**CAPITULO 1 - SOFTWARE Y INGENIERÍA EN SOFTWARE**

**SOFTWARE:** conjunto de instrucciones, estructuras de datos y documentos. Tiene un papel dual: como producto y como vehículo.

**CARACTERISTICAS**

1. **EL SW SE DESARROLLA PERO NO SE MANUFACTURA:** son actividades diferentes en lo fundamental. La manufactura incluye problemas de calidad inexistentes en el SW, la relación entre gente utilizada y el trabajo realizado es deferente. Lo mismo con el enfoque de construcción del producto
2. **EL SW NO SE DESGATA, PERO SE DETERIORA:** Existe una tasa de fallas en función del tiempo que se representa como una curva de ballena, que indica el HW tiene cierta cantidad de fallas al principio. Cuando se corrigen, la tasa de fallas baja y el HW se desgasta al pasar el tiempo. Conforme a los cambios de HW, el SW cambia, con la posibilidad de introducir errores. Estos producen picos en la curva y antes de que regrese a su estado original, sufre otro pico.
3. **A PESAR DE QUE LA INDUSTRIA TIENE UNA TENDENCIA HACIA LA CONSTRUCCIÓN POR COMPONENTES, LA MAYORÍA DEL SW AÚN SE CONSTRUYA A MEDIDA:** es debido a que el desarrollo de componentes en SW es escasa y difícil de implementar por la alta compatibilidad que requiere, aunque traería beneficios.

**CATEGORÍAS DE SW**

* **DE SISTEMAS:** colección de programas escritos para servir a otros programas
* **DE APLICACIÓN:** resuelven una necesidad de negocios especifica
* **CIENTÍFICO Y INGENIERÍA:** usa algoritmos devoradores de números
* **EMPORTADO:** reside en la ROM. Implementa y controla características y funciones
* **DE LÍNEA DE PRODUCTOS**: para ser usado por muchos clientes distintos
* **APLICACIONES BASADAS EN WEB**: Usa WebApps (archivos de hipertexto ligados)
* **DE INTELIGENCIA ARTIFICIAL:** usa algoritmos no numéricos.

**RETOS**

* **COMPUTACIÓN UBICUA:** desarrollar SW para dispositivos alternativos comunicados por red
* **ALIMENTACIÓN DEL A RED**: desarrollar SW que beneficie a usuarios específicos globalmente
* **FUENTE ABIERTA**: distribución de código fuente para que los usuarios hagan modificaciones
* **LA NUEVA ECONOMÍA:** desarrollar SW para facilitar la comunicación y distribución de productos en masa.

**SOFTWARE HEREDADO**

Es un software caracterizado por su longevidad y el ser critico para los negocios (en el sentido de que muchas empresas dependen de el para lograr su funcionamiento, como los bancos)

**CAPITULO 2 – EL PROCESO: UNA VISIÓN GENERAL**

**PROCESO DE SOFTWARE:** marco de trabajo para las tareas que requieren construir un SW de alta calidad. (El marco de trabajo establece las bases identificando un conjunto de actividades, que contiene acciones o tareas de ingeniería en software)

**INGENIERÍA EN SOFTWARE:** Es la aplicación o el estudio de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software.

**Abarca un proceso** (elementos que mantiene juntos todos los estratos de la tecnología y permite el desarrollo raciona y a tiempo del software), **método** (proporciona los como técnicos para la construcción del software) **y herramientas** (soporte automático o semiautomático para el proceso y los métodos)

**MARCO DE TRABAJO GENÉRICO PARA EL PROCESO**

* **Comunicación:** colaboración y comunicación con el clientes
* **Planeación:** plan de trabajo. Incluye tareas, riesgos, recursos, productos y programa de trabajo
* **Modelado:** requisitos de software y el diseño.
* **Construcción:** Generación de código y pruebas
* **Despliegue:** entrega al cliente y pruebas beta

El marco de trabajo completa una serie de actividades sombrilla. Estas ocurren a lo largo de todo el proyecto de software y se enfocan principalmente en la gestión, rastreo y control del proyecto.

**IMCM: Integración Del Modelo De Capacidad De Madurez**

Representa un modelo de proceso completo que puede tener dos formas: continuo o discreto.

El continuo describe un proceso en dos dimensiones (área de proceso y nivel de capacidad). Se define cada área en función a las metas específicas y las practicas requeridas en la misma. La diferencia con el discreto es que tiene cinco niveles de madurez, en vez de cinco niveles de capacidad. Los niveles de capacidad son:

* Nivel 0: Incompleto
* Nivel 1: Realizado
* Nivel 2: Administrado
* Nivel 3: Definido
* Nivel 4: Administrado de forma cuantitativa.
* Nivel 5: Mejorado

**PATRÓN DE PROCESO:** patrones que definen un conjunto de actividades, acciones, tareas de trabajo o comportamiento relacionados para construir un proceso que satisfaga las necesidades del proyecto. Incluye nombre de patrón, propósito (objetivo) y tipo. El tipo puede ser:

* **Patrón de tarea**: define una tarea de trabajo o acción de ingeniera en software
* **Patrón de escenario**: define una actividad del marco de trabajo para el proceso
* **Patrón de fase**: definen la secuencia de actividades del marco del trabajo junto con el proceso.

