# Calidad del software. Factores y métricas. Estrategias de prueba.

# TEMA 57

ABACUS NT

Oposiciones 2021

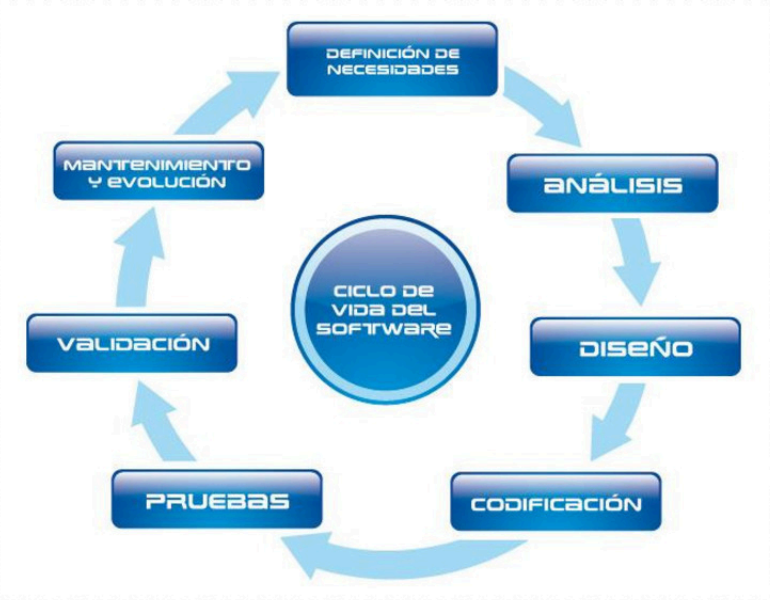
© AbacusNT 2021

## Índice


1. *Introducción*
2. *Ciclo de Vida del Software*
3. *Calidad del software.*
   - 3.1. Concepto de calidad.
   - 3.2. Calidad en la ingeniería del software.
4. *Modelos de calidad de software.*
5. *Factores y métricas.*
   - 5.1. Factores.
   - 5.2. Métricas.
6. *Metodologías de desarrollo*
   - 6.1. Principales Metodologías de Desarrollo
   - 6.2. Enfoques de desarrollo de software
7. *Pruebas*
   - 7.1. Estrategias de prueba.
   - 7.2. Pruebas del software
   - 7.3. Pruebas de integración.
8. *Conclusión.*
   - 8.1. Relación con el sistema educativo
9. *Bibliografía*

# 1. Introducción

La tarea de desarrollo del software es una de las más complejas que existen en informática. Atrás quedó la época en que un solo programador estoicamente abordaba la elaboración de un programa completo sin pestañear. Actualmente el proceso de desarrollo del software involucra a un equipo de profesionales, incluyendo analistas, programadores especializados en unos determinados lenguajes, diseñadores gráficos, etc.




Muchas veces tratamos de acortar el proceso de programación yendo directamente desde el análisis del problema a la codificación del mismo.

Este atajo puede resultar muy tentador e incluso parecer una buena forma de ahorrar tiempo. Sin embargo, este método necesita realmente más tiempo y esfuerzo .

EL desarrollo de una solución general antes de escribir realmente el programa puede ayudar al programador a manejar el problema, tomar el camino correcto y evitar errores innecesarios.

Además, puesto que la mayoría de los programas se usarán una y otra vez, la documentación y mantenimiento de programas son partes importantes de la programación.

# 2. Ciclo de Vida del Software

La creación de software consta generalmente de una serie de pasos claramente diferenciados:

1. **Análisis**: se recopila documentación, se estudian los **requisitos** del usuario final (qué se va a implementar, cuáles son las necesidades del usuario). Se suele empezar con una descripción en lenguaje natural de lo que quiere el usuario. Al final de esta etapa tendremos una descripción clara y precisa de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá.

2. **Diseño**: una vez que sabemos qué debemos hacer, debemos determinar cómo lo haremos. Estudiaremos el problema y lo descompondremos en problemas más pequeños; definiremos la estructura de la solución, identificando los módulos que hay que implementar y sus relaciones; definiremos los algoritmos a emplear y el lenguaje de programación a utilizar, etc.

3. **Implementación y codificación**: se escribe el programa fuente en el lenguaje de programación establecido y se genera el código ejecutable.

4. **Pruebas:** se comprueba que el programa no tenga errores, y que se cumplen los criterios de calidad.

