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”.

Marzo 2024

Tabla 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:

Estructura ideológica. Contiene el nombre de la empresa, visión, misión, valores, objetivos y un detalle de las ventajas competitivas del negocio.

Estructura del entorno. Es el arte y la ciencia, contiene un estudio de las fortalezas y las debilidades de la empresa, las tendencias del mercado, clientes potenciales y el comportamiento del sector donde se va a desarrollar.

- **Estructura mecánica**. Se describen las actividades que se deben ejecutar para alcanzar el éxito de la idea de negocio, tales como las ventas, las estrategias de distribución y la publicidad.
- **Estructura financiera**. Se verifica en términos económicos y proyecciones de escenarios la viabilidad de la idea de negocio, con el fin de observar si la utilidad es factible.
- **Recursos humanos**. Determina los puestos de trabajo requeridos, así como las obligaciones y derechos de cada miembro que integra la organización, independientemente que el emprendedor sea una sola persona.

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".

Análisis

Todo proyecto de "software" inicia con una fase de análisis. En el documento de especificación del software se determina la viabilidad y la definición de requisitos funcionales y no funcionales.

En la viabilidad se evalúa la rentabilidad, factibilidad y los costos de la construcción, además se define un pliego de condiciones, un plan y una estimación financiera y también la propuesta para el cliente, de ser necesario.

Posteriormente, se realiza la definición detallada de los requisitos, que debe incluir, el análisis de la situación de salida y un concepto. Los análisis de salida definen la problemática, y el concepto define qué funciones y características debe ofrecer el producto software para satisfacer las necesidades del cliente.

Diseño

En esta fase se formula una solución específica para la problemática definida o establecida en los requisitos, aquí los desarrolladores de "software" se encargarán de construir un diseño de la arquitectura del sistema, así como un plan de diseño, enfocados en los componentes, interfaces, entornos de trabajos y bibliotecas. Esta fase

da como resultado un estimado del plan de diseño y los planes de prueba para los diferentes componentes.

Implementación

La arquitectura otorgada en la fase de diseño se desarrolla en la fase de implementación, en la que se construye el código fuente que representa dicha arquitectura en un lenguaje de programación, también incluye la búsqueda de errores y pruebas unitarias.

Esta fase genera como resultado un producto "software" que se despliega por primera vez como insumo de la siguiente fase.

Verificación

Esta fase incluye el proceso de integración y despliegue del producto "software" en el entorno seleccionado, por lo general se prueba y envía con una serie de usuarios finales seleccionados. Se aplican las pruebas de aceptación incluidas en la fase de análisis y con ello se puede determinar si el "software" cumple o no con las necesidades definidas.

Los productos que superan estas pruebas, son aquellos que pueden continuar hacia el proceso de liberación hacia el público en general o su "target" específico.

Mantenimiento

Después que la fase de prueba haya finalizado con éxito, se autoriza para que la aplicación pase a un ambiente productivo, en esta última etapa se incluye la entrega, mantenimiento, garantía y mejoras del producto "software". Cabe destacar que la fase de mantenimiento incluye tres elementos:

- Mantenimiento correctivo: eliminar los defectos detectados durante su vida útil.
- Mantenimiento adaptativo: adaptar el "software" a nuevas necesidades.
- Mantenimiento perfectivo: añadir al "software" nuevas funcionalidades.

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

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.

División de Gestión de Calidad - ISO/IEC 2500n

Las normas en este punto, definen todos los modelos, términos y definiciones comunes referenciados por todas las otras normas de la familia 25000. Actualmente esta división se encuentra formada por:

- ISO/IEC 2500 – "Guide to SQuaRE": está compuesta por un modelo de la arquitectura de SQuaRE, la terminología de la familia, un resumen de las partes, los usuarios previstos, las partes asociadas y los modelos de referencia.
- ISO/IEC 2501 – "Planning and Management": determina los requerimientos y disposiciones para gestionar la especificación y evaluación de los requerimientos del producto "software".

División de Modelo de Calidad - ISO/IEC 2501n

Las normas de esta división presentan modelos detallados de calidad, así como las características para calidad externa, interna y en uso del producto software. Actualmente esta división está conformada por:

- ISO/IEC 25010 – "System and software quality models": define el modelo de calidad para el producto "software".

- ISO/IEC 25012 – "Data Quality model": describe un modelo general para la calidad de los datos, específicamente a los datos que se encuentran almacenados de forma estructurada y hacen parte de un sistema de información.

División de Medición de Calidad - ISO/IEC 2502n

Involucran un modelo de referencia de la medición de la calidad del producto, descripciones de medidas de calidad (externa, interna y en uso) así como guías con prácticas para su respectiva aplicación. Actualmente esta división está conformada por:

- ISO/IEC 25020 – "Measurement reference model and guide": representa una explicación introductoria y un modelo de referencia común a los elementos de medición de la calidad.
- ISO/IEC 25021 – "Quality measure elements": especifica un conjunto de métricas base que podrían ser utilizadas en el transcurso de todo el ciclo de vida del desarrollo "software".
- ISO/IEC 25022 – "Measurement of quality in use": especifica métricas para efectuar la medición de la calidad en uso del producto.
- ISO/IEC 25023 – "Measurement of system and software product quality:" especifica métricas para efectuar la medición de la calidad de productos y sistemas "software".
- ISO/IEC 25024 – "Measurement of data quality": especifica métricas para efectuar la medición de la calidad de datos.

División de Requisitos de Calidad - ISO/IEC 2503n

Las normas de esta división especifican requerimientos de calidad que podrían ser usados en el proceso de elicitación de requisitos de calidad del producto software a desarrollar. Actualmente esta división está conformada por:

- ISO/IEC 25030 – "Quality requirements": contiene una serie de recomendaciones para efectuar la especificación de los requisitos de calidad del producto "software".