**CAPITULO 3 – MODELOS PRESCRIPTIVOS DE PROCESO**

Son modelos que describen un conjunto de actividades dentro del marco del trabajo para que los adopte el software. (Comunicación, planeación, modelado, construcción y despliegue)

|  |  |  |
| --- | --- | --- |
| Modelo | Descripción | Dificultades/Explicación |
| En Cascada | Sistemático y secuencial. | Requiere que se siga un flujo secuencial.  Requiere que el cliente establezca todos los requisitos del producto de manera explicita.  El software estará listo luego de mucho tiempo. |
| Incremental | Cascada iterativo. | Entrega lanzamiento (incrementos) que proporcionan progresivamente mayor funcionalidad al software a medida que se entregan al cliente.  Es útil cuando no hay personal necesario para implementarlo completamente al principio. |
| DRA | Desarrollo rápido de aplicaciones, adaptado del modelo cascada. Incremental. | Utiliza varios equipos DRA que se encargan del modelado y la construcción. Tardan 60-90 días.  Necesita suficientes recursos para crear equipos DRA.  Los desarrolladores y clientes deben comprometerse con las actividades rápidas para completar el software.  Se debe modular adecuadamente.  Convertir interfaces en complementos afecta el rendimiento.  Es inapropiado si los riesgos técnicos son altos |
| Prototipos | Se implementa en todos los modelos de proceso. | Valido si el desarrollador esta inseguro de la eficacia de un algoritmo, los requisitos de entrada, procesamiento y salida, la adaptabilidad y la interacción humano-maquina.  Al hacerse muy rápido, no se suele considerar la calidad o facilidad de mantenimiento a largo plazo o se use un lenguaje de programación inadecuado. |
| En Espiral | Modelo de proceso evolutivo, combina la construcción de prototipos y el modelo en cascada. | Entregas evolutivas del software.  Se deben establecer todos los riesgos técnicos de todas las etapas del proyecto.  Requiere habilidad para evaluar el riesgo. |
| De Desarrollo Concurrente | Define una serie de eventos que dispararán transiciones de estado para cada una de las actividades | Para cuando inicia la comunicación, el sistema queda “en espera de cambios”. Cuando se modela, pasa a estado de “en desarrollo” y cuando termina una fase de modelado queda “en espera de cambios”. |
| Desarrollo basado en componentes | Modelo en espiral, evolutivo e iterativo | Se basa en configurar aplicaciones a partir de componentes de software ya empaquetados.  Lleva a la reutilización de software. |
| De métodos formales | Especificación matemática de software | Se especifica, desarrolla y verifica software aplicando una notación matemática rigurosa.  Es muy caro y consume mucho tiempo.  Se requiere una capacitación detallada.  Es difícil darse a entender al cliente, debido a que el lenguaje es muy específico. |
| Desarrollo orientado a aspectos | Intereses + funciones + características +información | Define los intereses generales del software |

**CAPITULO 4 – PROCESO DE SOFTWARE Y METIRCAS DE PROYECTOS**

**MEDIDA:** proporciona una indicación cuantitativa de la extensión, dimensión, cantidad, capacidad o tamaño de algunos atributos de un proceso o producto

**MÉTRICAS DE SOFTWARE:** relata de alguna forma las medidas individuales sobre algún aspecto.

* Privadas: métricas no visibles al individuo y solo sirven como indicador a ese individuo
* Públicas: métricas visibles a todos los miembros y sirven para cambios estratégicos y cambios tácticos durante el proyecto de software

**INDICADORES: \*** métrica o combinación de métricas que proporcionan una visión profunda del proceso **(indicadores de proceso)**, proyecto o producto **(indicadores de proyecto).**

**INDICADORES DE PROYECTO:** \* Permitiendo:

* Evaluar el proyecto de software
* Seguir la pista de riesgos potenciales
* Detectar áreas de problema antes de que se conviertan en críticas
* Ajustar el flujo de tareas de trabajo
* Evaluar la habilidad del equipo para controlar el proyecto

**Métricas de Proyecto**: Son tácticas. Calcula estimaciones de esfuerzo y tiempo para el proyecto actual según los proyectos anteriores. Se usan para minimizan la planificación de desarrollo y evaluar la calidad de los productos.

**Mediciones De Software**

* Directas: tiene en cuenta las características especificas del producto (ej: coste, esfuerzo)
* Indirectas: tiene en cuenta las capacidades del producto (ej: calidad, complejidad, eficacia)

**Medidas Orientadas Al Tamaño**: mide la calidad y productividad según el tamaño de software

* Errores por KLDC
* Defectos por KLDC
* Paginas de documentación por KLCD
* $ por página de documentación
* $ por LDC
* Errores por LDC
* Errores por Persona-mes

**PUNTOS DE FUNCIÓN (PF):** relación empírica entre las medidas contables del dominio de información y la evaluación de la complejidad del software. (Nro. De entradas de usuario, nro. De salidas de usuario, nro. De archivos, nro. De interfaces externas)

**Punto De Características:** ídem PF, pero agrega los algoritmos (problema de calculo limitado)

**MEDIDA DE LA CALIDAD**

* Corrección: mide la capacidad para lleva a cabo una función requerida.
* Integridad: mide la capacidad para resistir ataques contra la seguridad.
* Facilidad de Uso: mida cuanto es de amigable con el usuario.
* Facilidad de mantenimiento: mida la capacidad para corregir un error, mejorar o adaptar.