5. **Mantenimiento:** el programador se asegura de que el programa siga funcionando, y lo va adaptando también a nuevos requisitos que puedan ir surgiendo.

# 3. Calidad del software.

## 3.1. Concepto de calidad.

La definición oficial, y más completa, es la de la norma **ISO 8402**, que define la calidad, en general, como: "Totalidad de características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas".

Por tanto, tenemos que tener claro que: La calidad es **multidimensional**, y no se trata de un concepto absoluto, ya que el consumidor la juzga con todo **relativismo**.

## 3.2. Calidad en la ingeniería del software.

Para entender el concepto de calidad aplicado al software, podemos apoyarnos en la definición oficial del estándar IEEE Std.610-1990, el cual define la calidad del software como:

Grado con el que un sistema, componente o proceso cumple:

Los **requisitos** especificados.

Las **necesidades** o expectativas del cliente o usuario.

# 4. Modelos de calidad de software.

El carácter multidimensional y relativo de la calidad, junto con su complejidad inherente, **dificultan** en gran medida la evaluación formal de la calidad del software. Es por ello que existen modelos de calidad, los cuales establecen como se debe hacer dicha evaluación. Estos generalmente están estructurados como se muestra en la siguiente figura:

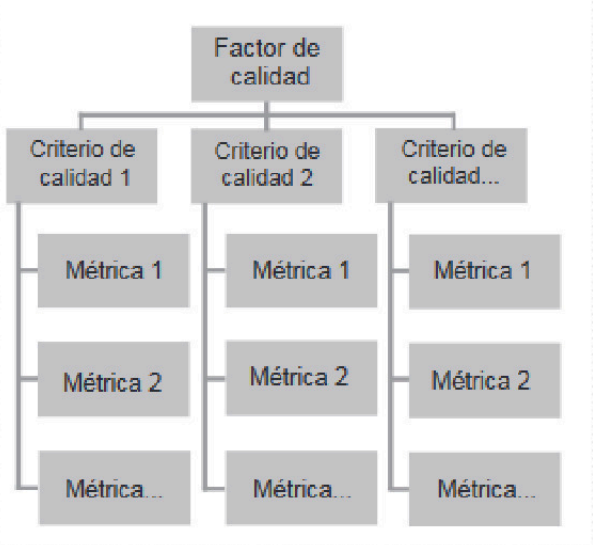



Figura 1. Estructura de un modelo de calidad.

De esta forma se abordar la evaluación desde lo general a lo particular. Así mismo, los modelos de calidad de software se clasifican de acuerdo con el enfoque de evaluación o ámbito en:

1. Organización
2. Proyecto
3. Producto

Como ejemplos de modelos de calidad que se aplican en la actualidad en el sector productivo tenemos:

**CMMI** (Capability Maturity Model Integration): Es de los modelos más utilizados en las empresas de construcción de software, con el propósito de verificar el cumplimiento de estándares de calidad de una organización a partir de la medición con niveles de madurez.

**ISO 25000**: También llamada como SQuaRE, es la sucesora de la norma ISO 9126, la cual estaba basada a su vez en el modelo de McCall. A continuación, vamos a verla más en detalle.

# 5. Factores y métricas.

## 5.1. Factores.

Como hemos visto anteriormente, los modelos de calidad recogen atributos denominados factores de calidad (**atributos de alto nivel**), los cuales a su vez pueden descomponerse en **atributos de bajo nivel** denominados criterios de calidad. Como ejemplo, vamos a ver los factores definidos en la ISO 25000 para el producto software y la calidad en uso, los cuales son denominados atributos, y son:

**Adecuación funcional**: Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas.

**Eficiencia de desempeño:** Esta característica representa el desempeño relativo a la cantidad de recursos necesarios para utilizar el software.

**Usabilidad**: Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario.

**Mantenibilidad**: Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente.

**Compatibilidad:** Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software

**Portabilidad**: Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, o software, a otro.

**Fiabilidad**: Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados

**Seguridad:** Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos.

## 5.2. Métricas.

Los criterios de calidad, a su vez, pueden asociarse con un conjunto de atributos de bajo nivel medibles directamente mediante métricas de calidad. Éstas son imprescindibles en los procesos de aseguramiento de la calidad, ya que difícilmente puede mejorar algo, si no se mide.

A continuación, presentaremos algunas de las métricas más utilizadas en la actualidad.

**LOC (Line of code):** Mide el número de líneas, como indicador del tamaño de un producto.

**Densidad de comentarios**: Mide el número de líneas de comentarios entre el número total de líneas del sistema.

**Porcentaje de código duplicado**: Indica el porcentaje de líneas de código que se encuentra duplicado frente al total de líneas de código.

LCOM4 **(Lack of Cohesion in Methods 4):** Es una propuesta a partir de la métrica inicial LCOM1 de Chidamber y Kemerer, para indicar la falta de cohesión. Si es mayor de 1 quiere decir que se nos aconseja que dividamos la clase en varias subclases. Para el cálculo del LCOM4 de una clase básicamente lo que hace es medir el número de relaciones diferentes que se establecen entre métodos,

**Profundidad del árbol de herencia**: Longitud del camino máximo entre un nodo y la raíz del árbol, en una jerarquía de clases. Depende del proyecto, pero no se aconseja normalmente tener un valor por encima de 3.

**Complejidad ciclomática de McCabe**: Esta métrica es un indicador del número de caminos independientes que existen en un grafo, sirviendo para indicar la complejidad inherente de un programa. La complejidad de McCabe V(G) se puede calcular:

V (G) = a - n + 2, siendo a el número de arcos o aristas del grafo y n el número de nodos.

En el sector productivo, para calcular estas métricas, normalmente se utilizan herramientas automáticas de análisis estáticos de código, como por ejemplo SonarQube

# 6. Metodologías de desarrollo

Una metodología de desarrollo es una recopilación de técnicas y procedimientos estructurados en fases que van a dar como resultado de su aplicación un producto software de calidad.

Al comienzo de la era de la informática las técnicas de desarrollo eran totalmente artesanas y sin metodología ninguna, A finales de los años setenta hay un fuerte tirón en la demanda de aplicaciones institucionales de gran complejidad con requerimientos muy exigentes a veces difíciles de identificar. Estos dos factores, aumento de demanda y aumento de complejidad, contribuyen a una importante crisis del software, que no es capaz de dar solución con las herramientas y metodologías existentes a las peticiones de los usuarios, a esto se le puede sumar el estado en que se encontraba el hardware.

A partir de estos momentos hay una evolución en varias líneas o frentes:

- Surgen nuevas técnicas metodológicas como superación del ciclo de vida clásico (técnicas estructuradas).
- Empiezan a surgir nuevas herramientas de desarrollo basadas en el ordenador.
- El hardware evoluciona, aparecen nuevos entornos, máquinas cada vez más pequeñas y a la vez más potentes sobre las que son posibles nuevos sistemas operativos y nuevos entornos de desarrollo basados en las nuevas metodologías y herramientas.

En este escenario en los años ochenta nace el término **CASE**, Computer Aided Software Engineering, o Ingeniería de Software Asistida por Ordenador, una evolución dentro del desarrollo de sistemas de información que pretende ser la solución a la crisis del software tal y como la hemos descrito anteriormente, es decir, en el entorno institucional con aplicaciones cada vez más complejas.

## 6.1. Principales Metodologías de Desarrollo

### MetricaV3

Se definía como "metodología de planificación, desarrollo y mantenimiento de sistemas de información" y fue creada por el Ministerio de Administraciones Públicas (MAP) español.

La versión 3 de Métrica fue, y es, la más famosa de las versiones. Y aún la usan muchas empresas, ya que en muchas subcontrataciones de la administración era, y es, un requisito obligado.

Aquella era una época en la que muchos países querían tener su propia metodología: **Merise** en Francia, **SSADM** en Reino Unido y **Métrica** en España.

### Agile

Con esta metodología, el desarrollo de software se divide en proyectos más pequeños, por lo que es ideal para los ciclos de lanzamiento corto y finitos, como los requeridos para las actualizaciones de un producto existente. Es conveniente cuando se tiene experiencia en la preparación de documentación, el equipo de desarrollo trabaja estrechamente y, preferiblemente, también en la misma ubicación física.

### Scrum

Los proyectos que utilizan esta metodología otorgan un peso considerable a la inteligencia, experiencia y habilidades que los miembros del equipo de desarrollo aportan a la hora de resolver problemas. Las tareas de proyecto se llevan a cabo en ciclos cortos conocidos como ***sprints***, muy manejables y adecuadamente priorizados, que permiten mostrar el progreso de forma muy sencilla. Los proyectos más largos se beneficiarían de la estructuración de este método, en comparación con otros modelos de desarrollo de software y uno de los motivos es que los desarrolladores se sienten comprometidos con las metas y responsables del éxito de la iniciativa.

### Programación Extrema

Su enfoque de evolución constante resulta muy beneficioso, sobre todo, si se tiene en cuenta que, uno de los procedimientos a aplicar es descomponer las tareas de desarrollo para incluir sólo las actividades críticas. Precisamente por eso resulta un método muy indicado para los casos en que el entorno de desarrollo carece de estabilidad o si los requisitos aplicables al proyecto son inciertos.

### Programación en pareja

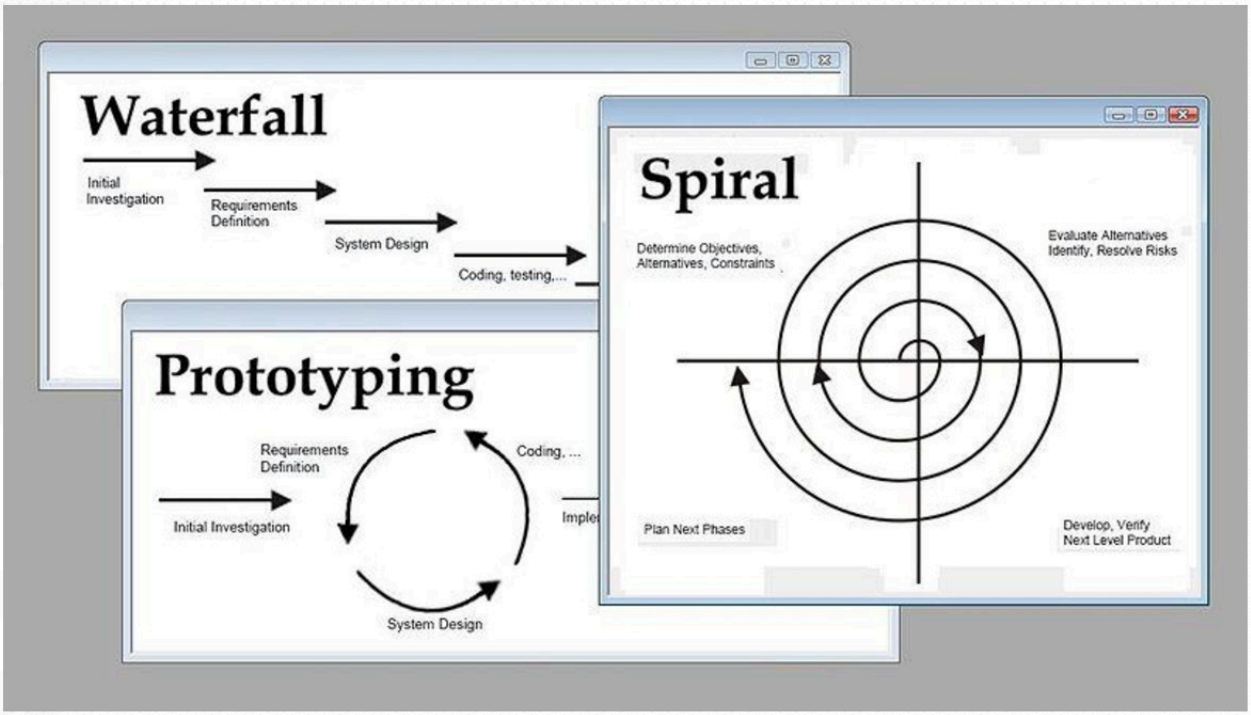
Consiste en juntar a dos programadores para que trabajen juntos en una sola estación de trabajo, facilitando que cada desarrollador se centre en diferentes aspectos del proyecto para conseguir que se reduzcan los costes de las pruebas y también los defectos en los programas. Merece la pena

apostar por esta alternativa cuando el equipo de desarrollo es pequeño y seguro, y siempre que los desarrolladores estén de acuerdo.

### Kanban

en base al sistema just in time, esta metodología trata de evitar los cuellos de botella, mediante una identificación muy precisa de las tareas en curso que funciona muy bien en proyectos que no sean demasiado complejos. Permite ser complementado por otras metodologías y es muy recomendable su utilización cuando se tiene confianza plena en el equipo de desarrollo.

## 6.2. Enfoques de desarrollo de software




Marcel Douwe Dekker - Own work by uploader, based on Selecting a development approach at cms.hhs.gov. CC BY-SA 3.0

**Cada metodología de desarrollo de software** tiene más o menos su propio enfoque para el desarrollo de software. Estos son los enfoques más generales, que se desarrollan en varias metodologías específicas. Estos enfoques son los siguientes:

### Modelo en cascada

Es un proceso secuencial, fácil de desarrollo en el que los pasos de desarrollo son vistos hacia abajo (como en una cascada de agua) a través de las fases de análisis de las necesidades, el diseño, implantación, pruebas (validación), la integración, y mantenimiento.

Los principios básicos del modelo de cascada son los siguientes:

- El proyecto está dividido en fases secuenciales.

- Se hace hincapié en la planificación, los horarios, fechas, presupuestos y ejecución de todo un sistema de una sola vez.
- Un estricto control se mantiene durante la vida del proyecto a través de la utilización de una amplia documentación escrita, así como a través de comentarios y aprobación.

### Iterativo

El prototipo permite desarrollar modelos de aplicaciones de software que permiten ver la funcionalidad básica de la misma, sin necesariamente incluir toda la lógica o características del modelo terminado. El prototipo permite al cliente evaluar en forma temprana el producto, e interactuar con los diseñadores y desarrolladores para saber si se está cumpliendo con las expectativas y las funcionalidades acordadas. Los Prototipos no poseen la funcionalidad total del sistema pero si condensa la idea principal del mismo, Paso a Paso crece su funcionalidad, y maneja un alto grado de participación del usuario.

### Incremental

Provee una estrategia para controlar la complejidad y los riesgos, desarrollando una parte del producto software reservando el resto de aspectos para el futuro.

### Espiral

La atención se centra en la evaluación y reducción del riesgo del proyecto dividiendo el proyecto en segmentos más pequeños y proporcionar más facilidad de cambio durante el proceso de desarrollo, así como ofrecer la oportunidad de evaluar los riesgos y con un peso de la consideración de la continuación del proyecto durante todo el ciclo de vida.

Cada viaje alrededor de la espiral atraviesa cuatro cuadrantes básicos: (1) determinar objetivos, alternativas, y desencadenantes de la iteración; (2) Evaluar alternativas; Identificar y resolver los riesgos; (3) desarrollar y verificar los resultados de la iteración, y (4) plan de la próxima iteración.3

Cada ciclo comienza con la identificación de los interesados y sus condiciones de ganancia, y termina con la revisión y examinación.

### Rapid Application Development (RAD)

El desarrollo rápido de aplicaciones (RAD) es una metodología de desarrollo de software, que implica el desarrollo **iterativo** y la construcción de **prototipos**.

Intenta reducir los riesgos inherentes del proyecto partiéndolo en segmentos más pequeños y proporcionar más facilidad de cambio durante el proceso de desarrollo.

Orientación dedicada a producir sistemas de alta calidad con rapidez, principalmente mediante el uso de iteración por prototipos, promueve la participación de los usuarios y el uso de herramientas de desarrollo computarizadas.

Hace especial hincapié en el **cumplimiento de la necesidad comercial**, mientras que la ingeniería tecnológica o la excelencia es de menor importancia.

### Otros enfoques de desarrollo de software

Metodologías de desarrollo Orientado a objetos, Diseño orientado a objetos (OOD) de Grady Booch, también conocido como **Análisis y Diseño Orientado a Objetos (OOAD)**. El modelo incluye seis diagramas: de clase, objeto, estado de transición, la interacción, módulo, y el proceso.

**Top-down programming**, evolucionado en la década de 1970 por el investigador de IBM Harlan Mills (y Niklaus Wirth) en Desarrollo Estructurado.

**Proceso Unificado**, es una metodología de desarrollo de software, basado en UML. Organiza el desarrollo de software en cuatro fases, cada una de ellas con la ejecución de una o más iteraciones de desarrollo de software: creación, elaboración, construcción, y las directrices. Hay una serie de herramientas y productos diseñados para facilitar la aplicación. Una de las versiones más populares es la de **Rational Unified Process**.

# 7. Pruebas

## 7.1. Estrategias de prueba.

Las pruebas tienen como objetivo principal la detección de defectos, con objeto de mejorar la calidad del software. La estrategia de aplicación y la planificación de éstas se hace en base a distintos niveles, los cuales, según el ISTQB (International Software Testing Qualifications Board), son:

**Pruebas unitarias:** Tienen como objeto comprobar el funcionamiento de un único componente.

**Pruebas de integración**: Su objetivo principal es comprobar la interacción de diferentes partes de un sistema, validando la comunicación entre sus interfaces.

**Pruebas de sistema**: Estas pruebas buscan probar si los componentes de un sistema son compatibles, e interactúan correctamente.

**Pruebas de aceptación de usuario**: el objetivo de estas pruebas es validar la disposición de un sistema para su despliegue y uso.

Los niveles de las pruebas están estrechamente relacionados con las **fases del ciclo** del software, así en el modelo o método **en V** (utilizado por ejemplo por la administración alemana) se establece dicha relación de forma **explícita**:

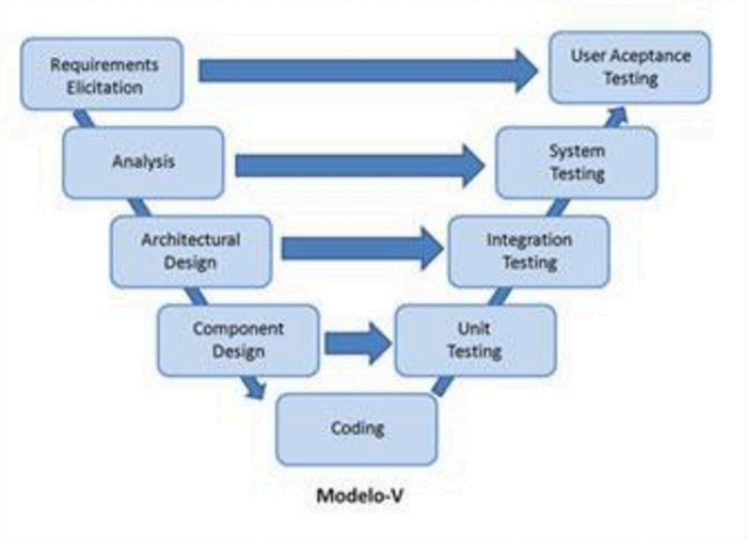



Fig. Modelo en V

## 7.2. Pruebas del software

Las pruebas del software son de verificación y validación; debemos comprobar que estamos desarrollando el software correctamente y de que sea el producto correcto.

Para ello es importante organizar las pruebas con **minuciosidad**, **planificándola** y diseñando los **casos de prueba** que permitan **recopilar y evaluar** los resultados.

Una prueba debe ser siempre **flexible** respecto a las acciones que pueda tomar el usuario y a la vez **rígida** respecto al seguimiento de la misma.

## 7.3. Pruebas de integración.

Las pruebas de integración validan la correcta **instalación** del programa así como su buen funcionamiento en la fase previa a la **explotación**.

Durante este tiempo **convivirán el sistema antiguo (en caso de existir) y el nuevo.**

Es importante que el software funcione no sólo aislado sino en el entorno real para el que ha sido diseñado. Para esto se establece un periodo de prueba (**fase beta**) en el que se pone en funcionamiento el programa pero es seguido de cerca por el desarrollador, haciendo cuantas correcciones sean necesarias para afinar su funcionamiento:

- Los errores deben ser corregidos en su totalidad y sin excepciones.
- Si los errores son críticos se debe suspender la actividad del nuevo sistema y volver a activar el anterior.

Las pruebas de integración finalizarán con un **informe final** donde se documente el estado del nuevo sistema y se incluyan los errores subsanados, las observaciones de los usuarios, etc.

© AbacusNT 2021

# 8. Conclusión.

La generación de conocimiento necesita un poso de calidad que se extiende desde el dato mismo hasta el programa desarrollado para interactuar con la información. Las métricas de calidad de software permiten monitorizar un producto para determinar su nivel de calidad, aunque, el seguimiento que este tipo de medidas permiten llevar a cabo brinda la oportunidad de conocer muchas más de una solución.

Un enfoque proactivo permite la identificación temprana de errores o defectos que pueden ser corregidos a tiempo, eliminando de raíz problemas mayores. Los efectos positivos empiezan a notarse y sus beneficios aumentan en un ciclo de mejora continúa propiciado por el control de las métricas de calidad de software.

## 8.1. Relación con el sistema educativo

GS – DAW - DAM –Entornos de Desarrollo (PES)

# 9. Bibliografía

Pressman, R. S. **Ingeniería del Software. Un enfoque práctico**, 3ª edición. Ed. McGraw-Hill, 2000.

Sommerville, I.: **Ingeniería de Software**. 6ª Edición. Addison-Wesiey Iberoamericana, 2002.

Piattini, M. y otros: Análisis y diseño detallado de Aplicaciones Informáticas de Gestión. RA-MA, 1996.

Cerrada, J. A.: **Introducción a la Ingeniería del Software**. 1ª Edición de 2000. Editorial Centro de Estudios Ramón Areces, S.A.

https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software