División de Evaluación de Calidad - ISO/IEC 2504n

Esta división involucra normas que aportan guías, recomendaciones y requerimientos para proceder a realizar el proceso de evaluación del producto software. Actualmente esta división está conformada por:

- ISO/IEC 25040 – "Evaluation reference model and guide": plantea un modelo de referencia general para la evaluación, por lo tanto, tiene en cuenta las entradas en el proceso de evaluación, las restricciones y los recursos requeridos para alcanzar las adecuadas salidas.
- ISO/IEC 25041 – "Evaluation guide for developers, acquirers and independent evaluators": detalla las recomendaciones y requerimientos para la implementación práctica de la evaluación del producto software según las percepciones de los desarrolladores, de los que adquieren el software y los respectivos evaluadores independientes.
- ISO/IEC 25042 – "Evaluation modules": define lo que la Norma considera un módulo de evaluación y la documentación, estructura y contenido que se debe utilizar a la hora de definir uno de estos módulos.

- ISO/IEC 25045 – "Evaluation module for recoverability": describe un módulo para la evaluación de la recuperabilidad.

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 los conjuntos predefinidos en cada área de proceso. Veamos su distribución:

Nivel 1 - Madurez inicial

Los procesos suelen ser ad hoc y 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.

Nivel 2 - Madurez administrativo

Los objetivos generales y específicos han sido alcanzados, por decirlo de alguna manera, los proyectos han asegurado que los requisitos son gestionados, los procesos son planificados, y ejecutados de manera controlada. Se garantiza que los procesos se sigan ejecutando bajo presión y alto nivel de estrés, conforme a los planes documentados, y los productos o servicios satisfacen las necesidades del cliente en los requisitos especificados, también los objetivos y normas.

Nivel 3 - Madurez definida

La organización ha logrado alcanzar todos los objetivos específicos de las áreas asignadas en los niveles 2 y 3. En este nivel los procesos están bien caracterizados, entendidos y donde se definen normas, procedimientos, herramientas y métodos para su aplicación. Entre el nivel 2 y el 3 hay una diferencia bien marcada desde los estándares, las descripciones de los procesos y los procedimientos dependiendo de las instancias, -por decirlo así- por proyecto en específico. Por decirlo de otra manera en el nivel 3, los procesos de la organización se adaptan a la ejecución de un proyecto en específico permitiendo de esta manera adaptarse fácilmente a nuevos retos.

Nivel 4 - Madurez administrativo cuantitativamente

Se ha logrado con éxito obtener los objetivos específicos de los niveles 2, 3 y 4 y los objetivos genéricos en los niveles 2 y 3. 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. Estas métricas posibilitan identificar casos que generen problemas para ser corregidos y no se repitan en el futuro, también para tomar decisiones de negocio que proyectan el crecimiento de la organización.

Una diferencia importante entre el nivel 3 y 4, es que en el nivel 3 los procesos son medidos de forma cualitativamente predecibles, dando paso a la especulación, mientras que en el nivel 4 se evalúa el rendimiento real de los mismos procesos, y esto se determina por medio de técnicas estadísticas y otras técnicas cuantitativas lo previsible dentro de la organización.

Nivel 5 - Madurez optimización

En este nivel la organización ha logrado alcanzar todos los “goals of” definidos en los niveles 1, 2, 3, 4 y 5, y los objetivos genéricos en los niveles 2 y 3. Este nivel está centrado en optimizar y en la mejora continua del rendimiento de los procesos de la organización. Una de las diferencias entre el nivel 4 y el 5 es que, en el nivel 4, los procesos están diseñados para causas especiales y proporcionan métricas para ser evaluadas por técnicas estadísticas; mientras que en el nivel 5, los procesos están diseñados para que se encarguen de las causas comunes de la variación y el cambio de los procesos, conservando las estadísticas para poder prever acciones futuras, este ajuste permite llegar a los objetivos cuantitativos de mejora continua.

Estructurada por nivel de capacidad

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:

- **Nivel 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.
- **Nivel 1 – Se ha realizado**.
- **Nivel 2 – Administrativo**. Se denominan procesos administrados, porque se puede contar con una planeación, una trazabilidad y un control en los proyectos individuales o grupales.
- **Nivel 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.
- **Nivel 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.
- **Nivel 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”:

Calidad

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 de “software” (QA)

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.

Aseguramiento de la calidad de “software” (SQA)

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.

Propósito del 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.

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 este 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”, y a la administración, la seguridad de que procesos oficialmente establecidos están siendo implementados, teniendo en cuenta:

a) Establecer estándares, procesos y planes, con el objetivo de satisfacer las políticas de la organización y que se ajusten a las necesidades de cada proyecto en particular, para así asegurar la calidad del producto software.
b) Auditar y revisar los productos y actividades desarrolladas para verificar que satisfacen los procesos y estándares definidos.
c) Proporcionar las herramientas necesarias que den soporte al proceso definido, facilitando su ejecución, visualización, administración y seguimiento.
d) Proveer al equipo de proyecto y a otros interesados, los resultados sobre las revisiones, auditorías y actividades.
e) Determinar las funcionalidades esperadas por el cliente.
f) Escalar problemas presentados dentro del equipo de un proyecto, hacia un nivel apropiado de administración para su resolución.
g) Garantizar que se cumpla el plan generado para el desarrollo del proyecto y del proceso de desarrollo de software.

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:

- **Proceso**. La calidad debe ser una constante a lo largo de todo el ciclo de vida del “software”. Es una acción que se construye en la medida que el producto se va desarrollando, más 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.
- **Equipo**. La calidad del “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.
- **Planificación**. 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 y definido que incluya que, quien, y cuando, va a realizar la verificación o medición de la calidad.
- **Prevención**. Se deben dirigir los esfuerzos a la 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.

- **Sistemas de detección**. 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 detectamos 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".
- **Productividad**. La calidad al igual que la productividad, los 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", porque de qué sirve un producto sin defectos cuando ha costado mucho más de lo presupuestado y se ha entregado cuando ya no será útil.
- **Participación**. 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.