**MÉTRICAS PARA ORGANIZACIONES PEQUEÑAS**

* Tiempo y/o Esfuerzo requerido para realizar el cambio
* Tiempo (horas o días) desde el momento que se hace una petición hasta que se completa la evaluación
* Tiempo desde que termina la evaluación hasta un nuevo cambio
* Esfuerzo (horas x persona) para la evaluación
* Errores y/o defectos descubiertos

**CAPITULO 13 – ESTRATEGIA DE PRUEBA DE SOFTWARE**

**PRUEBA:** conjunto de actividades planteadas con anticipación y sistemáticas

* **VERIFICACIÓN:** conjunto de actividades que aseguran que el software implemente correctamente una función específica
* **VALIDACIÓN:** es un conjunto diferente de actividades que aseguran que el software construido se corresponde con los requisitos del cliente

**Grupo independiente de pruebas:** grupo que realiza pruebas independientes, eliminando el conflicto de intereses

**ESTRATEGIA DE PRUEBA PARA ARQUITECTURAS CONVENCIONALES DE SOFTWARE**

* **PRUEBA DE UNIDAD:** prueba de componente
* **PRUEBA DE INTEGRACIÓN:** prueba el diseño y la construcción del a arquitectura
* **PRUEBA DE VALIDACIÓN:** prueba que valida los requisitos
* **PRUEBA DE SISTEMA:** prueba todo el software integrado

**DIRECTRICES O ESTRATEGIAS DE PRUEBA**

1. Especificar los requisitos del producto de manera cuantificable mucho antes de que empiecen las pruebas
2. Establece explícitamente los objetivos de la prueba
3. Comprender cuales son los usuarios de software y desarrollar un perfil para cada categoría de software
4. Desarrollar un plan de prueba que destaque la prueba de ciclo rápido
5. Desarrollar un enfoque de mejora continua para el proceso de prueba
6. Construir un software robusto diseñado para probas
7. Usar revisiones técnicas forales y efectivas como filtro previo a la prueba
8. Realizar revisiones técnicas formales para evaluar la estrategia de prueba y los propios casos de prueba

**PRUEBA DE INTEGRACIÓN:** Es una técnica sistemática para construir software y aplicar pruebas para descubrir errores, a partir de pequeños incrementos.

**INTEGRACIÓN DESCENCENTE**

* Primero en profundidad: integra todos los módulos en la ruta de control principal. Arbitrario y depende del observador. Se construye la ruta de control central y a la derecha
* Primero en anchura: integra todos los módulos directamente subordinados a cada nivel desplazándose horizontalmente.

**PROCESO DE INTEGRACIÓN**

1. El modulo de control principal se usa como controlador de prueba, y se sustituyen todos los módulos directamente subordinados al control principal
2. Dependiendo del enfoque de integración (en profundidad o anchura), se reemplazan (uno por uno) los resguardos por módulos reales
3. Se aplican pruebas cuando se integra un nuevo modulo
4. Al completar las pruebas, se reemplaza el resguardo por el modulo real
5. Se aplica prueba de regresión para asegurarse de que no se hubo nuevos errores.

Integración descendente: prueba desde los módulos atómicos (niveles más bajos) a los niveles más altos. (De abajo hacia arriba). Siempre está disponible el procesamiento requerido para los componentes subordinados y no necesita resguardos

1. Se combinan los módulos de bajo nivel en grupos que realicen una subfunción especifica
2. Se escribe un controladores que coordine la entrada y salida de casos de prueba
3. Se prueba el grupo
4. Se eliminan los controladores y se combinan los grupos de forma ascendente.

**PRUEBA DE HUMO:** es una prueba de integración que sirve para proyectos donde hay poco tiempo disponible para pruebas, haciendo pruebas diarias.Lo que hace es

1. Integrar los componentes con código , combinándolos en grupos
2. Diseñar pruebas que permitan descubrir errores
3. Se integra cada grupo de componentes con otros grupos y diariamente se aplican estas pruebas a todo el producto

* Minimiza el riesgo de integración.
* Mejora la calidad del producto final.
* Simplifica el diagnostico y corrección de errores.
* Facilita la evaluación del progreso.

**ESTRTEGIAS DE PRUEBA PARA SOFTWARE O.O.**

**PRUEBA DE UNIDAD O.O:** Prueba las operaciones dentro de una clase. Similar a la prueba de unidad convencional.

**PRUEBA DE INTEGRACIÓN O.O:** Hay dos estrategias diferentes

* **Basada En Subprocesos:** Seintegran las clases que responden a una entrada o evento del sistema. Se integra y prueba individualmente
* **Basada En Uso:** Se prueban las clases que tiene muy copas clases servidoras y luego las dependiente que usan clases independiente.

