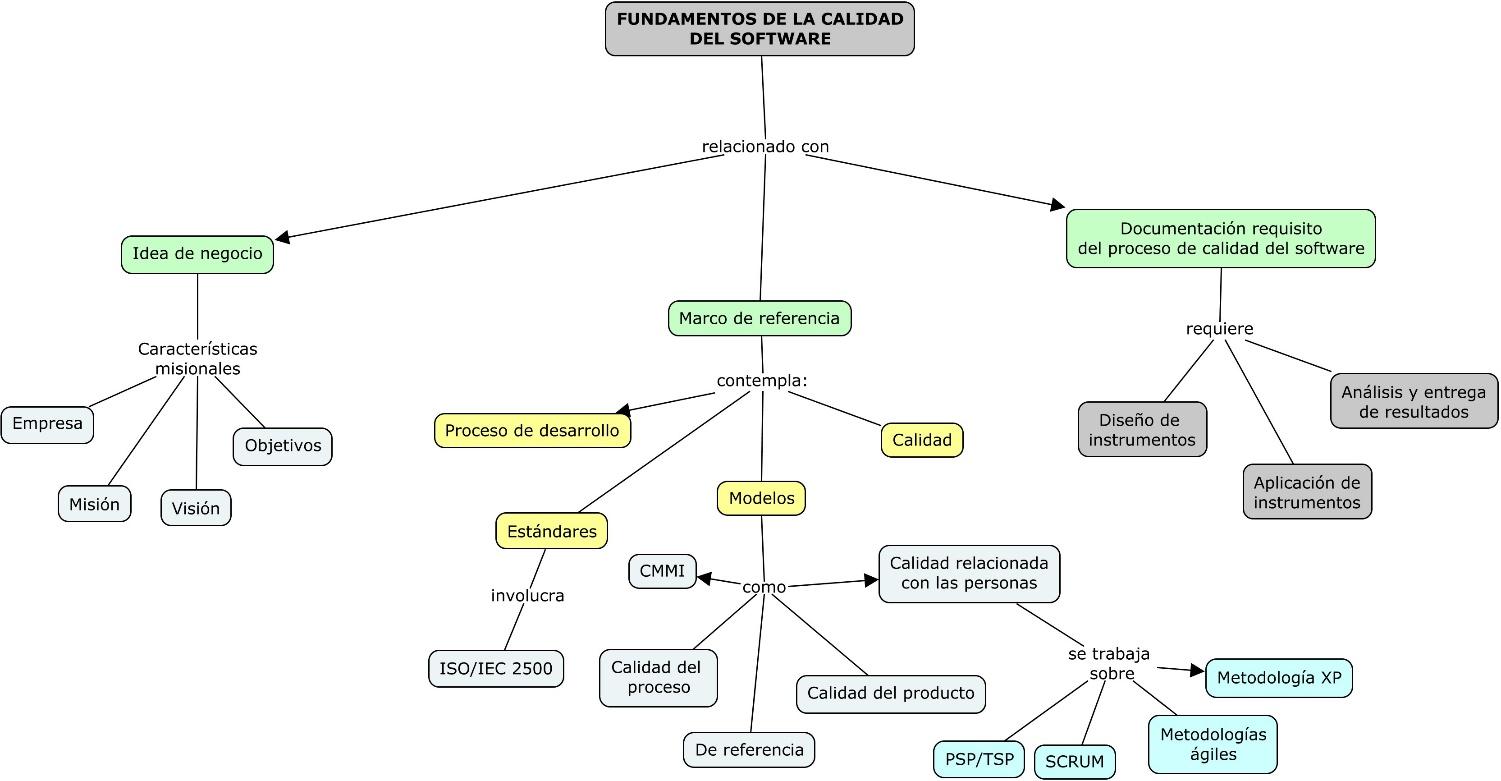
**Figura 5**

*Ciclo de vida de una incidencia de software (bug)*

****

1. **SÍNTESIS**

En el siguiente mapa conceptual podrá hacer un breve recorrido sobre las temáticas abordadas.