Componentes del aseguramiento de la calidad de software (SQA)

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

- **Procedimientos e instrucciones de trabajo**. Procesos claramente definidos e instructivos que ayuden al personal a realizar sus actividades con cierto nivel de estandarización.
- **Plantillas y listas de verificación**. Se crean plantillas base para estandarizar la documentación producida y también crear instrumentos que permitan medir la calidad de dichos documentos.
- **Capacitación del personal, la readaptación profesional y la certificación**. El personal debe estar capacitado para ejecutar las actividades que le corresponden a su rol, por eso la capacitación juega un papel muy importante dentro de SQA.
- **Acciones preventivas y correctivas**. Se deben realizar acciones para prevenir que un problema se presente y acciones para corregir los problemas que ya se presentaron, usualmente estas acciones preventivas y correctivas surgen a raíz de las verificaciones y de las auditorías.
- **Gestión de la configuración**. En la organización se deben de tener procesos que especifiquen cómo serán llevadas a cabo las actividades cotidianas como

son la recepción de los requerimientos y movimientos de activos de "software" entre las áreas, tales como el código fuente.

- **Control de la documentación**. Incluye controlar las versiones de los documentos y garantizar la disponibilidad y el acceso a los mismos, por todos los involucrados en el proceso.

Proceso del aseguramiento de la calidad de software (SQA)

En el proceso de aseguramiento de la calidad de software, se encuentran tres etapas:

- **Planificación**. Se planifica que es lo que se va a probar, es decir se define el alcance de las pruebas.
- **Construcción**. Se crean los artefactos de prueba y el script de pruebas.
- **Ejecución**. En esta etapa se encuentran los defectos y se ejecutan los scripts de pruebas creados previamente.

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

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.

Perspectivas	Factores	Criterios
	Fiabilidad – confiabilidad: capacidad de los sistemas de no fallar / la medida en que falla el sistema.	• Simplicidad. • Concreción.

Perspectivas	Factores	Criterios
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.

Perspectivas	Factores	Criterios
Transición del producto: factores de calidad que influyen en la capacidad de	Reusabilidad: facilidad de reutilización de “software” en un contexto diferente.	• Auto-descripción. • Generalidad. • Modularidad.

Perspectivas	Factores	Criterios
adaptar el “software” a los nuevos entornos.	Interoperabilidad: esfuerzo requerido para acoplar el sistema a otro sistema.	• Modularidad. • Similitud de comunicación. • Similitud de dato. • Independencia del sistema. • Independencia de la máquina.

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.

Factores	Criterios
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.

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.

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:

- **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 reutilizarse 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.
- **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.

- **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.

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.

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:

Fase 1 – Planeación

Se realiza el plan que guiará el proceso, por lo tanto, se establecen dos actividades:

- Análisis para obtener los requerimientos del software.
- Estimación de recursos, en la cual se realiza un cálculo del tiempo que se requiere para poder desarrollar el programa.

Fase 2 – Desarrollo

En esta fase se tienen en cuenta las siguientes actividades:

- Diseño: se hace revisión de los requerimientos para poder producir un diseño que verifique el cumplimiento de los mismos, por otro lado, si es el caso, se debe efectuar un registro de cualquier defecto encontrado en la definición de requerimientos.

- Codificación: se implementa el diseño siguiendo el estándar de codificación y se documenta el tiempo en el registro.

Fase 3 – Fase de "postmortem"

Se realiza un reporte que es utilizado para documentar detalles importantes de una actividad. Por ejemplo, que problemas surgieron y cómo se solucionaron.

En esta fase se tienen en cuenta las siguientes actividades:

- Registro de defectos: se hace revisión del resumen del plan de proyecto y se verifica que los defectos encontrados en cada fase de proyecto fueran registrados.
- Consistencia de información de defectos: se hace revisión de la información de cada defecto registrado el cual debe ser preciso y completado.
- Tiempo: se hace revisión de posibles errores u omisiones en el registro del tiempo.

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.

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

Formato del Registro de Tiempo

Estudiante _____ Fecha _____
Instructor _____ Programa # _____

Fecha	Inicio	Trmino	Tiempo de Interrupción	Tiempo Delta	Fase	Comentarios

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

Tipos de Defectos

10 Documentación	60 Chequeo
20 Sintáxis	70 Datos
30 Construcción, Empacar	80 Función
40 Assignación	90 Sistema
50 Interfaz	100 Ambiente

Formato del Registro de Defectos

Estudiante ____________ Fecha ____________

Instructor ____________ Programa # ____________

Fecha	Número	Tipo	Encontrado	Removido	Tiempo de compostura	Defecto arreglado

Descripción:

Fecha	Número	Tipo	Encontrado	Removido	Tiempo de compostura	Defecto arreglado

Descripción:

Fecha	Número	Tipo	Encontrado	Removido	Tiempo de compostura	Defecto arreglado

Descripción:

Fecha	Número	Tipo	Encontrado	Removido	Tiempo de compostura	Defecto arreglado

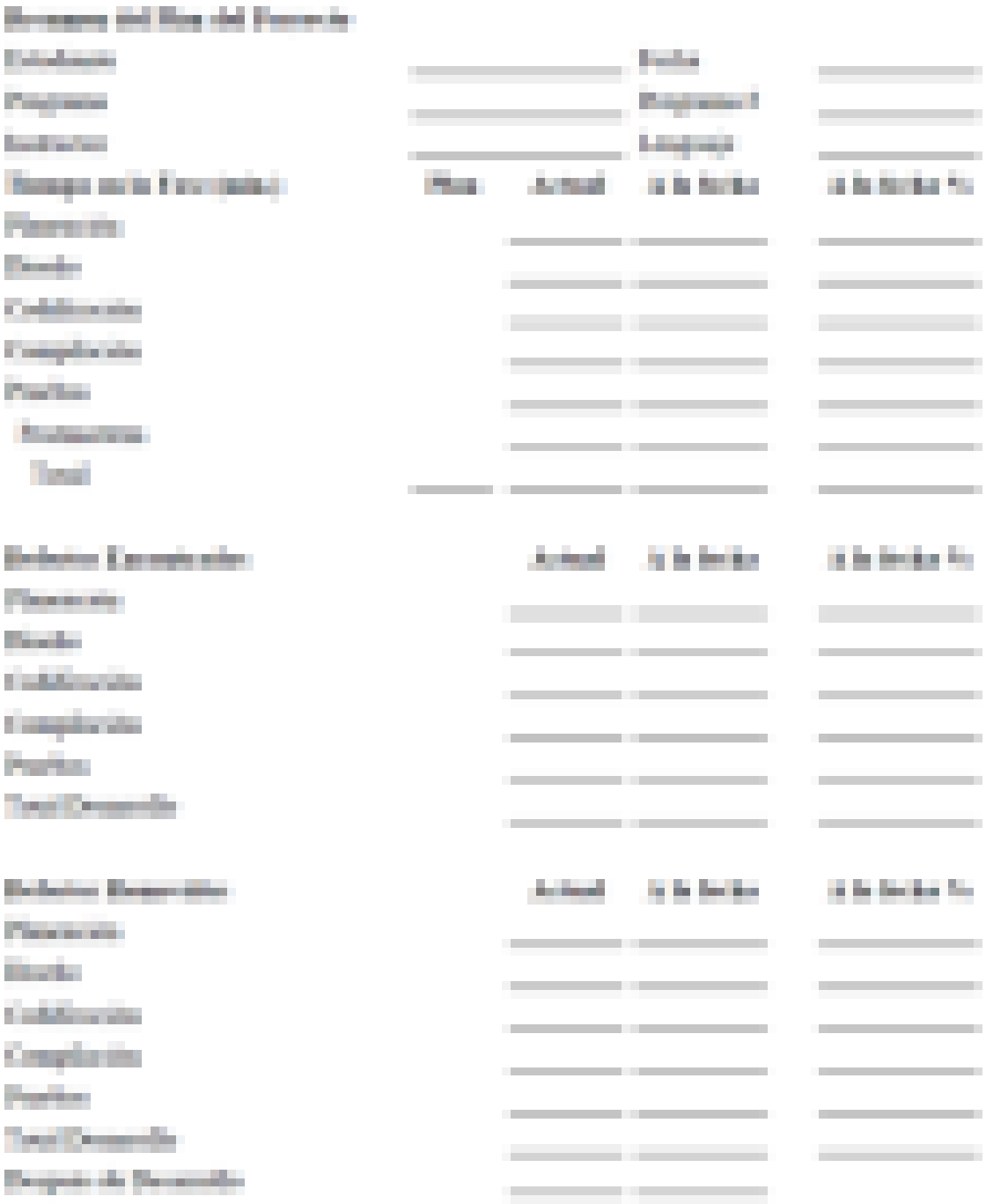
Descripción:

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 3. Ejemplo resumen del plan del proyecto



TSP

"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.

- **Lanzamiento**. En esta etapa se determinan las metas del equipo, se evalúan los objetivos, se establecen los roles y responsabilidades del equipo, se tienen en cuenta los requerimientos del cliente y la creación de estrategia para la finalización del proyecto.
- **Estrategia**. En esta etapa se crea un modelo conceptual de lo que se necesita para generar soluciones óptimas, determinando las estimaciones y los riesgos.
- **Planeación**. En esta etapa se generan los roles y las actividades de cada miembro del equipo.
- **Requerimientos**. En esta fase se determinan entrevistas con el cliente, con el objetivo de concretar lo que es necesario producir en realidad. Desarrollar un plan de pruebas para el producto terminado previamente al análisis de los requerimientos.
- **Diseño**. En esta etapa se desarrolla un plan de pruebas de integración, por lo tanto se especifica en un diseño, todos los detalles respecto a los procesos del producto.
- **Implementación**. En esta etapa el diseño se transfiere a nivel de código, realizando una revisión con el propósito de encontrar errores. También los módulos y las unidades se compilan y ejecutan con el propósito de analizar su calidad.
- **Pruebas**. En esta etapa el producto está a punto de finalizar, posteriormente se procede a la integración de los módulos y la documentación para los usuarios

finales, como son los manuales de uso. También se establecen las pruebas al software, con el objetivo de asegurar la calidad y poder así generar una evaluación en el desempeño del equipo de trabajo.

- **“Postmortem”**. En esta etapa se hace una evaluación de los resultados de las pruebas y del trabajo del equipo. Se redacta un reporte del ciclo de vida del proyecto a detalle.

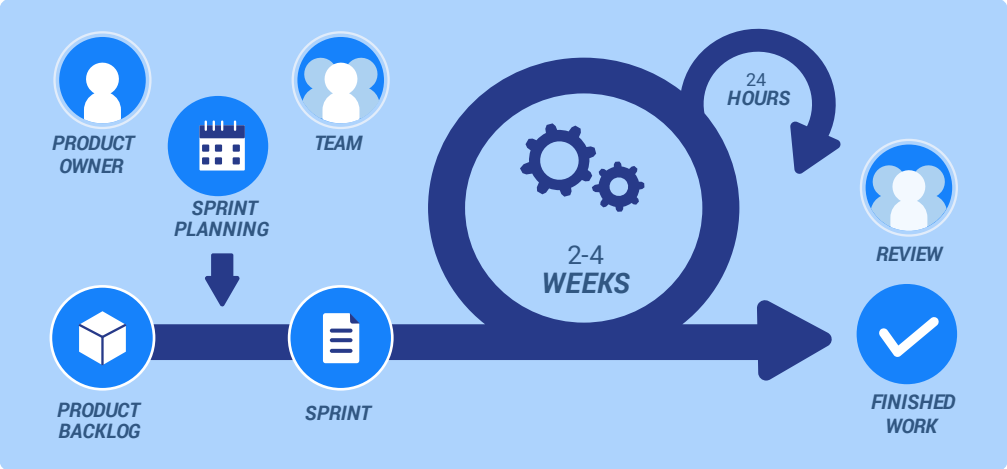
SCRUM

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):