**PRUEBA DE GRUPO:** Paso de la prueba de integración que intenta descubrir errores en colaboraciones de las clases.

**PRUEBA DE VALIDACIÓN:** Prueba que trata de describir errores en el nivel de requisitos.

**REVISIÓN DE LA CONFIGURACIÓN:** asegura que todos los elementos de software se hayan desarrollado apropiadamente

**PRUEBA ALFA Y BETA**

* **PRUEBA ALFA:** son pruebas que realiza el usuario en el lugar de trabajo del desarrollador.
* **PRUEBA BETA:** son pruebas que realiza el usuario en le lugar de trabajo del cliente.

La diferencia está en que el desarrollador suele no estar en la prueba beta.

**PRUEBA DE SISTEMA**: Prueba que ejercita profundamente el sistema de cómputo

* **PRUEBA DE RECUPERACIÓN:** prueba de sistema que oblicue al software a fallar y luego verifica que pueda recuperarse correctamente
* **PRUEBA DE SEGURIDAD**: prueba de sistema que comprueba que los mecanismos de seguridad funcionen correctamente
* **PRUEB DE RESISTENCIA:** prueba de sistema que intenta sobrecargar al programa (estrés) con situaciones anormales
* **PRUEBA DE SENSIBILIDAD:** es una variante de la prueba de resistencia que intenta descubrir combinaciones de cados dentro de clases de entrada que causen inestabilidad o procesamiento inapropiado

**DEPURACIÓN:** Es la acción de eliminar errores.

**TECTICAS DE DEPURACION**

* **FUERZA BRUTA**: El más común. Deja que el sistema encuentre el error. Se aplica cuando todo lo demás falla.
* **RESTREO HACIA ATRÁS:** lo que hace es recorrer el código hacia atrás desde el punto donde se origino hasta hallar la causa
* **PARTICION BINARIA:** consiste en crear hipótesis de la causa y tratar de probar o refutar la hipótesis.

**CAPITULO 14 - TÉCNICAS DE PRUEBA DE SOFTWARE**

**CAJA NEGRA:** son pruebas de comportamiento que se centran en los requisitos funcionales de software. (Interfaz)

**CAJA BLANCA:** método de diseño que usa la estructura de control del diseño procedimental para obtener caso de prueba. (Detalle procedimental)

**CASOS DE PRUEBA**: es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta ahora.

**PRUEBA:** proceso de ejecución de un programa con la intención de descubrir un error

**FACILIDAD DE PRUEBA** (indica el grado de la prueba)

* **OPERATIVIDAD:** cuanto mejor funcione, con mayor eficiencia podrá probarse
* **OBSERVABILIDAD:** lo que se ve es lo que se prueba.
* **CONTROLABILIDAD:** Cuanto mejor se controle el software, mejor se autorizarán y mejorarán las pruebas.
* **CAPACIDAD PARA DESCOMPONER:** cuanto mas se controle elalcance, más se aislarán los problemas y las pruebas se aplicarán con más inteligencia.
* **SIMPLICIDAD:** Cuanto menos haya que probar, más rápido se hará.
* **ESTABILIDAD:** Cuantos menos haya, menos alteraciones habrá en la prueba.
* **FACILIDAD DE COMPRENSIÓN:** cuanta mayor información se tenga, con mayor inteligencia se hará a la prueba.

**CARACTERISTICAS DE LA BUENA PRUEBA**

* Tiene alta probabilidad de encontrar el error.
* No es redundante.
* Debe ser la mejor de su clase.
* No debe ser ni demasiado simple ni demasiado compleja.

**PRUEBAS DE CAJA BLANCA:** Se podrán derivar casos de prueba que

* Garanticen que todas las rutas independientes dentro del modulo se han ejercitado por lo menos una vez
* Ejerciten los lados verdades y false de todas las decisiones lógicas
* Ejecuten todos los bucles en sus limites y dentro de sus limites operacionales
* Ejerciten estructuras de datos internos para asegurar su validez

**TÉCNICAS DE PRUEBA DE CAJA BLANCA**

* **PRUEBA DE LA RUTA BÁSICA:** es una técnica que permite que el diseñador de casos de prueba obtenga una medida de complejidad lógica de un diseño procedimental y que use como guía para definir un conjunto básico de rutas de ejecución.
* **PRUEBA DE CONDICIÓN:** ejercita las condiciones lógicas de un programa.
* **PRUEBA DE FLUJO DE DATOS:** selecciona las rutas de prueba de un programa de acuerdo con la ubicación de las definiciones y el uso de variables en el programa
* **PRUEBA DE BUCLES:** se centra en la validez de la construcción de bucles (simples, anidados, concatenados y no estructurados)

**TÉCNICAS DE PRUEBA DE CAJA NEGRA**

* **MÉTODOS GRÁFICOS DE PRUEBA:** sirve para entender los objetos que se modelan en le software y las relaciones que hay entre ellos. Para ellos, se definen una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas, empezando por una gráfica (colección de nodos y enlaces), pesos del nodo (propiedades del nodo) y pesos de enlace (características del enlace).
* **PARTICIÓN EQUIVALENTE:** divide el dominio de entrada en clases de datos que probablemente ejercitarán una función específica.
* **ANÁLISIS DE VALORES LÍMITE:** prueba la capacidad del programa de manejar datos en los límites de aceptabilidad.
* **PRUEBA DE TABLA OCTOGONAL:** método eficiente y sistemático de probar sistemas que usa números pequeño de parámetros de entrada.