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

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la actividad | Fundamentos de calidad de *software* |
| Objetivo de la actividad | Afianzar los conceptos más importantes asociados a la calidad del *software*, modelos y metodologías de desarrollo y plan de pruebas. |
| Tipo de actividad sugerida | Preguntas de selección múltiple con única respuesta. |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | CF11\_Actividad\_didactica.docx |

1. **MATERIAL COMPLEMENTARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del recurso o  archivo del documento o material |
| PSP/TSP | Callejas-Cuervo, M. & Alarcón-Aldana, A. C. (2017). Modelos de calidad del *software*, un estado del arte. *Entramado, 13*(1), pp. 236–250. | Artículo | <https://revistas.unilibre.edu.co/index.php/entramado/article/view/428> |
| Modelo CMMI | Chavarría, A., Bayona Oré, S. & Pastor, C. (2016). Aseguramiento de la Calidad en el Proceso de Desarrollo de Software utilizando CMMI, TSP y PSp. *Risti, 20*(12), pp.62-77. | Artículo | <https://scielo.pt/pdf/rist/n20/n20a06.pdf> |
| Diseñar los instrumentos de calidad de *software* | Ch Ga, F. (2017). *Plan de pruebas de software.* Mundo Testing. | Página web | <https://mundotesting.com/plan-de-pruebas-de-software/> |
| Calidad de *software* en metodologías ágiles | Canós, J. H., Letelier, P., & Penad, C. (s.f.). *Metodologías Ágiles en el Desarrollo de Software.* | Artículo | <http://aleteya.cs.buap.mx/~jlavalle/papers/agileMethodology/TodoAgil.pdf> |

1. **GLOSARIO**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| Incidencia: | suceso que se produce durante una actividad y puede causar, una disminución de calidad de este. |
| PSP: | *Personal Software Personal.* |
| QA: | Calidad de *software.* |
| SQA: | Aseguramiento de la calidad de *software.* |
| TSP: | *Teams Software Personal.* |

1. **REFERENCIAS BIBLIOGRÁFICAS**

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

Bustamante Ramírez, J. (2011). Sistema de informes para pruebas de software. Obtenido de <http://bibliotecadigital.iue.edu.co/xmlui/handle/20.500.12717/153>

Clemente, P. J., & Gómez, A. (2014). Aplicación de un proceso de mejora continua en una. Obtenido de <http://hdl.handle.net/2099/15497>

Jeffries, R. (2011). *What is Extreme Programming?* Ronjeffries.Com. <https://ronjeffries.com/xprog/what-is-extreme-programming/>

Kruchten, P. (2003). The Rational Unified Process: An Introduction (3rd Edition) (3rd ed.). Addison-Wesley Professional.

Maida, EG, Pacienzia, J. (2015). Metodologías de desarrollo de software [en línea]. Tesis de Licenciatura en Sistemas y Computación. Facultad de Química e Ingeniería “Fray Rogelio Bacon”. Universidad Católica Argentina, 2015. Disponible en: <https://bit.ly/3hJMwXP>

Manifiesto por el Desarrollo Ágil de Software. (2001). Agilemanifesto.Org. <https://agilemanifesto.org/iso/es/manifesto.html>

Martin, J. (1991). *Rapid* Application Development. Macmillan Coll Div.

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

Royce, W.W. (1970) Managing the Development of Large Software Systems. Proceedings of IEEE WESCON, 26, 328-388.

SCRUMstudy. (2013). A Guide to the Scrum Body of Knowledge (*SBOK Guide)* (2013th ed.). VMEdu Inc.

Sommerville, I., Galipienso, M. I. A., & Martinez, A. B. (2005). *Ingenieria del Software*. Pearson Educación.

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Ervin Andrade | Experto temático | SENA Regional Cauca CTPI | Octubre /2021 |
| Peter Pinchao | Experto temático | SENA Regional Cauca CTPI | Octubre /2021 |
| María Inés Machado López | Diseñadora Instruccional | Regional Distrito Capital. Centro de Diseño y Metrología | Octubre de 2021 |
| Carolina Coca Salazar | Revisora Metodológica y Pedagógica | Regional Distrito Capital. Centro de Diseño y Metrología | Noviembre de 2021 |
| Sandra Patricia Hoyos Sepúlveda | Revisión y corrección de estilo | Centro para la Industria de la Comunicación Gráfica - Distrito capital | Noviembre de 2021 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la unidad temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del cambio |
| Autor (es) | Ana Catalina Córdoba Sus | Evaluadora instruccional | Regional Antioquia - Centro de Servicios de Salud | Febrero 2024 | Actualización |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Regional Antioquia - Centro de Servicios de Salud | Febrero 2024 | Actualización |