- **Transparencia**. Hace referencia a que cualquier proceso de Scrum puede ser conocido por cualquiera. Esto es posible por medio de eventos como las reuniones de revisión y reuniones diarias, artefactos como la pila de producto, cronogramas de lanzamiento y documentos de visión del proyecto, e instrumentos de seguimiento como lo son el “BurnDown Chart” o el tablero de Scrum (“ScrumBoard”).
- **Inspección**. Permite que cualquiera puede estar enterado de las actividades realizadas por otros y en general conocer el estado actual de los procesos.
- **Adaptación**. Por medio de la transparencia y la inspección es posible fijar actividades de mejoras que permitan mejorar todo tipo de proceso, en pro de lograr más altos estándares de calidad.

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

Figura 4. Roles, eventos y artefactos en el trabajo ágil



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:

Dueño del producto ("Product Owner"). Persona con amplio conocimiento en el negocio del cliente, sus necesidades y las tendencias del mercado para el área específica. Este rol está encargado de maximizar el valor de negocio entregado al cliente y es el único responsable del control del "Product Backlog" (requerimientos)

y su priorización. Este rol también representa al cliente en algunos procesos de demostración de avances y determina cuándo aprobar o no una entrega.

Scrum Master. Es un rol que se encarga de facilitar los procesos al interior del equipo de trabajo, removiendo cualquier impedimento y apoyando procesos de empoderamiento personal, debe velar porque los elementos propios del marco de trabajo Scrum se apliquen de manera correcta.

Equipo de desarrollo ("Developer team"). Son los responsables de la transformación de los requerimientos en código ejecutable a ser usado por el cliente, pero también son responsables de la planificación de las iteraciones y establecimiento de características para tener en cuenta en la verificación de un requerimiento terminado y presentación de avances a los clientes. Generalmente es un equipo auto organizado y auto gestionado.

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:

Pila de producto (“Product Backlog”)

Lista priorizada de requerimientos generalmente descritos en formato de historias de usuarios, que representa todas las características del sistema a construir.

Pila del Sprint (“Sprint Backlog”)

Lista de requerimientos seleccionados desde el “Product backlog” por el equipo de trabajo, para ser desarrollados durante un “Sprint” particular. Este es un de los artefactos generados en la reunión de planeación del “Sprint”.

Burndown Chart.

Es un gráfico de dos ejes que muestra a los equipos la cantidad de trabajo pendiente por completar (eje Y) y el tiempo disponible para hacerlo (eje X).

Este gráfico generalmente se realiza por cada “Sprint”, ubicando la cantidad trabajo a realizar del “Sprint Backlog” (usualmente mide puntos de historia o horas de trabajo) en un tiempo 0 y por cada día finalizado se resta la cantidad de puntos de historia u horas de cada tarea completada, también es posible usar este mismo gráfico para representar el avance general del proyecto, ubicando en el eje Y la cantidad total de horas o esfuerzo del “Product Backlog” y en el eje X la cantidad de “Sprint” proyectados. Cada uno de estos puntos se unen por medio de una línea y es posible determinar visualmente si el flujo de trabajo está en una situación óptima o no respecto al tiempo restante para completar el “Sprint”.

Tablero de Scrum (“Scrumboard”)

Es un elemento visual en donde se integra la mayor parte de los elementos del marco de trabajo “Scrum”, en él se indica la carga de trabajo, el estado actual de cada una de las actividades y sus respectivos responsables. Este es un elemento que se

sincroniza de manera permanente y facilita la implementación los pilares de transparencia, inspección y adaptabilidad.

Si bien se aconseja el uso de un tablero existen diferentes tipos de herramientas digitales que permiten la implementación de un tablero de "Scrum".

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.

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 están definidas funcionalidades que pueden ser utilizadas por el cliente. En los proyectos ágiles existen dos tipos de planificaciones: "release"/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 "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:

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

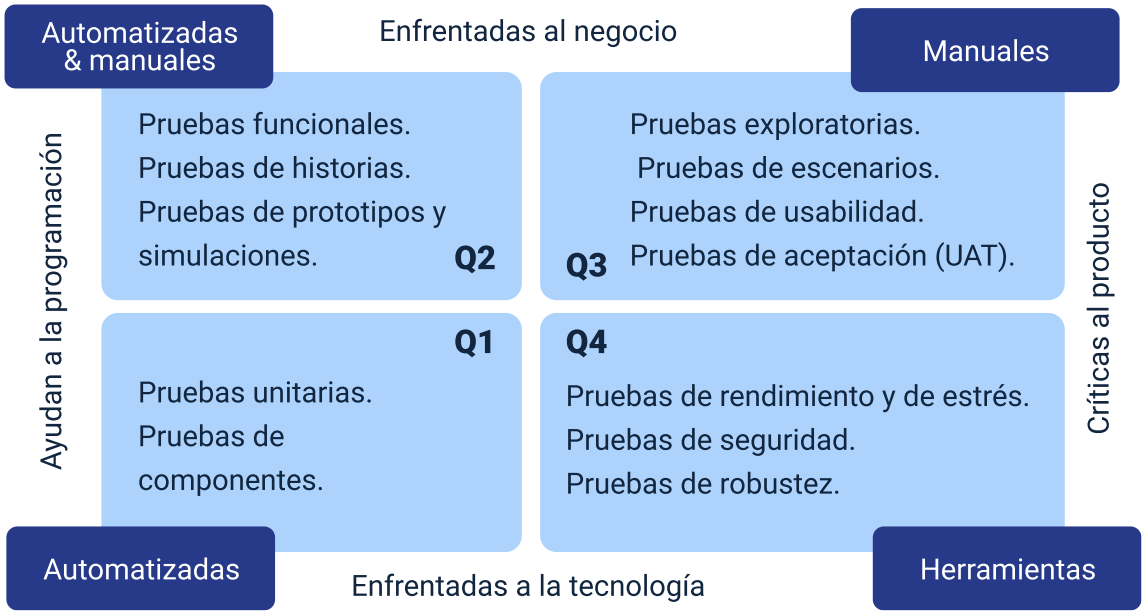
- **Entrega continua**. Se soporta en la integración continua, en este punto los desarrolladores realizan un control y la cantidad de veces que se realiza la entrega continua.
- **Elaborar menos documentación**. Los participantes de los equipos en metodologías ágiles pueden crear listas y tomar notas para centrarse en probar el "software", lo que permite no enfocarse en detalles secundarios que tomarían mucho tiempo.
- **Responder con velocidad a la retroalimentación**. Los involucrados en el negocio del producto están estrechamente conectados con las interacciones del producto, además el tiempo de respuesta se reduce debido a la realimentación continua en cada parte de la interacción.
- **Proporcionar retroalimentación constante**. En las pruebas agiles la retroalimentación es continua de modo que el diseño del producto logre los propósitos del negocio requeridos.