**MÉTODOS DE PRUEBAS ORIENTADOS A OBJETOS**

* **PRUEBA BASADA EN FALLAS** crea hipótesis sobre posibles fallas y luego idea pruebas para probar o refutar la hipótesis
* **PRUEBA DE ESCENARIO:** son pruebas basadas en la interacción del usuario con el software
* **ESTRUCTURA DE SUPERFICIE:** es la estructura visible para el usuario. Basada en caja negra. La estructura de fondo se basa en caja blanca

**MÉTODOS DE PRUEBA A NIVEL DE CLASES**

* **PRUEBA ALEATORIA:** prueba el historial de las clases (los métodos varias veces en distinto orden)
* **PRUEBA DE PARTICIÓN:**
  + **Basada en estado:** ordena en categorías las operaciones de clase a según su capacidad para cambiar de estado
  + **Basado en atributos:** ordena en categorías las operaciones de clase a según los atributos que usan
  + **Basado en categorías:** atributos ordena en categorías las operaciones de clase a según su función genérica.

**CAPITULO 23 – PRUEBAS ORIENTADAS A OBJETOS**

**SE PRUEBA DESDE LO PEQUEÑO HASTA LO GRANDE**

**ESTRATEGIAS DE PRUEBA O. O.**

* **PRUEBA DE UNIDAD:** prueba al sistema en **unidades atómicas.** Se aplican a cada clase.
* **PRUEBA DE INTEGRACIÓN:** de prueban las **unidades integradas** según:
  + **PRUEBA BASADA EN HILOS:** prueba las clases que responden a un evento o suceso
  + **PRUEBA BASADA EN USO**: prueba las clases que usan pocas clases servidoras (pocas clases dependientes)
  + **PRUEBA DE AGRUPAMIENTO:** fase de la prueba de integración. Se integran las clases colaboradoras para diseñar casos de prueba
* **PRUEBA DE VALIDACIÓN:** se centra en lo que **hace el usuario** (acciones y salidas). Orientada a **caja negra**.

**DISEÑOS DE CASOS DE PRUEBA O. O.**

1. Identificar cada caso de prueba separadamente
2. Desarrollar los pasos a seguir conteniendo lista de estados, mensajes y operaciones, excepciones, condiciones externas e información adicional.

* **PRUEBAS BASADAS EN ERRORES:** Realiza pruebas con alta probabilidad de encontrar fallos. Crea hipótesis de posibles fallos y luego hace pruebas para probarla. Para atributos y operaciones. Se usa en la prueba de unidad.
* **PRUEBAS BASADAS EN ESCENARIO:** se centra en lo que hace el usuario, revela errores de integración. Se usa en la prueba de validación.
* **ESTRUCTURA DE PRUEBA SUPERFICIAL Y PROFUNDA --** Para Prueba De Validación
  + **PRUEBA SUPERFICIAL:** pruebas para validar la estructura visible para el usuario
  + **PRUEBA PROFUNDA:** pruebas para validar la estructura interna del programa

**METODOS DE PRUEBA A NIVEL DE CLASES --** Para Prueba De Unidad

* **VERIFICACIÓN AL AZAR:** verificar los métodos de las clases en combinaciones al azar.
* **PRUEBA DE PARTICIÓN:** reduce el número de casos de prueba para validar cada clase**.**

**DISEÑO DE CASOS DE PRUEBA INTERCLASES**

* **PRUEBA DE MULTIPLES CLASES**
* **PRUEBA DERIVADA DE MODELOS DE COMPORTAMIENTO.**

**CAPITULO 24 – MÉTRICAS TÉCNICAS PARA SISTEMAS ORIENTADOS A OBJETOS**

Son métricas para el análisis cuantitativo, que permiten evaluar la calidad del diseño antes de construirlo, debido a que se necesitan criterios objetivos.

**CRATERÍSITICAS DE LAS METRICAS O.O.**

* **LOCALIZACIÓN:** indica cómo la información está en el programa
* **ENCAPSULACIÓN**: empaquetamiento de una colección de elementos
* **OCULTAMIENTO:** oculta detalles operaciones
* **HERENCIA:** mecanismo que permite propagar las responsabilidades a otro objeto
* **ABSTRACCIÓN:** mecanismo que permite concentrarse en los detalles esenciales

