# UNIDAD 1

## Ingeniería de software

Disciplina de la ingeniera cuya meta es el desarrollo costeable de sistemas de software, este es abstracto, intangible y fácil de modificar. La ingeniería de software comprende todos los aspectos de la producción de software, desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de este después que se utiliza.

### Retos que afronta la Ingeniería del software

- De **Heterogeneidad** (consiste en desarrollar técnicas para construir software que pueda operar como un sistema distribuido en redes con distintos tipos de computadoras, con sistema en otros lenguajes, etc)
- De la **Entrega** (reducir los tiempos de entrega para sistemas grandes y complejos sin comprometer la calidad del sistema)
- De la **Confianza** (desarrollar técnicas que demuestren que los usuarios pueden confiar en el software)

### Crisis del Software

La noción de la ingeniería de software surgió en 1968 debido a la crisis de software **causada** por la introducción de nuevas computadoras basadas en circuitos integrados que provoco la posibilidad de desarrollar software más grande y complejos, esto llevo a que los proyectos se atrasaban, otros se cancelaban, se sobrepasaban los presupuestos, como **consecuencia** se generaban software de mala calidad y desempeño pobre que requerían intensas actividades de mantenimiento.
Entonces fue evidente que para crear un software de esta magnitud tomar un enfoque informal no era adecuado, se necesitaban nuevas técnicas y métodos para controlar la complejidad de estos sistemas.

## Software

Es un set de programas, archivos de configuración y la documentación asociada
Tipos básicos de Software:

- **System Software** (S.O)
- **Utilitarios** (Winrar)
- **Software de Aplicación** (Office)

Tipos de Productos:

- **Productos Genéricos**: sistemas producidos por una organización y que se venden al mercado abierto a cualquier cliente, la especificación es controlada por quien lo desarrolla
- **Personalizados o A Medida**: son sistemas requeridos por un cliente en particular, la especificación es desarrollada y controlada por la organización que compra el software

### Buen Software

Los atributos reflejan la calidad del software, no están directamente asociados con lo que el software hace, más bien reflejan su comportamiento durante su ejecución y en la estructura y organización del programa fuente y en la documentación asociada.

- **Mantenibilidad** (debe escribirse de tal forma que pueda evolucionar para cumplir necesidades de cambio)
- **Confiabilidad** (no debe causar daños físicos o económicos en el caso de una falla del sistema)
- **Eficiencia** (no debe malgastar los recursos del sistema, como memoria o procesamiento)
- **Usabilidad** (debe ser fácil de usar por el usuario, con interfaz apropiada y su documentación)

### Software VS manufactura

El software no se gasta, no está gobernado por las leyes de la física, es menos predecible…

## Proceso de Software

Es un conjunto estructurado de actividades que la gente usa para desarrollar y mantener sistema de software, estas actividades varían dependiendo de la organización y el tipo de sistema, son llevadas a cabo por los ingenieros de software.

El desarrollo de sistema es tan complejo y es tan impredecible, que debe ser gestionado bajo un modelo empírico (asume procesos complicados con variables cambiantes, la administración y control es mediante inspecciones frecuentes y adaptaciones) de control de procesos. Debe ser explícitamente modelado si va a ser administrado.

Un proceso de desarrollo permite usarse con una variedad de ciclos de vida.

El proceso de desarrollo se elegirá de acuerdo al tipo de sistema que se vaya a desarrollar, el uso inadecuado del proceso puede reducir la calidad o la utilidad del producto de software a desarrollar o incrementar los costos de desarrollo.

Existen 4 actividades fundamentales comunes a todos los procesos:

- **Especificación de software:** los clientes e ingenieros definen el software a producir y las restricciones
- **Desarrollo de software:** el software se diseña y programa
- **Validad de software:** el software se válida para asegurar que es lo que el cliente realmente quiere
- **Evolución de software:** el software se módica para adaptarlo a los cambios requeridos por el cliente y el mercado

### Ciclos de vida (Modelos de proceso)

Es una descripción simplificada de un proceso de software que presenta una visión de ese proceso, son una serie de pasos a través de los cuales el producto progresa, los modelos especifican las fases del proceso (req, diseño…) y su orden, grafica una descripción del proceso desde una perspectiva particular.

Sirven para proveer una guía para la administración de proyectos.

Son una herramienta para planificar y monitorear proyectos, son muy abstractos.

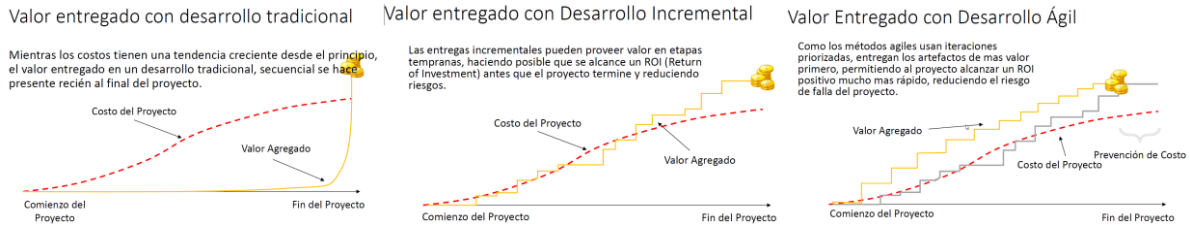
Pueden incluir Actividades, Productos y Papel de las personas involucradas.