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 5. Ejemplo formato de registro de tiempo



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

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.

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:

- 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.

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.

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:

Comunicación. El desarrollo de "software" requiere de trabajo en equipo, por lo cual es importante la transferencia de conocimientos y utilizar medios de comunicación apropiados, se propone la discusión cara a cara con herramientas que permitan dibujar o escribir como por ejemplo, un tablero.

Simplicidad. Se refiere a hacer solo las cosas que sean absolutamente necesarias, evitando el desperdicio, elaborar las cosas de forma que sea fácil entender por otros y abordar solo requisitos conocidos.

Retroalimentación. Permite identificar áreas de mejora y revisión constante de las prácticas implementadas en el proceso de forma, que puedan establecer procesos de mejora permanente.

Coraje. Se refiere al actuar sobre situaciones que pueden ser retadoras para el equipo como por ejemplo, enfrentar problemas organizacionales, intentar implementar funcionalidades de formas diferente cuando lo convencional no funciona, aceptar comentarios, etc.

Respeto. Por medio de la transparencia y la inspección es posible fijar actividades de mejoras que permitan mejorar todo tipo de proceso, en pro de lograr más altos estándares de calidad.

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):

- **Planificación**. Consiste en una reunión de planeación que realiza el equipo de desarrollo en conjunto con el cliente, para discutir y probar las características a ser desarrolladas.
- **Pequeños lanzamientos**. Esta práctica sugiere hacer iteraciones cortas con funcionalidades pequeñas que puedan ser probadas con el cliente, de forma que se obtiene realimentación temprana y frecuente.
- **Metáfora**. Los nombres usados para definir cualquier tipo de identificador en el sistema debe ser coherente. El diseño y la estructura general debe ser comprensible para cualquier persona.
- **Diseño simple**. Hacer código que funcione y que sea a la vez la solución más simple posible, evitando duplicación de código, menor número de métodos y clases.
- **Pruebas**. Sugiere el uso de técnicas donde se enfatice en el proceso de pruebas constantes antes de iniciar la construcción del código, como por ejemplo TTD (Desarrollo basado en pruebas).
- **Refactorización**. Eliminación de funciones innecesarias, desacoplar elementos, eliminar cualquier tipo de redundancia de forma que se mantenga un código lo más limpio posible, que sea fácil de entender y modificar.
- **Programación en pareja**. Esta práctica propone que un código sea desarrollado por el trabajo de dos personas, donde una de ellas codifica y la otra está haciendo revisión, sugerencias y corrección de errores en el mismo momento de escritura.
- **Propiedad colectiva**. Todo el equipo es dueño del sistema, esto promueve la cooperación y la libertad de cada miembro para aportar nuevas ideas.

- **Integración continua**. Implica el uso de sistema de control de versiones de forma que el código pueda ser compartido o reutilizado, pero también cada persona pueda trabajar de forma independiente y en paralelo sobre el mismo proyecto.
- **Semana de 40 horas**. Es importante que los desarrolladores tengan espacios de trabajo apropiados, donde puedan descansar y mantener un equilibrio entre la vida laboral y personal favoreciendo la eficiencia y puedan sostener la calidad durante todo el tiempo. Propone que no se deben exceder las 45 horas de trabajo semanales.
- **Cliente in situ**. El cliente final debe participar durante todo el proceso de construcción de software y estar presente para resolver cualquier pregunta que se pueda generar.
- **Estándar de codificación**. Se deben utilizar los mismos formatos y estilos en la redacción de código.

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 la 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:

El plan de prueba. Denota todos los procedimientos y métodos que serán empleados para la certificación del "software" determinando, si cumple con las especificaciones presentadas en los requisitos y dadas por el cliente. Este incluye los objetivos de calidad, los recursos, cronograma, asignaciones, y métodos, entre otros.

Casos de prueba. Listar los ítems que serán certificados y describe detalladamente los pasos que serán ejecutados para evaluar la verificación de las funcionalidades del producto "software".

Reporte de pruebas. Describe los fallos encontrados en el proceso de ejecución de pruebas y las pruebas exitosas.

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 "software" se compone de artefactos específicos que se forman en las diversas etapas del proceso de SQA, como se observa a continuación.

Planificación

- **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.

Construcción

- **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.

Ejecución

- **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.