**MËTRICAS PARA EL MODELO DE DISEÑO O.O.**

* **TAMAÑO:**
  + **LONGITUDES:** medida de una cadena de elementos de diseño interconectados
  + **POBLACIÓN:** recuento de entidades OO.
  + **MEDIDAS DE VOLUMEN:** ídem población, pero se realizan dinámicamente
  + **MÉTRICAS DE FUNCIONALIDAD:** proporcionan una indicación indirecta del valor agregado al cliente por una aplicación OO.
* **COHESIÓN:** medida de especialización con la que cuenta un objeto en el sistema
* **ACOPLAMIENTO:** conexiones físicas entre elementos de diseño OO.
* **COMPLEJIDAD:** examina como se interrelacionan las clases
* **INTEGRIDAD:** conjunto de características, que se comparan con la abstracción o componente
* **ORIGINALIDAD:** es el grado en que una operación es atómica
* **SIMILITUD:** es el grado en que dos o más clases son similares
* **SUFICIENCIA:** es el grado en que una abstracción posee los rasgos mínimos
* **VOLATILIDAD:** mide la probabilidad de que el cambio ocurra

**MÉTRICAS ORIENTADAS A CLASES**

**MÉTRICAS CK**

* **Árbol de profundidad de herencia (APH):** se define como la máxima longitud del nodo a la raíz del árbol
* **Acoplamiento entre clases objeto (ACO):** es el nro. De colaboraciones listada para una clase, en al tarjeta CRC
* **Carencia de cohesión de métodos (CCM):** es el nro. De métodos que accede a uno o mas de los mismos atributos
* **Métodos ponderados por clase (MPC):** sumatoria de los métodos de complejidad para una clase
* **Número de descendientes (NDD**): es el nro. De subclases inmediatamente subordinadas a una superclase
* **Respuesta para una clase (RPC):** es el nro. De métodos del conjunto respuesta.

**METRICAS DE LORENZ Y KIDD**

* **Tamaño de clase (TC):** puede medirse según el nro. de atributos y el total de operaciones.
* **Nro. de operaciones redefinidas para una subclase (NOR):** redefiniciones de operaciones de una superclases para la subclase
* **Nro. De operaciones añadidas por una subclase (NOA):** las subclases se especializan añadiendo operaciones y atributos privados
* **Índice de especialización:** grado de especializaciones para cada subclase de un sistema OO.

**MÉTRICAS MDOO**

* Factor de herencia de métodos: es el grado en que las clases hacen uso de la herencia
* Factor de acoplamiento: es el grado es que se conectan los elementos de diseño
* Factor de polimorfismo: la medida indirecta de la cantidad relativa de ligadura dinámica de un sistema

**MÉTRICAS ORIENTADAS A OPERACIONES**

* Tamaño medio de operaciones (TMO): el nro. De mensajes enviados por operación proporciona una alternativa para el tamaño de operación
* Complejidad de operación (CO): mantener el CO tan bajo como sea posible
* Nro. De parámetros de media por operación (NP media): debe mantenerse tan baja como sea posible

**MÉTRICAS PARA PRUEBAS O.O**

* **ENCAPSULACIÓN:**
  + **CARENCIA DE COHESIÓN DE MÉTODOS (CCM):** indica que los métodos no tengan efectos colaterales
  + **PORCENTAJE PUBLICO Y PRIVADO (PPP):** índica el nro. de atributos que son públicos para una clase
  + **ACCESO PÚBLICO A DATOS MIEMBRO (APD):** índice le nro. de clases o métodos que pueden acceder a otras clases
* **HERENCIA:**
  + **NRO.DE CLASES RAÍZ (NCR):** índice el nro. de superclases que se deban comprobar. Cuanto mayor sea el valor, mas esfuerzo de comprobación.
  + **NRO. DE PADRES DIRECTORIO (NPD)**: Indica la herencia múltiple (si >1)
  + **NRO. DE DESCENDIENTES (NDD) Y ARBOL DE PROFUNDIDAD DE HERENCIA (APH):** indica el nro. de métodos de la superclase que se prueban en la subclase

**MÉTRICAS PARA PROYECTOS O.O**

* **Nro. de escenarios (NE):** indica el tamaño del programa
* **Nro. de clases clave (NCC):** indica el dominio del negocio del problema
* **Nro. De subsistemas:** indica la asignación de recursos ,planificación y esfuerzo de integración global

**CAPÍTULO 25 - GESTIÓN DE RIESGO**

Son una serie de pasos que ayudan a gestionar y comprender la incertidumbre

* **Incertidumbre:** el riego puede ocurrir o no
* **Pérdida**: las consecuencias o perdidas si ocurre un riesgo

**TIPOS DE RIESGOS**

* **Del proyecto:** amenazan el plan del proyecto
* **Técnicos:** amenazan la calidad y actualidad del software a producir
* **De Negocios**: amenazan la viabilidad del proyecto o negocio. Los 5 mayores riesgos son
  + **Riesgo de marcado:** la construcción de un producto que nadie quiere
  + **Riego de estrategia:** la construcción de un producto que no encaja con la estrategia comercial
  + **Riesgo de ventas:** la construcción de un producto que no se sabe como vender
  + **Riesgo administrativo:** la perdida de apoyo de los altos ejecutivos de la empresa
  + **Riesgo presupuestal:** la pérdida del presupuesto o personal asignado
* **Riesgos conocidos:** son aquellos que pueden descubrirse después de una evaluación del plan del proyecto, del entorno de negocios y técnico donde se desarrolla el proyecto
* **Riesgos predecibles**: Se extrapolan de la experiencia de proyectos previos
* **Riesgos impredecibles**: Son difíciles de descubrir.