Agilizan el proyecto, mejora la calidad, minimiza costos, minimiza riesgos, mejora la relación con el cliente, etc.

La elección del modelo depende de factores como riesgos técnicos, riesgos de administración, ciclo de tiempo requerido, tamaño del equipo, etc.

Tipos de Ciclo de Vida básicos:

- **Secuencial (Tradicional):** ha fallado en muchos proyectos grandes, una actividad no inicia hasta que ha terminado la anterior.
  Por ejemplo el **Cascada**: ordena rigurosamente las etapas del ciclo de vida del software, para que inicie una debe tener su anterior, cualquier error de diseño detectado en etapas de prueba conduce necesariamente el rediseño y nueva programación del código afectado. Se tiene todo bien organizado y no se mezclan las fases, es bueno para proyectos rígidos y con buena especificación de requerimientos. Sin resultados o mejoras visibles y rara vez el cliente va a establecer al principio todos los requerimientos.
- **Iterativo/Incremental:** hacer algo una y otra vez, son la tendencia de hoy. Todos los modelos recursivos son iterativos, pero no al revés. Por ejemplo **Iterativo:** creado en respuesta a las debilidades del modelo tradicional de cascada, donde se saca ventaja de lo aprendido a lo largo del desarrollo anterior incrementando entregables. Permite mejorar y ajustar el proceso. Desarrollo lento por la necesidad de la retroalimentación del cliente, no es bueno para aquellos proyectos grandes en recursos y largos en el tiempo.
- **Recursivo**: significa que se comienza con algo en forma completa, como una subrutina que se llama a sí misma en un ciclo completo que comienza nuevamente. Por ejemplo el **Espiral**



La aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo de software y sus productos para suministrar información relevante a tiempo, así el líder de proyecto junto con el empleo de estas técnicas mejorará el proceso y sus productos. Su dominio se divide en métricas de Proyecto, de Proceso o de Producto.

- **Métricas de Proceso:** mayor enfoque sobre la calidad lograda como consecuencia del proceso repetible o administrado. Se recompilan en el curso de todos los proyectos y durante largos periodos.
  Se mide el proceso para conocer y mejorar sus costos, disminuir el tiempo de construcción, mejorar la performance, etc. Su propósito es proporcionar un conjunto de indicadores de proceso que conduzcan a la mejoría de los procesos de software de largo plazo. Por ejemplo defectos que detectan y reportan los usuarios.
  Medidas directas del proceso: costo, esferzo…
- **Métricas de Proyecto:** Esfuerzo/Tiempo por tarea, Errores no cubiertos por hora de revisión, Fechas de entrega reales vs programadas. Su objetivo es minimizar el tiempo de desarrollo (disminuyendo riesgos, problemas potenciales) y valorar la calidad del producto sobre una base actual.
  Se mide el proyecto porque un proyecto debería ser entregado con las capacidades funcionales y no requeridas por el cliente, las restricciones impuestas, el presupuesto y el tiempo planificado.
  Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.
- **Métricas de Producto:** pueden ser Dinámicas (recogidas por las mediciones hechas en un programa en ejecución) o Estáticas (recogidas por las mediciones hechas en las representaciones del sistema como diseño, el programa o documentación)
  Se miden para garantizar que los artefactos producidos (componentes y modelos) cumplan con los requerimientos del cliente, estén libres de errores, cumplan con los criterios de calidad.
  Medidas directas del producto: LOC, cantidad de defectos…

Medidas indirectas: funcionalidad, calidad, complejidad, eficiencia, etc.

## Buenas métricas

- Simples y calculables: fácil aprender a derivar la métrica y fácil de calcular
- Empírica e intuitivamente persuasivas para el ingeniero
- Consistentes y efectivas: sin ambigüedad
- Consistentes en el uso de unidades y dimensiones
- Independiente del lenguaje de programación
- Dar retroalimentación efectiva.

## Métricas básicas para un Proyecto

Métricas apropiadas para el nivel apropiado:

- Desarrollador: esfuerzo, numero de defectos encontrados en revisión por pares, duración estimada y actual de una tarea.
- Equipo de desarrollador: tamaño del producto, numero de tareas planificadas y completadas, % de test ejecutados, numero de defectos encontrados, distribución de esfuerzo
- Organización: tiempo calendario, performance actual y planificada del esfuerzo, performance actual y planificada del presupuesto, defectos en reléase, precisión de estimaciones de Schedule y esfuerzo.

Métricas básicas:

- Tamaño como contar CU, SP, reportes, LOC
- Esfuerzo como horas-hombre
- Tiempo como calendario en días
- Defectos.

Los proyectos se atrasan en calendario pero se recuperan en esfuerzo.
Se seleccionan métricas que provean información para que las personas puedan tomar mejores decisiones, las métricas dan visibilidad de los problemas y las personas resuelven los problemas no las métricas

## Indicadores

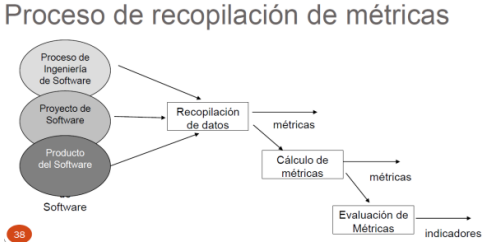
Ellos nos dan las tendencias. Principales (sugieren tendencias o eventos futuros) o Secundarios (proveen información sobre las salidas)

- Estratégicos: permiten tener una visión profunda de la eficiencia del proceso, que los gestores evalúen que funciona y que no

- Tácticos: permiten evaluar el estado del proyecto, seguir la pista de los riesgos potenciales, detectar áreas de problemas antes que se conviertan críticas, ajustar el flujo y las tareas de trabajo.

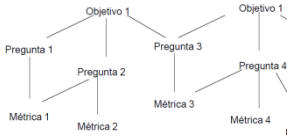
## Eficiencia de un Proceso de Software

Se mide de forma indirecta y se incluyen medidas de: errores detectados antes de la entrega del software, errores detectados e informados por el usuario, esfuerzo humano y tiempo consumido, ajuste con la planificación.




## GOAL QUESTION METRIC (GQM – OPM - Objetivo – Pregunta – Métrica)

Técnica para identificar métricas significativas aplicables en cualquier parte del proceso de software.
Se basa en que para que una organización pueda medir de manera útil debe primero especificar sus objetivos y los del proyecto, luego buscar los datos que definen esos objetivos y finalmente proveer un marco para interpretar los datos con respecto a los objetivos establecidos.
Establecer un objetivo de medición, definir preguntas que deben responderse para alcanzar el objetivo, identificar métricas bien formuladas que ayuden a responder esas preguntas.
El modelo GQM tiene una estructura jerárquica que comienza con un objetivo (nivel conceptual), este se refina en varias preguntas (nivel operacional), y cada pregunta se refina en métricas (nivel cuantitativo)
Paradigma:

# UNIDAD 3

## Auditoria

El aseguramiento de calidad de software (ACS) implica revisar y auditar los productos y actividades de software para verificar que cumplen con los procedimientos y estándares correspondientes, como así también proveer al gerente de proyecto y otros gerentes involucrados el resultado de estas revisiones y auditorias.

### Auditoria

Examen sistemático e independiente para determinar si las actividades de calidad y los resultados cumplen con lo planeado y si lo planeado se implementó efectivamente y es adecuado para alcanzar los objetivos.

Proceso de evaluación para determinar el grado de cumplimiento con requerimientos preestablecidos (estándares, criterios, etc). El resultado es la opinión sobre el grado de cumplimiento.

***Son un instrumento fundamental para el Aseguramiento de Calidad en el Software.***

**Auditoria vs Revisiones**: la revisión es un proceso o reunión durante la cual se presenta un producto de software a los miembros del proyecto, gerentes, clientes, etc con el fin de obtener sus comentarios y aprobaciones.

**Objetivos:**

- Realizar controles apropiados del software y el proceso de desarrollo
- Asegurar el cumplimento de los estándares y procedimiento para el software y el proceso
- Asegurar que los defectos en el producto, proceso o estándares sean informados a la gerencia

**Beneficios**

- Porque se da una opinión objetiva e independiente
- Permite identificar áreas de insatisfacción potencial del cliente
- Asegura al cliente que estamos cumpliendo con nuestras expectativas
- Permite identificar oportunidades de mejora

SON un instrumento fundamental para asegurar la Calidad en el Software

**Métricas de Auditoria:**

- Esfuerzo por auditora
- Cantidad de desviaciones
- Duración de auditoria

### Auditoria Informática

Proceso de recoger, agrupar y evaluar evidencia para determinar si un sistema informático, mantiene la integridad de los datos, lleva a cabo eficazmente los fines de la organización, utiliza eficientemente los recursos y cumple con las leyes y regulaciones establecidas.

Evaluación independiente de los productos o procesos de software para asegurar el cumplimiento con estándares, lineamientos, especificaciones y procedimientos, basada en un criterio objetivo incluyendo documentación que especifique: los productos a ser desarrollados, el proceso para desarrollarlos, como debería medirse el cumplimiento con estándares o lineamientos.

Los resultados de las auditorias solicitando acciones correctivas conllevan a la mejora del proceso y el producto.

Auditoría Interna: realizada con recursos materiales y personas que pertenecen a la empresa auditada, los empleados deberán ser remunerados.

Auditoría Externa: realizada por personas afines a la empresa auditada, es siempre remunerada

### Beneficios de la Auditoria de Calidad de software

- Permite evaluar el cumplimiento del proceso de desarrollo
- Nos da una opinión objetiva e independiente
- Permite identificar áreas de insatisfacción potencial del cliente
- Permite identificar oportunidades de mejora
- Asegurar al cliente que estamos cumpliendo con las expectativas
- Busca asegurar que los defectos en el producto, proceso o estándares sean informados a la gerencia.
- Permiten determinar la implementación efectiva del proceso de desarrollo organizacional, del proceso de desarrollo del proyecto, de las actividades de soporte
- Da mayor visibilidad a la gerencia sobre los proceso de trabajo
- Los resultados de las auditorias solicitando acciones correctivas conllevan a la mejora del proceso y el producto.

## Tipos de auditorías al proceso de desarrollo de software:

- Auditoria de Proyecto (valida el cumplimiento del proceso de desarrollo): se llevan a acabo de acuerdo a las PACS (plan de aseguramiento de calidad de software), las PACS deberían indicar quien es el responsable de realizar estas auditorías. Se lleva a cabo durante la ejecución del proyecto de desarrollo de software.
  Incluye las inspecciones de software y las revisiones de documentación de diseño y prueba
  Objetivo: verificar la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo
- Auditoria de Configuración Funcional (valida que el producto cumpla con los req): compara el software que se ha construido con los req especificados en la ERS. Se realiza antes de entregar el software al cliente y verificar que se cumplan todos los requerimientos.
  Objetivo: asegurar que el código implementa los requerimientos descriptos en la ERS
- Auditoria de Configuración Física (valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe): compara el código con la documentación de soporte
  Objetivo: asegurar que la documentación que se entregara es consistente y describe correctamente al código. Se lleva a cabo antes de entregar el software al cliente.
  Las PACS deberían indicar quien es el responsable de realizar estas auditorías.

## Responsables (Roles)

- Gerente de la SQA: prepara el plan de auditorías, calcula su costo, asigna recursos, responsable de resolver las no-conformidades
- Auditor: acuerda la fecha de la auditoria, comunica el alcance de la autoría, recolecta y analiza evidencia objetiva que es relevante y suficiente, realiza la auditoria, prepara el reporte, realiza el seguimiento de los planes de acción acordados con el auditado.
- Auditado: acuerda la fecha, participa de la auditoria, proporciona evidencia, contesta el reporte, propone el plan de acción para deficiencias, comunica el cumplimiento del plan de acción

## Proceso de auditaría


Solicitar la auditoria, acordar fecha y lugar, acordar participantes, alcance y objetivo de la auditoria, solicitar documentación, preparar la documentación requerida para la audición.

Compuesta por: Reunión de apertura (Acordar horario, formalismo, planteo de inquietudes) y Ejecución (preguntas y respuestas, solicitar evidencias, checklist, acordar fecha de envió de reporte)

Compuesta por: Evaluación de los resultados, Reunión de cierre (presenta el reporte preliminar, explica resultados, acuerdan resultados, ponen fecha de reporte final), Entrega de reporte final

Analizar desviaciones, preparar el plan de acción, aprobarlo, seguimiento de acción, cerrar desviaciones, informar su cierre

## Herramientas

Cuestionario general inicial, checklist, monitoreo, estándares, herramientas automatizadas.

### *Checklist de Auditoria*

Permite documentar la información obtenida como resultado de la auditoria.
Contenido: fecha de autoría, lista de auditados (con rol), nombre del auditor, nombre del proyecto, fase actual del proyecto, objetivo y alcance de la auditoria, lista de preguntas (pueden ser ya armadas o estándares, checklist abiertas, checklist de atributos SI/NO, checklist variables %)

## ROL del auditor: (para asegurar el éxito de la auditoria)

Debe tener objetividad, conocimiento técnico, conocimiento de auditorías. Debe tener habilidades Mecánicas (para recolección de datos), Intelectuales (para procesar información y comunicarse con otros), Emocionales (para la relación con otras personas)

- Debe escuchar y no interrumpir, observar el lenguaje corporal del auditado, manejar el suyo, tomar notas, preguntar.
- Debe lograr una relación armoniosa, evitar culpar a la gente por problemas, siempre actuar éticamente, evitar la pérdida de tiempo, se base únicamente en información objetiva.

- Utiliza muestreo al azar para obtener resultados representativos, ayudarse con un checklist durante la entrevista, evitar dar opiniones personales, siempre mirar a los ojos.

**Cuestionarios:** Técnica: comenzar con preguntas de final abierto (quien, que, como, donde), dar preguntas cortas y puntuales, finalizar con preguntas de final cerrado para clarificar conceptos.

## Reporte de la Auditoria

Un buen reporte está completo, preciso, objetivo, convincente, claro, conciso.
Contenido: id, fecha, auditor, auditados, nombre del proyecto, fase actual, **lista de resultados**, comentarios, solicitud de planes de acción.
Tipos de Listas de resultados:

- Buenas Prácticas (se desarrolló mucho mejor de lo esperado): el auditado ha establecido un sistema efectivo
- Desviaciones (requiere un plan de acción por parte del auditado): cualquier desviación que resulta en la disconformidad de un producto respecto de sus requerimientos, falta de control para satisfacer los req.
  Se registran con la forma de "Solicitud de Acción Correctiva" (SAR). Es una desviación? Debe responder: puede probarse que existe?, agrega valor al proyecto?, puede rastrearse?
- Observaciones (sobre condiciones a mejorar pero sin necesidad de un plan de acción): opinión sobre una conducción incumplida, practica a mejorar, puede resultar en una futura desviación.

**Alcance de la auditoria:**

- Minutas de reunión
- Métricas
- Seguimiento de acciones
- Control de cambios
- Administración de configuraciones

**Salida de la auditoria**

- Reporte de la auditoria
- Acciones correctivas

## Calidad y Modelos de mejora

**Calidad:** todos los aspectos y características de un producto o servicio que se relacionan con su habilidad de alcanzar las necesidades manifiestas

- La calidad no se inyecta, ni se compra, debe ser embebida, las personas son la clave para lograrlo (capacitación).
- Se necesita soporte gerencial
- El a seguramente de calidad debe planificarse.
- El aumento de las pruebas no aumenta la calidad.

El proceso es el único factor <controlable> al mejorar la calidad del software y su rendimiento como organización.

### Administración de la Calidad del Software

Implica la definición de estándares y procesos de calidad apropiados y asegurar que los mismos sean respetados.
La Administración de calidad debería estar separada de la Administración de proyectos para asegurar independencia.
Reportes generados por el GAP (grupo de aseguramiento de calidad).
**Estándares**: encapsulamiento de las mejores prácticas, son la clave para la administración de calidad efectiva, pueden ser internacionales, nacionales, organizacionales o de proyecto. Si no están soportados por herramientas automatizadas, a menudo se debe realizar trabajo manual, tedioso, para mantener la documentación.

- Estándares de Producto: características que todos los componentes deberían tener, Ej: estilo de programación común
- Estándares de Proceso: como deberían ser implementados los procesos de software

**Actividades**:

- Aseguramiento de calidad (establecer estándares y procedimientos de calidad)
- Planificación de calidad (selección de procedimientos y estándares para un proyecto y modificar si fuera necesario)
- Control de calidad (asegurar que estos sean respetados por el equipo de desarrollo)

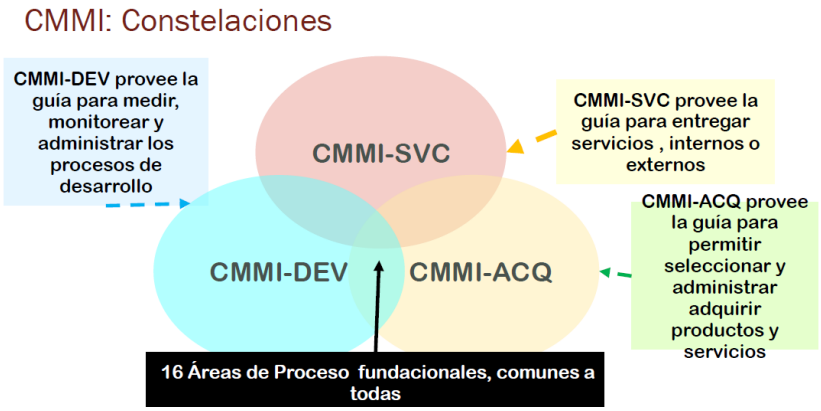
**Calidad de proceso:**

- Definir procesos estándares como: como debería conducirse revisiones, como debería realizarse la administración de configuración, etc.
- Monitorear el proceso de desarrollo para asegurar que los estándares sean respetados.
- Reportar en el proceso a la Administración de proyectos y al responsable del software.

### Modelos de Mejora

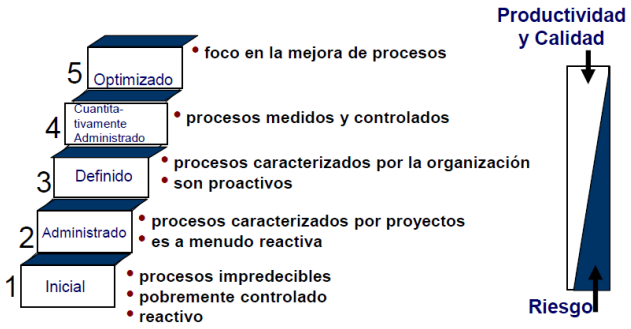
- Mejora de Procesos: mejora la comunicación, se reducen los defectos en forma temprana, reducción de re-trabajo, reducción costos de desarrollo y reducción de mantenimiento…
    - SPICE
    - IDEAL (Initiating, Diagnosing, Establishing, Acting, Learning)

### CMMI 1.3 2010 – Un modelo de calidad




Es la evolución del SW-CMM. No es una norma, y no se “certifica”, sólo se evalúa a través de profesionales reconocidos por el SEI como Lead Appraisers.
CMMI puede representarse por:

- **Por Etapas**: 5 niveles (1 – 5), definidos por un conjunto de áreas de proceso, los niveles indican Madurez organizacional




- **Continua**: 6 niveles (0 – 5), definidos por cada área de proceso, los niveles indican Capacidad de un área de proceso

# UNIDAD 4

## Software Configuration Management (SCM)

## Administración de Configuración de Software (ACS)

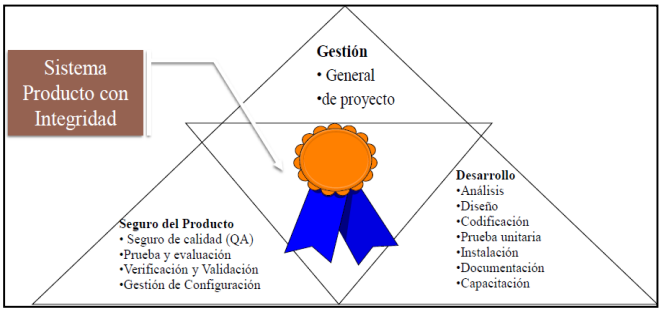
Su propósito es establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida del mismo.
Ventajas: aumenta la protección contra cambios innecesarios, mejora la visibilidad del estado del proyecto y sus componentes, disminuye el tiempo de desarrollo, aumenta la calidad….
La SCM como disciplina de gestión es trasversal a todo el proyecto durante todo su ciclo de vida.
El objetivo principal de la administración de configuraciones de software es evitar o solucionar problemas asociados con los cambios de un software, para ello se usa la Integridad del producto
Integridad:

- Satisface las necesidades del cliente
- Puede ser fácil y completamente rastreado durante su ciclo de vida (trazabilidad)
- Satisface criterios de performance (trabajar concurrentemente)
- Cumple con sus expectativas de costo

Configuración:

- Una configuración es el conjunto de todos los componentes fuentes que son compilados en un ejecutable consistente
- Son todos los componentes, documentos e información de su estructura que definen una versión determinada del producto a entregar

### Disciplinas




Cambios:

- Internos: Correctivo (defectos) o Perfectivo (mejoras)
- Externos: se generan del lado del cliente, Adaptativos (nuevos requerimientos o cambios en requerimientos)

Problemas en el manejo de componentes:

- Perdida de un componente
- Perdida de cambio (el componente que tengo no es el último)
- Doble mantenimiento
- Superposición de cambios
- Cambios no validos

Desarrollo en equipo:

- Doble actualización, Actualización múltiple, Componentes compartidos.

### Planificación de Gestión de Configuración

Se comienza en las primeras fases del proyecto, se deben definir los documentos o clases de documentos a ser administrados (documentos formales), aquellos documentos que se requerirán para el mantenimiento futuro del sistema deben especificarse como documentos administrados.
Todos los productos del proceso de software deben ser administrados (especificaciones, diseño, programas, pruebas, manuales)

## Actividades para la gestión de configuración

- Identificarla en un momento dado
- Controlar los cambios
- Registro e informes de estado
- Auditorias y revisión

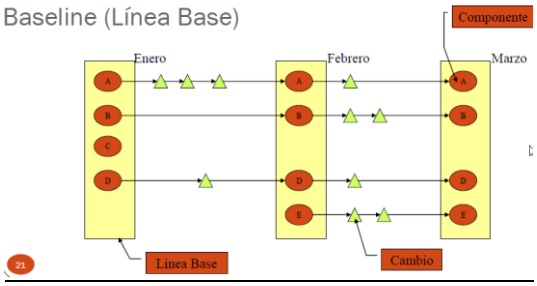
### *1 - Identificación*

Durante el proceso de planificación de la gestión de configuración, se decide exactamente qué elementos (o clases de elementos) se van a controlar.

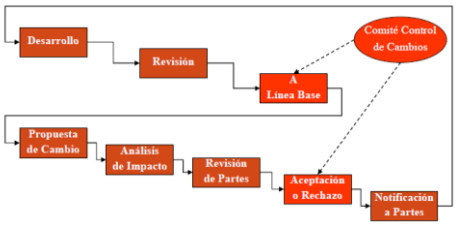
- Se logra mediante la definición de sus líneas base (baselines) y de los cambios que los mismos pueden sufrir durante la evolución del sistema.
- Las líneas base (baseline) y sus cambios especifican la evolución del sistema.

Los documentos o grupos de documentos relacionados del control de la configuración son documentos formales o elementos de configuración. Todos los documentos que son necesarios para el mantenimiento futuro del sistema deben ser controlados el sistema de control de configuración.
El esquema de asignación de nombres a los documentos debe asignar un nombre único a todos los documentos de control de la configuración y la relación entre elementos es mediante un esquema de asignación de nombres jerárquico.
Ítem de Configuración de Software (SCI):

- Documentos de diseño, código fuente, código ejecutable, etc.
- Desde el punto de vista de SCM una **baseline** es un conjunto de ítems de configuración de software revisados, corregidos y aprobados y que sirve como base para desarrollo posterior.
- Se pueden introducir cambios a las baselines después de evaluarlos y aprobarlos mediante un procedimiento formal de control de cambios.




### *2 – Control de cambios*




**Control de versiones (con herramientas automáticas):** fácil acceso a todos los componentes, reconstrucción de cualquier versión, registro de historia, no se pierden, centralización, información de resumen. Se pone bajo control de versiones: casos de prueba, código fuente, manual de usuario, requerimientos, plan de calidad, arquitectura del software, plan del configuración, gráficos, etc…

**Gestión de repositorio:** un repositorio de información conteniendo los ítems de configuración, mantiene la historia de cada ítem de configuración con sus atributos y relaciones.

### *3 - Reporte de estado de la configuración*

Se ocupa de mantener los registros de la evolución del sistema. Maneja mucha información y salidas por lo que se suele implementar dentro de procesos automáticos.
Incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida.
Responde preguntas como:

- Cuál es el estado del ítem?
- Cuál es la diferencia entre una versión y otra dada?

- Causas del reporte de problemas

### *4 – Auditorias y revisión*

¿Cómo se puede garantizar que el cambio se ha implementado con propiedad?
La respuesta es doble:

**Auditorias**

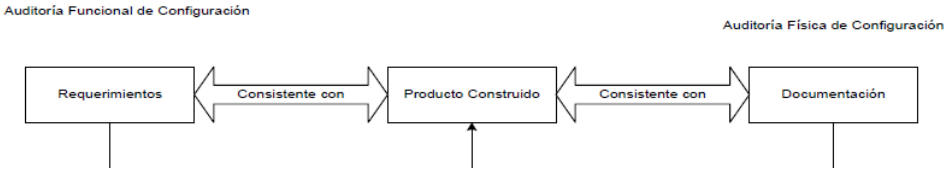
- Determinar la semejanza entre el estado actual del sistema y el establecido como línea base.
- Provee el mecanismo para establecer una línea base.

Sirve a dos procesos básicos:

- **Validación** (el problema es resuelto de manera apropiada que el usuario obtenga el producto correcto)
- **Verificación** (asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de líneas base (línea base). Todas la funciones son llevadas a cabo con éxito y los test cases tengan status “ok” )

**Tipos**

- **Auditoría Funcional de Configuración (FCA):** Evaluación independiente de los productos de software, verificando que la funcionalidad y performance reales de cada ítem de configuración sean consistentes con la especificación de **requerimientos** de software. (Rastreabilidad a través de la Matriz de Trazabilidad)
- **Auditoría Física de Configuración (PCA):** Evaluación independiente de los SCI para verificar que el software y su **documentación** son internamente consistentes y están listos para ser entregados al cliente. (Si lo que indico en el plan de configuración realmente se esté haciendo).




## Testing (Prueba de software)

Asegurar calidad:

- La calidad no puede inyectarse al final
- La calidad depende de tareas realizadas durante todo el proceso
- Detectar errores en forma temprana ahorra esfuerzo, tiempo, recursos

SQA = Pruebas de software + Calidad del proceso + Calidad del producto + Administración de configuración
Testing no es debbugear código, no es verificar que se implementen las funciones, no es demostrar que hay errores.
Es verificar que el software se ajusta a los requerimientos y además validar que las funciones se implementan correctamente.
Proceso destructivo de tratar de encontrar defectos en el código, se debe tener una actitud negativa.

Es **exitoso** aquel que encuentra muchos defectos, por lo que un desarrollo exitoso lleva a un test fallido (sin defectos)
Un programador no debería testear su propio código, ni una unidad de programación sus propios proyectos.
**Clasificación**: Invalidante, Severo, Leve, Mejora, Cosmética

Cuando parar? “Good enough”: cierta cantidad de fallas no criticas es aceptable

### Niveles de prueba

- Pruebas Unitarias: la primera etapa de la prueba, está enfocada a los componentes más pequeños del Software que se puedan probar (programas y módulos), se verifican tipos de datos inconsistentes, precisión en las funciones de cálculos, comparación entre tipos de datos, terminación de loops, correspondencias entre parámetros y argumentos…

Se realiza sombre una unidad pequeña de código, claramente definida.
Generalmente son hechos por los desarrolladores, Ej: JUnits.

- Pruebas de Integración: test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto. Se debe conectar de a poco las partes más complejas.
- Pruebas de Iteración o Sistemas: es como un test de unidad, de una unidad formada por otras unidades ya integradas.
    - Prueba de verificación de versión: su objetivo es tener una rápida visión de la estabilidad de la aplicación, antes de realizar una prueba en profundidad.
    - Prueba de Sistema: prueba cuando una aplicación está funcionando como un todo, y se busca probar que el sistema opera satisfactoriamente completamente.
- Pruebas de Aceptación: es la prueba realizada por el usuario para determinar si la aplicación se ajusta a las necesidades, tanto pruebas alfa (el usuario en el laboratorio) como beta (en el ambiente de trabajo)

## Tipos de pruebas

- Funciones de negocio
- Interfaces de usuario
- Performance
- Carga
- Estrés
- Volumen
- Configuración
- Instalación

## Pruebas de Regresión

Al concluir un ciclo de pruebas (la ejecución de la totalidad de los casos de prueba), y reemplazarse la versión del sistema sometido al mismo, debe realizarse una verificación total de la nueva versión, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los detectados.
“La única manera de estar seguro de que no existe un error es resetear todo”

Técnicas de Prueba

- Sin testeo de regresión: pueden surgir errores nuevos desconocidos en lugares donde no había anteriormente y estos nunca se solucionaran porque no se vuelven a correr en ciclos de prueba posteriores.
- Con testeo de regresión: se corren todos los test cases en cada ciclo de prueba, lo que permite detectar los nuevos problemas.

## Métodos

Caja Negra:

- **Partición de equivalencias**: identificar un conjunto de clases de prueba representativas de grandes conjuntos de otras pruebas posibles, la idea es que el producto bajo prueba se comportará de la misma manera para todos los miembros de la clase
- **Análisis de valores límites**: variante del anterior, pero en vez de seleccionar cualquier elemento como representativo de una clase de equivalencia, se seleccionan los bordes de una clase. Además se definen clases de equivalencia de salida, no solo de entrada como el anterior.
  Ej: número entre 3 y 8, probar para 2, 3, 8 y 9
- **Adivinanza de defectos**: basado en intuición y experiencia para identificar pruebas que probablemente expondrás defectos. Se lista de defectos posibles o situaciones propensas a error, desarrollo de pruebas basadas en la lista

Caja Blanca:
Utiliza la estructura de control del diseño procedural para derivar casos de prueba que:

- Garanticen que todos los caminos independientes dentro de un módulo han sido ejercitados por lo menos una vez
- Ejerciten que todas las decisiones lógicas en sus lados verdaderos y falsos.
- Ejerciten sus estructuras de datos internas para asegurar su validez.

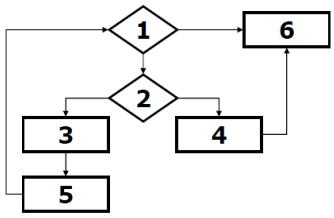
El testing de caja blanca se utiliza porque hay errores que pueden no ser detectados por testing de caja negra, como errores tipográficos, caminos lógicos que se cree que no se ejecutara, etc

- **Cobertura de secuencias:** recorrer cada uno de los posibles caminos lógicos
- **Cobertura de decisión:** ejecutar cada decisión por lo menos una vez obteniendo un resultado verdadero y uno falso.
- **Cobertura de condición:** cada condición en una decisión tenga todos los resultados posibles al menos una vez, ejercita las condiciones lógicas contenidas en un módulo de programa
- **Cobertura de loop:** se focaliza en la validez de las instrucciones de loops, se prueba saltearse completamente el bucle, solo una iteración, dos, m iteraciones donde m < n, n – 1, n, n + 1 iteraciones.
- **Cobertura de caminos básicos:** ejecutar por lo menos una vez cada instrucción del programa y cada decisión se habrá ejecutado en su lado verdadero y falso.

## Complejidad Ciclomática

Es una métrica de software que provee una medición cuantitativa de la complejidad lógica de un programa
Usada en el contexto de testing, define el número de caminos independientes en el conjunto básico y entrega un límite inferior para el número de casos necesarios para ejecutar todas las instrucciones al menos una vez. Se basa en el recuento del número de caminos lógicos individuales contenidos en un programa.
Un **camino independiente** es cualquier camino a través del programa que introduce al menos un nuevo conjunto de instrucciones o una nueva condición; en términos de un grafo de flujo, debe incluir al menos un arco no utilizado antes de definir el camino.

Cobertura de caminos básicos o cobertura de enunciados:




**Caminos básicos:**

- 1, 2, 3, 5, 1, 6.
- 1, 2, 4, 6.
- 1, 6

**Complejidad ciclomática**
**V(G) =** 3

Cualquier otro camino utilizará alguno de los

Una vez calculada la complejidad ciclomática de un fragmento de código, se puede determinar el riesgo que supone utilizando los rangos definidos en una tabla: 1 – 10 Simple sin mucho riesgo, 11 – 20 Más complejo, riesgo moderado, 21 – 50 Complejo, programa de alto riesgo, 50 No testeable, muy alto riesgo.