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:

a) Debe contener el nombre del sistema a probar, el nombre del documento y la versión.
b) Especificar en qué consiste el sistema y cuál sistema se va a probar. También se incluye y describe el alcance de las pruebas.
c) 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.
d) Representa el enfoque recomendado para la comprobación de las aplicaciones.
e) En esta parte se especifican los criterios de entrada y salida de la aplicación al área de Aseguramiento de Calidad de "Software" (SQA).
f) Se describen las actividades (plan, diseño, implementación y ejecución de pruebas, así como la evaluación de los resultados) que realizarán el equipo de SQA.
g) Se describe el informe de resultados de prueba y los defectos detectados durante el esfuerzo de pruebas.
h) Describe los recursos del sistema.
i) Describe las características del personal requerido para el esfuerzo de pruebas, así como las necesidades de entrenamiento.

j) En este punto se plantean las fechas estimadas de inicio y fin de las principales actividades de prueba.

k) 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

Factores	Criterios
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:

- **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**. 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.

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

Un "reléase" define funcionalidades que ya están disponibles para el cliente. Por lo tanto, una planificación de un "reléase" tiene ciertos 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**. Define qué se incluirá en la automatización.
- **Ambiente de pruebas**. Define qué se necesita en relación al ambiente de pruebas.
- **Datos de las pruebas - Resultados de las pruebas**. Se establecen los datos arrojados en cada una de las pruebas, junto con los resultados encontrados.

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:

- Introducción.
- Alcance.
- Recursos en este caso el nombre de los "tester".
- Descripción de funcionalidades a probar.
- Los tipos de pruebas (pruebas de rendimiento, pruebas de aceptación - UAT) que se van a realizar.
- Infraestructura lista.
- 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:

- **Impacto**. Analizar el impacto del defecto.
- **Problema**. Investigar el problema, en qué condiciones se produce este fallo.
- **Severidad**. Analizar la severidad, si es alta o baja.
- **Pasos**. Variar los pasos realizados, se pueden ejecutar en orden diferente para observar si existe alguna variante en el resultado esperado.
- **Configuración**. Variar opciones de configuración; si estamos probando una aplicación web podemos probar con diferentes navegadores.
- **Condiciones**. Determinar condiciones específicas bajo las cuales se reproduce el defecto. Si estamos probando entradas de datos, probar con diferentes datos, es decir, se va a tener más información sobre cómo se reproduce el defecto, si por ejemplo, en los campos numéricos se prueba números positivos y negativos.

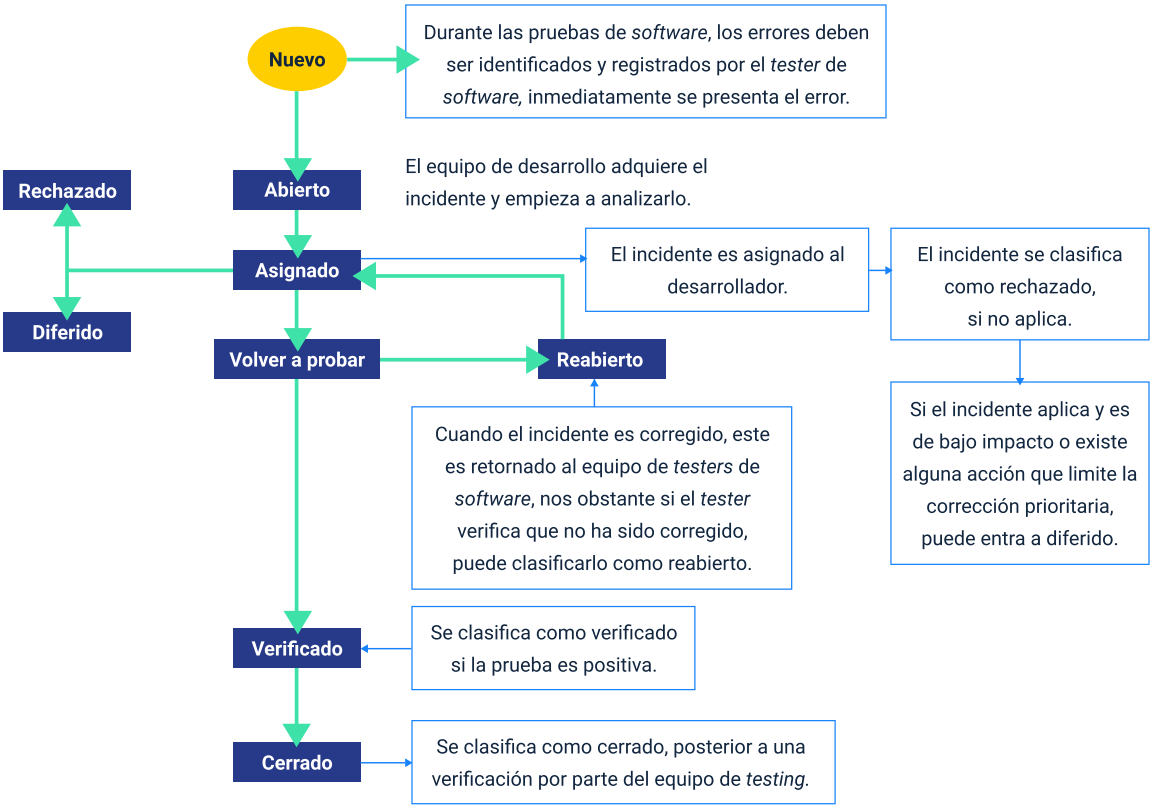
Incidencias detectadas

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.