**IDENTIFICACIÓN DE RIESGOS**

**RIESGOS GENÉRICOS**: amenaza potencial de todo proyecto de software

**RIESGOS ESPECÍFICOS:** son riesgos identificables si se tiene suficiente conocimiento de la tecnología, el personal y el entorno de software.

**SUBCATEGORÍAS GENÉRICAS**

* Características del cliente
* Definición del proceso
* Entorno de desarrollo
* Impacto del negocio
* Tamaño del producto
* Tamaño y experiencia del a plantilla

**COMPONENTES Y CONTROLADORES DEL RIESGO**

* **RIESGO DE DESEMPEÑO:** Grado de incertidumbre de que el software satisfaga los requisitos del cliente y el uso que pretenda darle.
* **RIESGO DE COSTO**: grado de incertidumbre de que se mantenga el presupuesto
* **RIESGO DE SOPORTE:** grado de incertidumbre de que el software sea fácil de corregir, adaptar y mejorar
* **RIESGO DE CALENDARIZACIÓN:** grado de incertidumbre de que se mantenga la calendarización del proyecto y de que el producto se entregue a tiempo

**PROYECCIÓN EL RIESGO**

Trata de clasificar la probabilidad de que el riesgo sea real y las consecuencias de ese riesgo. Establece

* Una descripción del riesgo, para que no haya malas interpretaciones
* Una escala que refleje la probabilidad
* Las consecuencias del riesgo
* El impacto del riesgo (catastrófico, critico, marginal y despreciable)

**PLAN RSGR (REDUCCIÓN, SUPERVISIÓN, GESTIÓN DEL RIESGO)**

Documenta el trabajo realizado como parte del análisis del riego en una hoja de riesgo teniendo

* **Reducción:** actividad que evita el problema
* **Supervisión:** actividad de seguimiento con tres objetivos:
  + Valorar si los riesgos ocurren
  + Asegurar los pasos para evitar el riesgo
  + Recopilar información para futuros análisis
* **Gestión:** el gestor emplea los resultados de la reducción y supervisión en el proyecto.

**CAPÍTULO 26 - GESTIÓN DE CALIDAD**

**Control de la variación:** es al clave para productos de alta calidad. Intenta controlar la variación en el proceso genérico en el que se aplica y la calidad del trabajo

* **Calidad:** características mensurables (cosas que se comparan para conocer estándares)
* **Calidad de Diseño:** son las especificaciones de diseño
* **Calidad de concordancia:** grado de aplicación de las especificaciones de diseño

**Control de calidad:** bucle de retroalimentación con el proceso de creación del producto de trabajo

**Garantía de calidad:** conjunto de funciones complejas (de auditoria e información) que evalúan la efectividad del producto-

**Costo de calidad:** son los costos para conseguir la calidad deseada (costos de prevención, evaluación y fallas, tanto internas como externas)

**GARANTÍA DE LA CALIDAD DE SOFTWARE (SQA)**

Estándares de desarrollo, características, requisitos funcionales y de desempeño que se esperan conseguir en cualquier software.

**ACTIVIDADES SQA**

* Prepara un plan de SQA para el proyecto
* Participar en el desarrollo de la descripción del proceso de software
* Revisar las actividades de ingeniería de software
* Auditar productos de trabajo de software seleccionados
* Garantizar que las desviaciones en el software y el producto de trabajo están documentadas
* Registrar cualquier falta de ajuste e informarlo al gestor ejecutivo

**REVISIONES TÉCNICAS FORMALES (RTF):** es un medio para descubrir errores durante el proceso de software, para que no se conviertan e defectos después.

* **Error:** es un problema de calidad descubierto antes de entregar el software al cliente
* **Defecto:** es un problema de calidad descubierto después de entregar el software al cliente

**OBJETIVOS**

1. Descubrir errores en la función, lógica o implementación del software.
2. Verificar que el software satisfaga los requisitos.
3. Garantizar que el software concuerde con los estándares predefinidos.
4. Lograr el software desarrollado de manera uniforme.
5. Hacer proyectos más manejables.

**DIRECTRICES DE LA REVISIÓN**

1. Revisar al producto, no al productos
2. Establecer una agenda y respetarla
3. Limitar el debate y la impugnación
4. Enunciar áreas de problema, pero no se intente resolver todos los que se hayan señalado
5. Tomar notas
6. Limitar el nro. De participantes e insistir en la preparación anticipada
7. Desarrollar una lista de verificación para cada producto que tenga la probabilidad de ser revisado
8. Asignar recursos y programar las RTF
9. Realizar un entrenamiento significativo de todos los revisores
10. Analizar las revisiones previas

**GARANTÍA DE LA CALIDAD ESTADÍSTICA DE SOFTWARE:** Consiste en

* Recopilar/clasificar la información de cada defecto
* Determinar la causa del mismo
* Aislar el 20% mediante el principio de pareto
* Y corregir los problemas

**SEIS SIGMA**: Estrategia del aseguramiento de la calidad estadística industria.

* **Definir, Medir, Analizar, Mejorar y Controlar: método DMAMC**
* **Definir, Medir, Analizar, Diseñar y Verificar: método DMADV**
* Definir los requisitos del cliente, entregables y metas del proyecto por métodos bien definidos del a comunicación con el cliente
* Medir el proceso existente y su salida para determinar el desempeño de calidad actual
* Analizar las métricas de defecto y determinar las causas poco vitales
* Mejorar el proceso eliminando las causas originales de los defectos
* Controlar el proceso para garantizar que el trabajo futuro no vuelva a introducir las causas de defectos
* Diseñar el proceso para evaluar las causas de los defectos y satisfacer los requisitos del cliente
* Verificar el modelo de proceso para evitar los defectos y satisfacer los requisitos del cliente

**FIABILIDAD DEL SOFTWARE**: es la probabilidad de que una operación esté libre de fallas en un entorno específico durante un cierto tiempo.

**ESTÁNDARES DE CALIDAD ISO 9000**

**Sistema De Garantía De Calidad**: Es un sistema que ayuda a garantizar que los productos satisfagan las expectativas del cliente. La que se aplica a la ingeniería en software es la ISO 9001:2000. Contiene 20 requisitos para generar un sistema eficiente de garantía de calidad.

**PLAN DE SQA**

Es un plan para proporcionar la garantía de calidad. Se centra en

1. Propósito y ámbito del plan
2. Descripción de los productos de trabajo
3. Estándares y practicas aplicables
4. Acciones y tareas SQA
5. Herramientas y métodos que soportan las acciones y tareas SQA
6. Métodos para ensamblar, guardar y mantener los registros
7. Papeles y responsabilidades de la organización

**CAPÍTULO 27 - GESTIÓN DEL CAMBIO**

**GESTIÓN DE LA CONFIGURACIÓN DE SOFTWARE** Son los elementos de la información producida (programas, productos de trabajo, datos) más el cambio de los mismos.

El cambio se genera por nuevas condiciones en el negocio o mercado, necesidades del cliente, reorganización, crecimiento o reducción del negocio, restricciones presupuestales o de calendarización.

Las metas del gestor de configuración es garantizar que se siguen los procedimientos para crear,cambiar y probar el código,asi como la información del proyecto.

**ELEMENTOS DE UN SISTEMA DE GESTION DE CONFIGURACIÓN**

* **Elementos De Componenetes:** conjunto de herramientas acopladas dento de un sistema de gestión de archivos que permiten el acceso y gestión de cada elemento de configuración
* **Elementos De Proceso:** conjunto de procedimientos y tareas que definen un enfoque eficaz para gestionar el cambio (y actividades relacionadas) para todos los participantes
* **Elementos De Contrucción:** conjunto de herramientas que automatizan la contruccion del software, al asegurar que se ensamblaron un conjunto adecuado de componenetes válidos
* **Elementos Humanos:** conjunto de herramientan y caracterisicas de proceso que requiere el equipo de software para trabajar de forma eficaz

**LINEA BASE:** Es un hito en el desarrollo de software. Demuestra que uno o más elementos de la configuración se aprobaron como consecuencia de una revisión técnica formal.

**ELEMENTOS DE LA CONFICURACION DEL SOFTWARE (ECS):** Es información que se crea en el proceso de la ingeniería en software, Es un documento. Representa un caso de pureba completo o componenete.

**EL DEPÓSITO DE ECS:** Es un conjunto de mecanismos y estrucutras de datos que permiten que el equipo de software maneje el cambio. Tiene como funciones:

* Comparitir información
* Estandaricación de los documentos
* Fortalecimiento de la metodología
* Integridad de datos
* Integración de los datos
* Integración de herramientas

**CARACTERISTICAS DE LA GCS**

* Versiones (guarda las versiones)
* Gestión Del A Configuración (guarda las configuraciones, como los hitos y liberaciones)
* Gestión Del Seguimiento De La Dependencia Y El Cambio (guarda relaciones objetos)
* Seguimiento De Requisitos (sigue lso componentes y entregables de diseño y contrucción)
* Rutas De Auditoría (sobre Who,why,when los cambios)

**EL PROCESO DE GCS**

* **IDENTIFICACION:** identifica todos los elementos que definene la configuración del software
* **CONTROL DE CAMBIO:**  gestiona los cambios de uno o mas elementos ya identificados. Es un conjunto de procedimientos que asegura la calidad y consistencia según los cambios que se realizan en los objetos de configuración
* **CONTROL DE VERSION:** facilita la contruccion de diferentes versiones de una aplicación. Es un conjunto de porcedimientos para gestionar el empleo de objetos.
* **AUDITORÍA DE CONFIGURACIÓN:** es una actividad SQA que ayuda a garantizar que la calidad se conserva según se realizen los cambios
* **REPORTE:** son informes de estado que ofrecen información sobre cada cambio cambio a quienes los necesiten