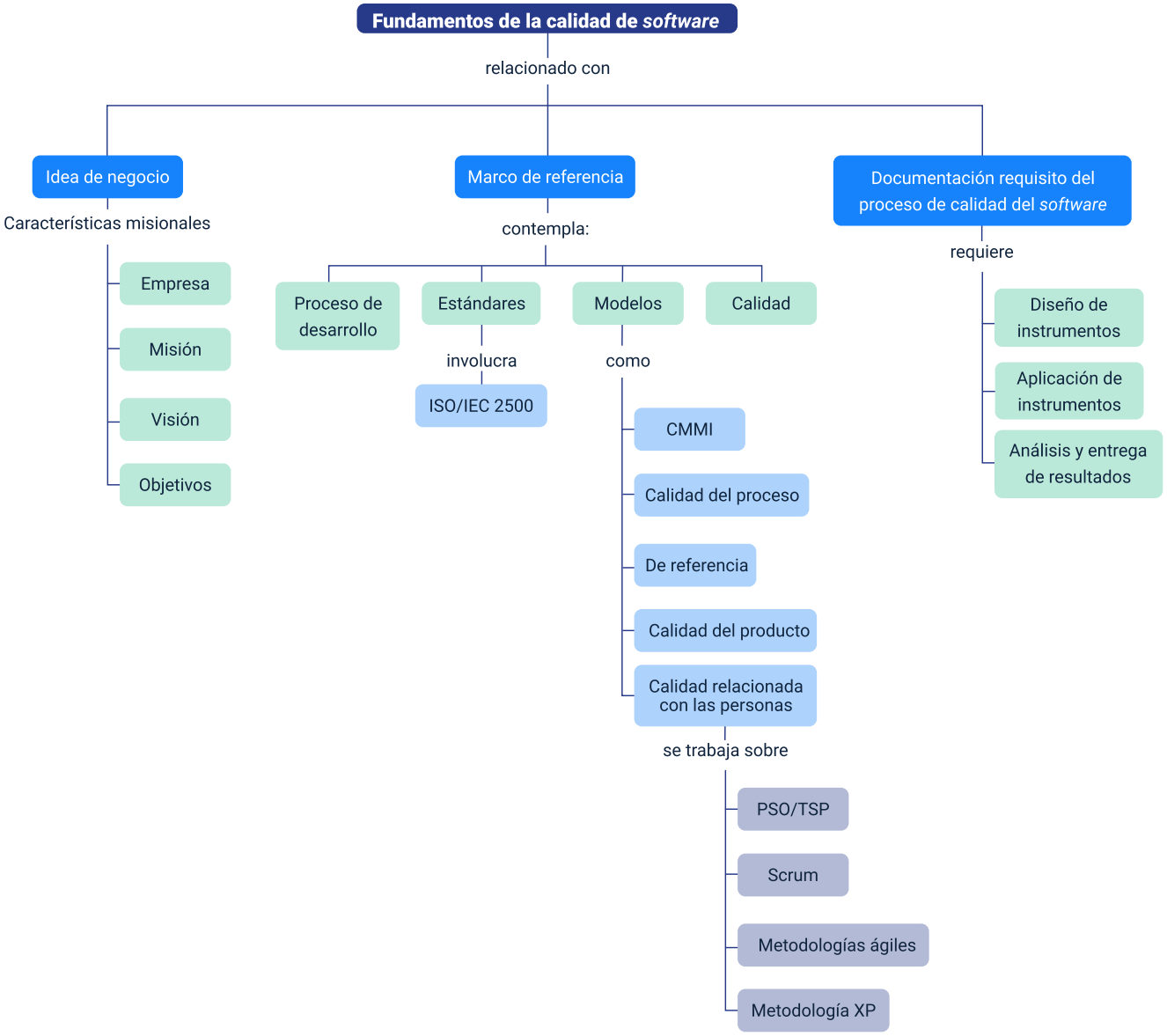
Ciclo de vida de una incidencia

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 6. Ciclo de vida de una incidencia de software (bug)



Síntesis


Fundamentos de la calidad de *software*
relacionado con
Idea de negocio
Características misionales
Empresa
Misión
Visión
Objetivos
Marco de referencia
contempla:
Proceso de desarrollo
Estándares
involucra
ISO/IEC 2500
Modelos
como
CMMI
Calidad del proceso
De referencia
Calidad del producto
Calidad relacionada con las personas
se trabaja sobre
PSO/TSP
Scrum
Metodologías ágiles
Metodología XP
Calidad
Documentación requisito del proceso de calidad del *software*
requiere
Diseño de instrumentos
Aplicación de instrumentos
Análisis y entrega de resultados

Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
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/
Diseñar los instrumentos de calidad de “software”	SO/IEC/IEEE 29148:2011. (s. f.). ISO.	Página web	https://www.iso.org/standard/45171.html
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

Glosario

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”.

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 softwarehttp://bibliotecadigital.iue.edu.co/xmlui/handle/20.500.12717/153

Clemente, P. J., & Gómez, A. (2014). Aplicación de un proceso de mejora continua en una asignatura de Desarrollo de Software Dirigido por Modelos. http://hdl.handle.net/2099/15497

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

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

Maida, EG, Pacienzia, J. (2015). Metodologías de desarrollo de software [en línea]. Tesis de Licenciatura en Sistemas y Computación. Facultad de Química e Ingeniería “Fray Rogelio Bacon”. Universidad Católica Argentina, 2015 https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf

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

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

Mera Paz, J. A. (19 de 10 de 2016). Pruebas de Calidad software. 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..

Créditos

Nombre	Cargo	Regional y Centro de Formación
Milady Tatiana Villamil Castellanos	Responsable del Ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable de Línea de Producción	Centro de Servicios de Salud - Regional Antioquia
Ervin Andrade	Experto Temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
Peter Emerson Pinchao Solis	Experto temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
Ana Catalina Córdoba Sus	Evaluadora Instruccional	Centro de Servicios de Salud - Regional Antioquia
Blanca Flor Tinoco Torres	Diseñador de Contenidos Digitales	Centro de Servicios de Salud - Regional Antioquia
Luis Jesús Pérez Madariaga	Desarrollador Fullstack	Centro de Servicios de Salud - Regional Antioquia
Edgar Mauricio Cortés García	Actividad Didáctica	Centro de Servicios de Salud - Regional Antioquia
Margarita Marcela Medrano Gómez	Evaluador para contenidos inclusivos y accesibles	Centro de Servicios de Salud - Regional Antioquia
Daniel Ricardo Mutis Gómez	Evaluador para contenidos inclusivos y accesibles	Centro de Servicios de Salud - Regional Antioquia
Luis Gabriel Urueta Álvarez	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Jaime Hernán Tejada Llano	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia