SUPER FINAL POST

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.

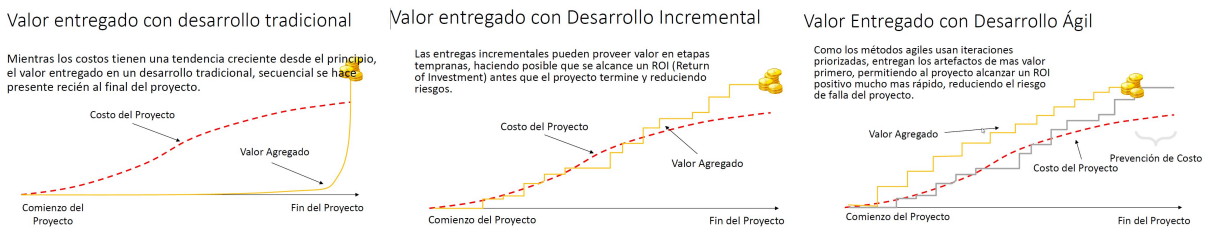
Pueden incluir Actividades, Productos y Papel de las personas involucradas.

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**



Proyecto

Un proyecto es una planificación que consiste en un conjunto de actividades que se encuentran interrelacionadas y coordinadas. La razón de un proyecto es alcanzar objetivos específicos dentro de los límites que imponen un presupuesto, calidades establecidas previamente y un lapso de tiempo previamente definido.
Surge como respuesta a una necesidad. El proyecto finaliza cuando se obtiene el resultado deseado, y se puede decir que colapsa cuando desaparece la necesidad inicial o se agotan los recursos disponibles.
Características

- **Está orientado a objetivos**
  - Los proyectos están dirigidos a obtener resultados y ello se refleja en los objetivos, indican su fin.
  - Guían al proyecto, no deben ser ambiguos, deben ser claros, alcanzables y medibles.
  - ***Los objetivos me dicen dónde empieza y donde termina el proyecto***
- **Tiene una duración limitada en el tiempo, tiene principio y fin siempre**
  - Son temporales, cuando se alcanza el objetivo, el proyecto termina.
  - Una línea de producción no es un proyecto
- **Implican tareas interrelacionadas basadas en esfuerzo y recursos**
  - Trabajo en equipo, calendarización, control, monitoreo…
- **Son únicos**

Ciclo de vida de un proyecto

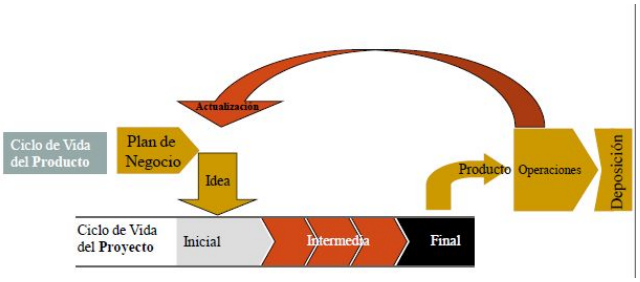
Definen para cada fase:

- Que trabajo técnico debería realizarse
- Quien debería está involucrado en cada fase
- Como controlarla y aprobarla
- Como se deben generar los entregables y como se revisa, verifica y valida cada producto

Características de la mayoría de los ciclos de vida:

- Costos y nivel de personal son bajos al principio y más alto al final, y cae cuando el proyecto termina
- El riesgo y la incertidumbre son altos al comienzo, por lo tanto la probabilidad del éxito al completar el proyecto es muy baja al comienzo.

Ciclo de vida del Producto vs Proyecto



Éxito de un proyecto

- Monitoreo y Feedback
- Tener una misión/objetivo claro
- Comunicación

Fracaso de un proyecto

Fallas al definir un problema, planificar con datos insuficientes, **el plan de proyecto no tiene seguimiento**, estimación está basada en supuestos sin consultar datos históricos, requerimientos incompletos o cambiantes, falta de recursos, falta de involucramiento del usuario.

Errores

- **Gente** (falta de motivación, personal mediocre…)
- **Proceso** (planificación insuficiente, diseño inadecuado…)
- **Producto** (exceso de requerimientos, desarrolladores meticulosos…)
- **Tecnología** (Síndrome SilverBullet ósea el equipo se aferra solo a una nueva técnica, cambio de herramientas a mitad del proyecto…)

Factores de calidad

La restricción triple, el balance de ellos afecta directamente la calidad del proyecto y es responsabilidad del Líder del proyecto balancearlos, es la única forma de negociar proyectos.

- Objetivo del proyecto (Alcance)

- Tiempo para completarlo
- Costos

Roles

Líder del proyecto

Skills: sentirse cómodo con los cambios, entender la organización y estructura, poder guiar al equipo para cumplir los objetivos

- Hard: conocimiento del producto, conocer las herramientas y técnicas de la administración de proyecto
- Soft: ser capaz de trabajar con gente, saber escuchar, liderazgo, motivación, creativo...

Equipo

Grupo de personas comprometidas en alcanzar objetivos de los cuales se sienten mutuamente responsables 7 +/- 2. Poseen diversidad de conocimientos y habilidades, pueden trabajar en conjunto efectivamente, es pequeño, tienen sentido de responsabilidad como una unidad.

Administración de proyecto

Es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto.
Objetivo: tener el trabajo echo en tiempo, con el presupuesto acordado y habiendo satisfecho todas las especificaciones o requerimientos.
Incluye:

- Identificar los requerimientos: es la primera etapa que se aborda cualquier sea la metodología a utilizar, es descubrir, explicitar, obtener el máximo de información para el conocimiento del objeto en cuestión. Un requerimiento es una característica que debe incluirse en un nuevo sistema. A través de Cuestionarios, Entrevistas, Lluvia de ideas, etc. ***Los requerimientos me dicen el alcance (límites del sistema)***
- Establecer los objetivos
- Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (Stakeholders, director, equipo, interesados...)

Etapas de un proyecto

Cumple lo Planificado	Cumple el Estándar	Acción
SI	SI	Todo va bien
SI	NO	Estudiar la desviación del estándar y ajustarlo si es necesario, disciplinar
NO	SI	Replanificar y estudiar la situación para crear medidas correctivas
NO	NO	Estudiar con cuidado. Aplicar las dos medidas anteriores.

1 - Iniciación del proyecto

Antes de iniciar un proyecto se debe

- Identificar el cliente y el mercado: identificar el dominio, si se debe cumplir con algún marco de calidad (ej CMMI) y las líneas de reporte, se dialoga con el cliente
- Estudio de factibilidad: Puede o no ser necesitado?, es un estudio para verificar si el proyecto puede realizarse o no, se hacen estudios de factibilidad técnica, económica y operativa. Se definen requerimientos iniciales
- Propuesta de proyecto: incluye el que se hará, cómo, cuándo y cuánto.
    - Resumen ejecutivo: cuales son las funciones básicas y los beneficios al cliente
    - Arquitectura del sistema: que es lo que se va a construir
    - **Plan del proyecto**: cuales son los entregables y sus fechas
    - Costos: de recursos y costos fijos.
- Negociaciones:
    - Características del producto: funcionalidades del producto, deben ser realistas y medibles
    - Cronograma: fechas de comienzo y fin, hitos, entregas...
    - Presupuesto: staff, sw, hw...
    - Estructura del equipo: cantidad de personas por rol
    - Consideraciones: propiedad del código fuente, si los riesgos se compartirán...
- Inicio del proyecto: elegirle un nombre y anunciarlo, dará el Kick-off que dará comienzo al proyecto y se deberá definir la estructura del equipo, necesidades de soporte, necesidades financieras, objetivos de calidad.

2 - Planificación del proyecto y armado del equipo

- Definición del proyecto: determinar los problemas/oportunidades, misión/objetivos (deben ser críticos, observables, distinguibles y medibles) y definir las asunciones, riesgos y las condiciones de aceptación del producto (permite marcar los límites del proyecto)
- Creación del plan: **el Plan de proyecto *es una hoja de ruta, sirve para comparar con la realidad, se documenta como va a funcionar el equipo, debe actualizarse*,** provee una herramienta de comunicación estándar a través del ciclo de vida del proyecto, documenta que es lo que hacemos, cuando, como y quien. Fija los recursos disponibles, divide el trabajo y crea el calendario de trabajo, se usa como conductor para el proyecto.

Un plan preparado (por el Gestor de proyecto) al inicio del proyecto debe utilizarse como un conductor para el proyecto, debe ser el mejor posible de acuerdo a la información disponible y evolucionara conforme el proyecto progrese y la información sea mejor y más clara.
El Plan de Proyecto responde Que hacemos? Cuando? Como? Quien?

- Durante la planificación se debe:
  - Infraestructura del proyecto: los proyectos necesitan empezar con una infraestructura básica, que puede incluir, administración de configuración, directorio para el proyecto, lista de mails, sitios web para realizar la comunicación, herramientas varias como entornos de desarrollo y testing.
  - Formación del equipo: determinar quiénes serán los miembros iniciales, los que provienen de otros proyectos y cuáles serán sus roles, también asegurar el entrenamiento.
- Producir documentos importantes (Salida): **REQB** libro de requerimientos, **SPMP** plan de proyecto, **SQAP** plan de aseguramiento de calidad, **SCMP** plan de gestión de configuración, Directorio del proyecto o **repositorio**.
  REQB + SPMP = contrato virtual entre el proyecto y cliente.

La planificación de Proyecto de Software implica:

- **Definición del alcance del proyecto:**
  - Alcance del proyecto es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas. Su cumplimiento se mide contra el **Plan de Proyecto.**
  - Alcance del producto son todas las características que pueden incluirse en un producto o servicio, su cumplimiento se mide contra la **Especificación de Requerimientos.**
- **Definición del proceso y ciclo de vida**
- **Estimación**
- **Gestión de riesgos**
- **Asignación de recursos**
- **Programación de proyectos**
- **Monitoreo y control (Definición de controles)**
- **Métricas de software**

3 – Monitoreo y Control

El Monitoreo y Control es uno de los factores para el éxito de un proyecto junto a tener una misión/objetivo claro y la comunicación.

Objetivo

Determinar si el proyecto está bajo control

- Para ello se debe evaluar si se está alcanzando los hitos del proyecto (a tiempo, con los recursos, con el nivel de calidad, continua siendo aceptable económicamente)

Fuera de control

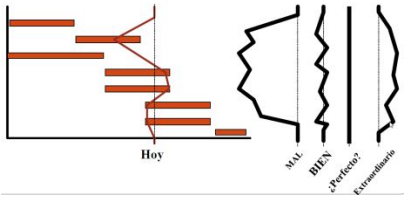
- Tan pronto se observen desviaciones se debe re-planificar y renegociar el plan del proyecto

Actividades involucradas

- Reuniones semanales de seguimiento
- Encuentros con el cliente
- Revisiones mensuales del proyecto
- Check-list de fin de fase
- Revisiones del plan
- Auditorias
- Reuniones de Postmortem: todos los proyectos tienen problemas, la idea es documentar, analizar y aprender de las cosas que han ido mal. Documenta aquellas que podrás hacer de forma diferente en el futuro, ítems reusables, herramientas reusables, lecciones aprendidas.

Comparar lo Planificado con lo Real

Sobre un gráfico de Gantt con lo planificado trazamos una línea quebrada que comienza y termina en la vertical del día actual, los vértices de esta línea se sitúan en la intersección de las tareas no realizadas o de aquellas que deberían haber comenzado pero que aún no lo han hecho. Estimar el % de las tareas no finalizadas para pasar la línea por ese punto.



Cuando aparecen problemas con respecto al proyecto, la decisión de que hacer debe ser tomada al nivel jerárquico apropiado, Operativo (los tecnicos, pequeños ajustes), Tactico (director del proyecto, retraso de una semana), Estrategico (grandes retrasos)

Seguimiento

Tipos

- Procesos: Hitos (momentos claves del proyecto) y Tareas: comienzo, fin y recursos
- Productos: Entregables, Calidad (conformidad del cliente)

Ambas visiones no son independientes, se planifica con las dos en mente**.**

Frecuencia

- Micro (semanal)
- Marco (mensual)
- Basado en eventos

Objetivo

- Para minimizar los riesgos
- Conocer y aceptar la realidad
- Tomar acciones correctivas
- Para evitar ser bomberos
- Para mejorar ciertas áreas como errores en el código, calidad del producto, etc.

Control

En administración de proyectos significa comparar el progreso con respecto al plan, así acciones correctivas pueden ser tomadas cuando desviaciones significativas ocurren con respecto al plan
La información y datos son el componente primario del control.
Calendarización: determina que tareas deben ser hechas secuencialmente y cuáles pueden ser echas al paralelo, aplica el tiempo estimado para cada una y se representan gráficamente por:

- Gráfico de **Gantt**: gráfico de barras, el largo de las mismas es proporcional a la duración, es la forma más fácil para visualizar un Schedule y es útil para el tracking, no así para mostrar dependencias de tareas.
- Gráfico de **Pert**: diagrama de red, muestra las tareas y sus dependencias, útil para ver cuando un set de tareas debe ser completado, útil para mostrar dependencias de tareas y hace evidente el camino crítico (aquellas actividades que no pueden atrasarse sin aumentar el tiempo total del proyecto), pero no sirve tracking.

Se **Replanifica** para que el calendario se ajuste a la realidad, se **Corrige** las desviaciones, en lugar de modificar el plan, se debe forzar al equipo para que la situación real se aproxime a la planificada.
Se llama **crisis de proyecto** al periodo desde que se produce una situación seria de desajuste hasta que se corrige.

Reuniones y Revisiones:

- **Revisiones Técnicas:** identificar aspectos técnicos: evolución del sistema, servicios, soluciones
  Ej: inspecciones formales, walkthrough de código
  La realiza personas con conocimientos técnicos del tema, no administradores
  Objetivo: Reportar el estado técnico real del proyecto a la administración
  Criterios para un producto o servicio: está completo? Cumple con los estándares? Cumple con el cronograma?
  Proveen entradas para las Revisaciones de administración (revisión del diseño de software)
- **Revisiones de la Administración:** identificar aspectos del proyecto: estado vs planes, presupuesto, estándares
  Ej: revisión de req (revisión de diseño, revisión de completitud de prueba)
  Objetivo: Mantener a la administración sobre el estado, dirección y acuerdos
  Son conducidas por líderes técnicos, administrador de proyecto y administradores
  Identificar y resolver riesgos
  Criterios: los recursos fueron distribuidos adecuadamente? Estamos tomando buenas decisiones basadas en métricas? Los riesgos ponen en peligro el éxito? Debemos cambiar de dirección o revisar planes?
- **Reuniones:** reunión de personas para un propósito de negocio, Ej: capacitación, staff, comité…

4 – Entregables y Soporte

Actividades del proyecto: deben producir outputs tangibles, a fin de poder juzgar el progreso del proyecto.

Hitos (o Milestone): son puntos finales de las actividades del proceso (no necesariamente entregas), representan el fin de una etapa lógica en el proyecto y para cada uno de ellos debe existir una salida formal. No necesariamente todas las actividades deben finalizar en un hito (consumen mucho tiempo).

Entregables: son hitos, resultado del proyecto entregado al cliente al final de una fase principal del proyecto, estos están definidos en el libro de requerimientos o de en plan. La entrega no tiene que ser necesariamente la culminación del proyecto, se puede comenzar una fase de soporte que genere nuevas entregas con agregados de funcionalidad al producto.

Las entregas son hitos, pero los hitos no necesariamente son entregas.

Riesgo

Una tarea importante del gestor de proyectos es anticipar los riesgos que podrían afectar a la programación del proyecto o a la calidad del software a desarrollar y emprender acciones para evitar estos riesgos.
Los resultados de este análisis de riesgo se deben documentar a lo largo del plan de proyecto junto con el análisis de consecuencias cuando el riesgo ocurra. Identificar estos y crear planes para minimizar sus efectos en el proyecto se llama gestión de riesgos. El proceso de gestión de riesgos es un proceso iterativo, se aplica a lo largo de todo el proyecto, a medida que se tenga más información, los riesgos deben analizarse y priorizarse, y deben modificarse los planes de mitigación y contingencia si fuera necesario.
Los primeros riesgos pueden ser identificados en el análisis de requerimientos por ejemplo que tenga atributos de complejidad, dificultad, novedad, etc.
Un riesgo se escribe como “Si <condición> entonces <consecuencia tangible>
Riesgo: es un problema esperando para suceder, un evento que podría comprometer el éxito del proyecto, probabilidad de que una circunstancia adversa ocurra, toda actividad lleva implícita un riesgo y los riesgos están caracterizados por la ***incertidumbre (probabilidad que ocurra)***.
Clasificación:

- riegos del **Proyecto** (afecta la calendarización o incrementa los costos)
- del **Producto o Técnicos** (afectan la calidad o el rendimiento del software)
- del **Negocio** (afectan a la organización que desarrolla el software, puede ser de mercado, de ventas, de gestión, de presupuesto, de estrategia)

Grupos:

- Genéricos a todos los proyectos
- Específicos, implican un conocimiento profundo del proyecto.

Categorías:

- Relacionados con el tamaño del producto
- Con el impacto en la organización
- Con el tipo de cliente
- Con la definición del proceso de producción
- Con el entorno de desarrollo
- Con la tecnología
- Con la experiencia y el tamaño del equipo

Los **riegos** tienen condiciones y consecuencias inciertas y pueden ser dinámicos, los **problemas** son ciertos.
Se deben clasificar los riesgos en orden de exposición, buscar un punto de quiebre en los valores de la lista y focalizarse sobre las acciones para los riesgos más altos. Esta lista debe ser revisada regularmente, agregar ítems si es necesario y ajustar la exposición según sea apropiado.

Top 10 Riesgos

Déficit de personal, cronogramas y presupuesto irreales, desarrollo de funciones de software erróneas, interfaces de usuario erróneas, detalles de interfaz y cambios de requerimientos, flujo constante de cambio en los requerimientos, déficit en las tareas desempeñadas externamente, déficit en componentes provistos externamente, déficit de performance, innovación de tecnologías.

Etapas

1. **Identificación de los riesgos:** descubrimiento de posibles riesgos sin valorarlos o priorizarlos.
2. **Análisis de riesgos:** valorar la **probabilidad** (bajo 10-25%, ..., alto >75%) y consecuencia de los riesgos o **impacto** (catastrófico, serio, tolerable o insignificante, este se mide de acuerdo a dos aspectos, el Alcance y la Temporalizarían de los efectos). La **exposición** se calcula como el producto entre la probabilidad y el impacto y representa la amenaza total del riesgo, los riesgos más importantes para un proyecto se ordenan por exposición.
   Exposición de Riesgo = Probabilidad * Impacto
   Se crea una tabla de riesgos, ordenada por probabilidad y prioridad, indica: nombre, categoría, probabilidad, impacto, RMMM (gestión, monitorización, mitigación de riesgos) la cual es la estrategia y forma de actuar del equipo de trabajo frente a los riesgos, indica como Mitigarlos (estrategia general y pasos para mitigar el riesgo), Monitorizarlos (factores a monitorizar) y Gestionarlos (planes de contingencia)

3. **Planificación de riesgos:** crear planes para abordar los riegos. **Planes de Mitigación: Prevención** (para reducir su probabilidad), de **Minimización** (reducir el impacto) y **Planes de Contingencia** (plan a seguir cuando ocurra)
4. **Supervisión de riesgos:** valorar los riesgos identificados para decidir si este es más o menos probable y si han cambiado sus efectos. La supervisión debe ser un proceso continuo y en cada revisión cada uno de los riesgos debe ser analizado por separado.

Estrategias frente los riesgos

- **Reactivas:** se evalúan las consecuencias del riesgo cuando este ya ha ocurrido, esto pone en peligro el proyecto, actual en consecuencias. Ej: apagado de incendios.
- **Proactivas:** se caracteriza por una evaluación previa y sistemática de riesgos y sus consecuencias, creando:
    - planes de mitigación y minimización de consecuencias
    - planes de contingencia (estar preparado para lo peor)

Gestión de Riesgos

Estimación

Es una predicción del tiempo que llevará un proyecto o el costo de desarrollarlo.
Estimar no es planear y planear no es estimar, las estimaciones son las bases de los planes pero los planes no tienen que ser lo mismo que lo planeado, mayor diferencia entre lo estimado y lo planeado, mayor riesgo.
El problema de las estimaciones que es difícil hacer predicciones hacia el futuro, su principal problema a la hora de determinar estimaciones es la falta de conocimiento de lo que se va a hacer para llevar a cabo el proyecto, por lo que se Subestima y se va determinando a medida que avanzamos en el proyecto.
Objetivo de la estimación es la **precisión**, y del plan es obtener un resultado. El principal propósito de las estimaciones es determinar si los objetivos son lo suficientemente realistas para lograrlo.
Estimamos para:

- Predecir completitud
- Administrar riesgos

Típicamente la primera estimación difiera hasta un 400 %
La estimación es un numero único (es un objetivo disfrazado de estimación), cuando es un numero único hay que preguntarse qué probabilidad hay de alcanzarlo, toda estimación tiene asociada una probabilidad de alcanzarlo.
Antes de que el proyecto comience, el gestor de proyecto y el equipo de software deben estimar el trabajo que habrá de realizarse, los recursos que se requerirán y el tiempo que transcurrirá desde el principio hasta el final.
La estimación para una tarea de ingeniera de software requiere experiencia y acceso a buena información histórica.
La estimación coloca los cimientos para que las demás actividades de planificación del proyecto y esta proporciona la ruta para la ingeniera de software exitosa. El riesgo de estimación se mide por el grado de incertidumbre.
Se debe estimar Tamaño, Esfuerzo, Calendario, Costo y Recursos Críticos.
Las estimaciones del costo y el esfuerzo nunca será una ciencia exacta, sin embargo para lograr estimaciones confiables tenemos varias opciones:

- Demorar la estimación hasta más tarde en el proyecto (no es practica ya que la estimación se tiene que producir por adelantado)
- Basar las estimaciones en proyectos similares que hayan sido completados (puede funcionar pero hay más variables en juego como el equipo, el cliente, etc)
- Emplear técnicas de descomposición relativamente simples para generar estimaciones de costo y esfuerzo del proyecto
- Utilizar uno o más modelos empíricos en la estimación de costo y esfuerzo

Las últimas dos son viables, deben aplicarse juntas, cada una empleada como una marca de verificación de la otra.
Se recomienda el uso de diferentes métodos de estimación para un mismo proyecto y asi constatar los resultados obtenidos, ninguno es 100% seguro.

Errores de Estimación

Vienen de información imprecisa del software a estimar, información imprecisa de las capacidades de la empresa que realizara el software, demasiado caos en el proyecto, imprecisión generada por el proceso de estimación.
Lo que más afecta la estimación es:

- Requerimientos no relevados (no funcionales generalmente)
- Requerimientos mal relevados

Siempre guardar unos días extras por algo que salga mal o falte (buffer o colchón)

Técnicas de estimación -> Contar

Se cuenta todo aquello relacionado con el tamaño del software que se está estimando, lo que se cuenta debe requerir poco esfuerzo y se debe buscar algo que esté disponible lo más pronto posible en el proyecto.
Se debe contar:

- En etapas tempranas: requerimientos, características, CU, historias, etc.
- En mitad del proyecto: perdidos de cambios, páginas web, reportes, tablas, etc.
- Más avanzado del proyecto: defectos, clases, tareas, etc.

Formas de realizar Estimaciones

- Usar la experiencia individual del estimados
- Usar la experiencia de la organización: crear una base de datos en la misma para llevar a cabo las estimaciones transfiriendo conocimiento de cada individuo a la organización.

Si no dispongo de datos históricos se puede usar las tablas de productividad por tipo de software como inicio.

Una Buena Estimación

Una buena estimación provee una vista lo suficientemente clara del proyecto para permitirle al líder del proyecto tomar las decisiones correctas para controlarlo y lograr los objetivos. Esto nos da mejor visibilidad, mejor calidad (minimiza errores), mejor coordinación con diferentes aéreas, mayor credibilidad del esfuerzo de desarrollo.

Estrategia de Estimación

- Tenga una idea del dominio y busque una unidad de peso
- Use un método para convertir las unidades de peso en estimaciones
- Use juicio de expertos como último resorte

Estimación de Tamaño

Tamaño del Producto = Req Funcional (tamaño de la funcionalidad del producto) + Req No Funcional
El número que más se busca en la estimación de software es el tamaño del software a ser construido, el tamaño puede ser expresado en LOC, PF, numero de req, numero de web pages, caso de uso, etc.

- Líneas de Código: la métrica de tamaño tradicional para estimar el esfuerzo de desarrollo y productividad ha sido LOC. Se han propuesto varios modelos de estimación, la mayoría de ellos son funciones de las líneas de código. Los principales problemas de utilizar LOC como métrica para estimación del esfuerzo son la falta de una definición universal de línea de código, su dependencia con el lenguaje de desarrollo y la dificultad de estimar en fases tempranas de desarrollo la cantidad de líneas que tendrá una aplicación
- Puntos de función: el análisis por puntos de función es un método para cuantificar el tamaño y la complejidad de un sistema software en términos de las funciones de usuario que este desarrolla. Esto hace que la medida sea independiente del lenguaje o herramienta utilizada. Otra ventaja es que puede ser estimados a partir de la especificación de requisitos, haciendo posible de este modo la estimación del esfuerzo en etapas tempranas.

Métodos para Estimación

Basado en la experiencia

- Datos Históricos: se compara un nuevo proyecto con uno pasado, se utilizan para evitar las discusiones entre desarrolladores y el cliente, se recolectan datos de tamaño (LOC, PF, CU), esfuerzo, tiempo, defectos y luego se calibran (convierte las cuentas a estimados, x LOC por mes, LOC a esfuerzo)
- Juicio experto: es el más usado, el 75%, usa la fórmula de (Opt + 4Hab + Pes)/6
    - Puro: un experto estudia las especificaciones y hace su estimación, se basa en los conocimientos del experto, si este desaparece, a empresa deja de estimar
    - Delphi: se les dan las especificaciones a un grupo de personas que intentan adivinar lo que costara el desarrollo tanto en esfuerzo como en duración, las grupales suelen ser mejores que las individuales. Luego les remiten sus estimaciones individuales al coordinador, se reparten las estimaciones anónimamente, se discute, se revisa sus propias se envía al coordinador nuevamente. Se repite el proceso hasta que la estimación converge de forma razonable.
- Analogía: consiste en comparar las especificaciones de un proyecto, con las de otros proyectos, se compara usuarios, complejidad, tamaño, tecnologías, etc.

Basado en los recursos
Basado en el mercado
Basado en los componentes del producto o en el proceso de desarrollo
Métodos algorítmicos

Modelado Ágil

Es un conjunto de principios y prácticas para modelado y análisis de requerimientos, que complementa a la mayoría de los métodos de desarrollo iterativos e incrementales (los métodos agiles son un subconjunto de los modelos iterativos)

Manifiesto Ágil

Fundaciones sobre las que hoy en día se basan las metodologías agiles, los métodos con pensamiento ágil deben tener los siguientes principios:

- **Individuos e interacciones por sobre procesos y herramientas** (los roles son intercambiables, se centra en los individuos no los roles)
- **Software funcionando por sobre documentación detallada** (se documenta aquello que agregue valor al producto, lo útil se documenta, si una documentación para el proyecto tiene valor agregado se debe hacer en forma creciente)
- **Colaboración por sobre negociación con el cliente** (el cliente se vuelve alguien importante en el proyecto, no se negocia, se colabora en forma conjunta y así surgen los requerimientos emergentes)
- **Responder a cambios por sobre seguir un plan**

La diferencia inmediata es la exigencia de menor documentación, sin embargo las más importantes son:

- ***Los métodos agiles son adaptables en lugares de predictivos***
- ***Los métodos agiles son orientados a la gente en lugar de orientados al proceso***
- La prioridad es satisfacer al cliente con release tempranos y frecuentes (los métodos agiles usan ciclos de vida iterativos con entregas frecuentes) + release + feedbak + calidad
- Se pueden recibir cambios de software, aun en etapas finales
- Técnicos y no técnicos trabajan juntos en el proyecto
- Ambiente de trabajo abierto, comunicación face to face, el espacio físico debe favorecerla comunicación.
- La mejor métrica de progreso es la cantidad de software funcionando.
- Simplicidad en las herramientas
- Gente motivada en el equipo
- El negocio fija las prioridades
- Incrementos visibles y usables
- Modelos públicos a todo el equipo de trabajo.
- A intervalos irregulares el equipo evalúa su desempeño y ajusta la manera de trabajar.
- Triple restricción: recursos (fijo el equipo), tiempo (timeboxing), alcance (requerimiento cambiantes).
- Es conducido por el valor (el grado de progreso en el proyecto se mide atreves de inspecciones frecuentes) y no por el plan.

SCRUM (proceso agil)

Scrum is just one of the many iterative and incremental agile software development method.
In the SCRUM methodology a sprint is the basic unit of development.
So if in a SCRUM sprint you perform all the software development phases (from requirement analysis to acceptance testing), you can say SCRUM sprints correspond to AGILE iterations.

Es un framework que permite crear nuestro propio proceso para crear nuevos productos, es simple (pocas reglas) y puede ser implementado en pocos días, pero puede tomar mucho tiempo perfeccionarlo.
Su **objetivo** es ofrecer el más alto valor de negocio en el menor tiempo.

Sprint

Periodo fijo en el tiempo (30 días), dentro de él no se puede cambiar su alcance, ni agregar funcionalidad, ni modificar las reglas del equipo. Su objetivo es un objetivo claro a alcanzar. NO SE PERMITEN CAMBIOS!. Se puede cancelar, pero esto lleva acompañado un costo.

Beneficios

- Se gestionan los cambios de requerimientos
- Los clientes ven incrementos que refinan los requerimientos en tiempos razonables
- Mejores relaciones con el cliente.

Usamos Scrum cuando

Para proyectos con muchos cambios de requerimientos

- Nuevos
- Procesos de investigación

- Productos complejos (se hace una iteración 0 en la que se estudia bien el proyecto y se lleva a cabo una estimación de la dimensión de la arquitectura y del proyecto mismo)

Cimientos

- **Empirismo**: totalmente basados en la experiencia de quien lo ejecuta, se alcanza con inspecciones frecuentes y correspondientes ajustes y a veces alcanza objetivos reduciendo funcionalidades, las metodologías rigurosas se basan en métodos definidos donde tienen definido cuáles son sus entradas y las salidas (como línea de ensamble)
- **Timeboxing**: crea un ritmo que guía el desarrollo, es una técnica de planificación de proyectos donde el Schedule se divide en un numero de periodos (2 a 6 semanas) cada uno con entregables, fechas y costos.
- **Auto organización**: pequeños grupos sin un líder, los equipos se auto organizan a fin de determinar la mejor manera de entregarlas funcionalidades de más alta prioridad, define sus horarios, preservar los equipos entre proyectos, sinergia.
- **Colaboración**: líderes de Scrum, diseñadores y clientes colaboran con los desarrolladores.
- **Priorización**: trabajar en lo más importante, se debe priorizar de acuerdo a lo que quiere el cliente y es de urgencia.

Proceso

1. Update product backlog
2. Sprint Planning meeting
3. Daily meeting and Daily work (Daily Circle)
4. Product Increment -> Release n
5. Sprint Review
6. Sprint Retrospective -> Back 1

3 Roles (equipo)

- **Scrum Master** (responsable que las practicas, valores y reglas se realicen, es el nexo entre la gerencia y el equipo, **dirige los Scrum diarios**, asegura la toma de decisiones rápidas, identifica el dueño del producto, responsable de las reuniones, **toma decisiones en reuniones de Scrum**, realiza seguimientos, registra y resuelve problemas, mantiene enfocado el equipo, mantiene actualizado en backlog del Sprint basado en las reuniones diarias...)
- **Equipo** (+- 7 personas, auto organizado, compromiso de entregar un backlog al final de un sprint, libertad de acción, sin roles, todos codifican)
- **Dueño del producto** (controla y gestiona el backlog, es una persona, define que quiere y se debe cumplir, define prioridades)

El Scrum Master, el dueño del producto y el equipo producen un Backlog de producto

4 Reuniones (obligatorias)

- **Planning:** el equipo se reúne con el dueño, la gerencia y los usuarios para definir la funcionalidad de a implementar, el equipo decide cómo llevarlo a cabo. 4 hs max **para planificar el sprint siguiente**, genera un entregable (la minuta del sprint, la cual tiene una lista de tareas a entregar, los miembros del equipo, con que se comprometen y la fecha de finalización). Para **discutir criterios de aceptación y reestimación** con el Product Owner los ítems de alta prioridad del producto Backlog, entre el final de una iteración (Sprint) y el comienzo de la otra.
- **Daily (1/2 Tareas obligatorias):** deben ser cortas (15 – 30 min) cada 24 hs, con hora y lugar fijos, solo participan los desarrolladores, no es una reunión de diseño, **asiste todo el equipo**, no son para solucionar problemas. El Scrum Master pregunta que se completó desde la última reunión, que obstáculos se presentaron, que se hará en la próxima reunión. La gerencia asiste pero solo los desarrolladores participan. Su objetivo es conocer el estado de avance de las tareas (no resolver problemas), es un **constante monitoreo y control**. Mejora la comunicación, elimina otras reuniones, promueve decisiones rápidas, mejora el conocimiento de todos.
- **Review:** reunión informativa de 4 hs max, el equipo presenta el incremento desarrollado a gerentes, clientes, usuarios, owner, etc, todo el equipo participa y se invita a todo el mundo. Se reportan los problemas, cualquier ítem puede ser agregado, quitado o re-priorizado, se estima el nuevo Sprint y se asignan las tareas.
- **Retrospectiva:** periódicamente que echa un vistazo a lo que funciona y a lo que no, 4 hs max, se realiza luego de cada Sprint (no necesariamente si son cortos los Sprints), participa todo el equipo, es una reunión que se lleva a cabo para aprender que se hizo mal y que bien. Responde que hay que comenzar a hacer, que hay que dejar de hacer y que hay que continuar haciendo. Deja un formulario documentado.

- **Grooming** (opcional): reunión de refinamiento del Producto Backlog de cara al Sprint Planning, en la cual participa todo el equipo y se prepara el próximo Sprint Planning por lo cual es previa a esta, casi siempre a mitad de cada sprint. También se discuten criterios de aceptación y prioridades.

3 Entregables

- **Product Backlog:** lista de todos los trabajos deseados en el proyecto, cola de prioridades de funcionalidades técnicas y de negocio que deben ser desarrolladas, contiene la lista de requerimientos (funcionales y no funcionales, esta se obtiene de los CU o US) posee características, mejoras, bugs, etc, está incompleto al principio aunque solo se necesita lo suficiente para realizar el primer Sprint de 30 días.
  Debe ser priorizado por el dueño y contiene problemas que deben ser solucionados.
  Trabajar con Backlog es como jugar Asteroides: grandes rocas (Epicas) se rompen en pedazos de rocas más pequeñas (Historias) hasta que son lo suficientemente pequeñas para ser eliminadas (desarrolladas y entregadas).
  Su estimación es iterativa, si un ítem no puede ser estimado se debe dividir en el Backlog.
  Estimación a nivel de User Storys con Poker Planning, y a nivel de Epica o Feature con Shirts Size.
  Backlog de trabajo: es la cantidad de trabajo que queda por ser realizado.

Backlog item	Estimación
Permitir que un invitado a hacer una reserva.	3
Como invitado, quiero cancelar una reserva.	5
Como invitado, quiero cambiar las fechas de una reserva.	3

- **Sprint Backlog (2/2 Tareas obligatorias):** el equipo determina que debe hacerse para cumplir el objetivo del Sprint. El dueño suele asistir, se realizan listas de tareas que demoran 4 a 16 hs para completarse, el Sprint Backlog no se modifica durante el desarrollo, si este no puede desarrollarse el Scrum Master y el dueño deciden si algún ítem puede ser removido y si alguna funcionalidad puede eliminarse. Debe estar actualizado y con los últimos estimados de los desarrolladores

Tareas	L	M	M	J	V
Codificar UI	8	4	8		
Codificar negocio	16	12	10	4	
Testear negocio	8	16	16	11	8

- **Release n**

User Stories

Son una descripción corta de una funcionalidad valuada por un usuario o cliente de un sistema. Se comunican en conversaciones entre el dueño del producto y el equipo. No son requerimientos, son marcadores para conversaciones más detalladas y análisis que deberán ocurrir conforme esas historias vayan implementándose. Los requerimientos pueden ser divididos en 2 o más US, esto lo hace el Owner con el Master.
Se expresan con “Como <<quien realiza la acción>> yo quiero << acción que realizara el sistema>> de forma tal que <<porque es necesaria la actividad (valor de negocio)>>
Están compuestas por

- Tarjeta (descripción de la historia)
- Conversación (discusiones sobre ella)
- Confirmación (pruebas usadas para determinar cuándo una historia está completa)

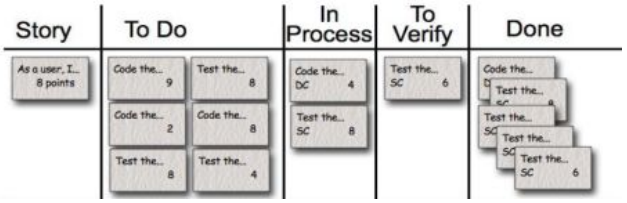
Pruebas de aceptación de User Stories: expresan detalles resultantes de las conversaciones entre clientes y desarrolladores, suelen usarse para completar detalles de las historias, las pruebas se ven como notas en el dorso de las historias. Deben escribirla el cliente antes de que la programación empiece.
Spike: tipo especial de historia, utilizada para quitar riesgo e incertidumbre de una User Story u otra faceta del proyecto.
Las Historias se derivan de:

- **Épicas** (Epics, son requerimientos de alto nivel que se utilizaran para coordinar el desarrollo)
- **Características** (Features)

Epics -> Features -> User Stories

Herramientas

- **Tarjeta** (código del ítem de backlog, numero de sprint, nombre de la actividad, iniciales de la persona, tiempo estimado en hs)
- **Tablero** (Story, To do, In process, To verify, Done)


Story
To Do
In Process
To Verify
Done
As a user, I...
8 points
Code the...
9
Test the...
8
Code the...
2
Code the...
8
Test the...
8
Test the...
4
Code the...
DC
4
Test the...
SC
8
Test the...
SC
6
Code the...
Test the...
Test the...
Test the...
SC
6


Código del Item BackLog
Nro de Día del Sprint
15
3
Nombre de la actividad
Generar cronograma de Pago
RU
4h
Iniciales de la Persona
Tiempo estimado en horas

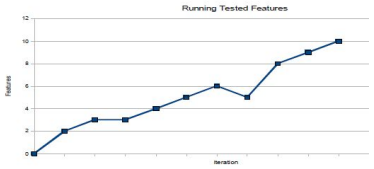
Métricas (Medidas) Agiles

Regla de oro agil sobre metricas: la medicion es una salida, no una actividad
La mejor metrica es la cantidad de software funcionando
Clases:

- Informacional: dicen que esta pasando
- Diagnostico: indican areas de mejoras
- Motivacional: influyen en el comportamiento

RTF (Grafico): su principio para guiar a la eleccion de la metrica es “software funcionando es la mejor medida de progreso”.

- **Running** (entregado de un producto)
- **Test** (pruebas de aceptacion continuas, pasadas)
- **Features** (aspectos dados por el cliente)



Capacidad: es cuanto trabajo se puede realizar en un periodo de tiempo dado basado en la cantidad de tiempo ideal disponible por el equipo, con el tiempo la capacidad deberia aumentar.
Se determina como (Horas_disponibles_trabajo x Dias_disponibles_iteracion) = Capacidad

- Se puede medir en Esfuerzo (horas), Puntos de historia (story points)

Velocidad: es una observacion empirica de la capacidad de un equipo para completar el trabajo por iteracion, no es una estimacion, no es un objetivo y tampoco es comparable entre equipos ni proyectos.
Cantidad de SP quemados por Sprint, luego se suman y se saca un promedio. Solo cuenta el trabajo completado

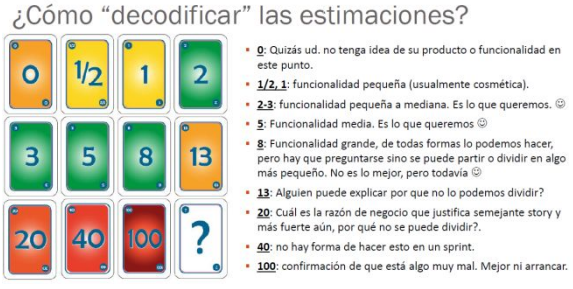
- Se puede medir en Horas ideales, Puntos de historia, Historias
- La velocidad corrige errores de estimacion.

Brundown Chart: indica cuanto trabajo puedo trabajar y cuanto trabajo se ha completado

Estimacion Agile

En los equipos Agile, las features/stories son estimadas usando una medida de tamaño relativo (las personas no saben estimar en términos absolutos, somos buenos comparando cosas) conocida como story points (SP) la cual no es una medida basada en el tiempo, es una medida especifica (del equipo) de, complejidad, riesgo y esfuerzo, la cual indica, cuan compleja (grande) es. La complejidad tiende a incrementarse exponencialmente.
El tamaño es una medida de la cantidad de trabajo necesaria para producir una feature/story.
Tamaño no es lo mismo que esfuerzo, para medir el tamaño usamos talle de remeras, Fibonacci..., una vez elegida la escala, no se cambia.
Las estimaciones basadas en tiempo son más propensas a errores.
Método de estimación -> Poker Estimation: popular entre los profesionales Agiles, combina la opinión experto, analogía y desegregación. Los participantes en el son desarrolladores “las personas más competentes en resolver una tarea deben ser quienes las estiman”.
Prerequisitos: lista de Stories/Features a ser estimadas, cada estimador tiene un mazo de cartas.
Agile es EMPIRICO, inspeccionar y adaptar es Mandatorio.
Pasos:

1. Determinar cuál será la base story (canónica) que se utilizara para comparar al resto
    a. La story a ser estimada se lee al equipo, se discute y se le hacen preguntas al dueño del producto, cada estimador elige una carta, se dan vueltas todas y si todos eligen el mismo valor ese es el estimado, sino se discuten los resultados y se vota de nuevo
2. Se toma la próxima story y se discute con el dueño
3. Cada estimador asigna a la story un valor de comparación contra la base “cuán grande, compleja, riesgosa es esta comparada con la canónica?”



UNIDAD 2 – METRICAS

Medida: proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto. Pueden ser Directas (longitud tornillo) Indirectas (la calidad de los tornillos), Directas del proceso (costo), Directa del producto (LOC).
Medición: proceso necesario para determinar la medida.
Indiciador: es una métrica o combinación de métricas que proporcionan una visión profunda del proceso de software, del proyecto de software o del producto en sí.
Escala: conjunto de valores con propiedades definidas, puede ser Nominal (categoría, atributo como fase de desarrollo, tipo de producto, ciclo de vida), Ordinal (define un orden, sin intervalos como el nivel de impacto leve medio alto), Intervalo (designa un ordenamiento equidistante como la temperatura), Radio (intervalos equivalentes con un punto cero absoluto como líneas de código). A partir de ellos se pueden obtener criterios de calidad utilizables para definir los release de un proyecto.
Unidad: una cantidad particular, definida y adoptada por convención, con la que poder comparar otras cantidades de la misma clase para expresar sus magnitudes respecto a esa cantidad particular. Una unidad sirve para expresar una o varias métricas cuyo tipo de escala sea intervalo o ratio.
Relaciones: una métrica está definida para uno o más atributos, dos métricas pueden relacionarse mediante una función de transformación, una métrica puede expresarse en una unidad (solo para métricas cuya escala sea de tipo intervalo o ratio)

Métrica

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
Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.
Es una forma de medir y una escala, definidas para realizar mediciones de uno o más atributos.
Es un buen medio para entender, monitorizar, controlar, ordenar y probar el desarrollo de software y los proyectos de mantenimiento.
Medimos para informar, motivar, comparar, entender, señalar (tomar control y lograr supervisarlo), evaluar (tener noción de los costos), predecir (estimar nuevos proyectos), mejorar...
Su objetivo es entender que ocurre durante el desarrollo y mantenimiento, permitir controlar que es lo que ocurre en nuestro proyecto, poder mejorar nuestros procesos y productos.
La cantidad de métricas a utilizar depende del tamaño del proyecto y del tipo, si se trata de un proyecto simple hay que usar 4 métricas y en uno con Scrum con 2 alcanza.
Las métricas deben elegirse que contar y cuando contar, “todo lo que puede ser contado no necesariamente cuenta y todo lo que cuenta no necesariamente puede ser contado”.
En métricas no se debe adaptar uno para que la métrica de correctamente (comportamiento adaptativo), cuando sabemos que nos están midiendo nos adaptamos.
Si no me gusta métrica, no la cambio, cambio el contexto, no hay que forzar la métrica al resultado que uno quiere obtener.

Calidad de una Métrica

La calidad de una métrica está dada por la **Validez** (contraparte de exactitud, grado en el cual la métrica refleja el significado real del concepto que se observa) y la **Confiabilidad** (contraparte de presión, su objetivo son las métricas sin errores, puede representarse con la desviación estándar). Menor variación mayor confiable la métrica.

Errores de medición

Valor Medido = Valor Real + Error Sistemático (asociado a la validez) + Error Aleatorio (a la confiabilidad)
Ej: la métrica “línea de código” se puede definir para realizar mediciones del “tamaño” de un “modulo”

Principios de la Métricas:

- Hay que automatizar tanto el proceso como sea posible para obtener el resultado de la métrica lo antes posible y así poder analizar el resultado.

- Análisis que se obtienen con las métricas: Calendario, Esfuerzo, Funcionalidad %
- Se debe elegir métricas que provean información que ayuden a responder preguntas.
- Comenzar con lo posible
- Las métricas no tienen que ser usadas ni diseñadas para medir la performance de un individuo.
- Automatización de la recopilación de datos y su análisis
- Cada métrica debe validarse empíricamente antes de publicarse o aplicarse a la toma de decisiones.

Clasificación

- **Métrica Directa:** las que tomamos directamente sobre el proyecto, una métrica de la cual se pueden realizar mediciones sin depender de ninguna otra métrica y cuya forma de medir es un método de medición, esta métrica puede ser utilizada en funciones de cálculo.
  LOC, longitud de un tornillo, Horas-programador diaria, Costo por hora-programador en unidades monetarias, etc.
- **Métrica Indirecta:** no miden un atributo directamente, una métrica cuya forma de medir es una función de caculo, es decir, las mediciones de dicha métrica utilizan las medidas obtenidas en mediciones de otras métricas directas o indirectas. Su forma de medir es una función de cálculo y puede ser utilizada en una función de cálculo.
  Horas-programador totales, calidad de un tornillo medidos contando la cantidad de defectuosos, LOC por hora de programador, Costo total actual del proyecto en pesos..

Métricas de Software

- **Orientadas al tamaño**: provienen de la normalización de las medidas de calidad y/o productividad considerando el tamaño de software que se haya producido. Se seleccionan las LOC como valor de normalización, no se aceptan universalmente como la mejor forma de medir proceso de software, muchos dicen que son algo fácil de contar y que están presentes en todos los proyectos pero otros dicen que no son algo estable y que las LOC dependen del lenguaje de programación. Errores x KLOC, Defectos x KLOC, etc.
- **Orientadas a la función**: usan como valor de normalización una medida de la funcionalidad que entrega la aplicación, la métrica orientada a la función más usada es el Punto de Función. Los partidarios afirman que el PF es independiente del lenguaje y que se basa en datos que es más probable que se conozcan temprano en la evolución del proyecto. Los opositores dicen que el cálculo se basa más en datos subjetivos que objetivos y que el PF no tiene significado físico, es solo un número.

Métricas de calidad de Software

El primer objetivo en el proyecto es medir errores y defectos, las métricas que provienen de estas medidas proporcionan una indicación de la efectividad de las actividades de control y de la garantía de calidad.
Medidas de calidad:

- Corrección
- Facilidad de mantenimiento
- Integridad
- Facilidad de eso

Dominio de las métricas

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

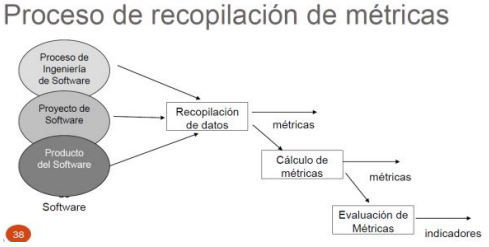
Indicadores

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)

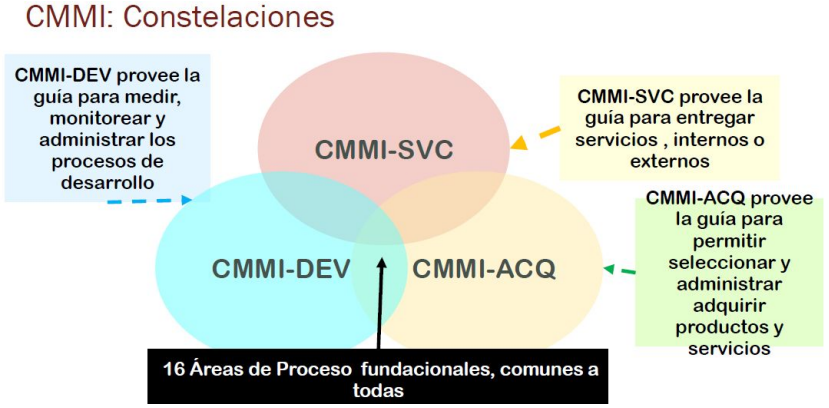
Calidad de proceso:

- 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

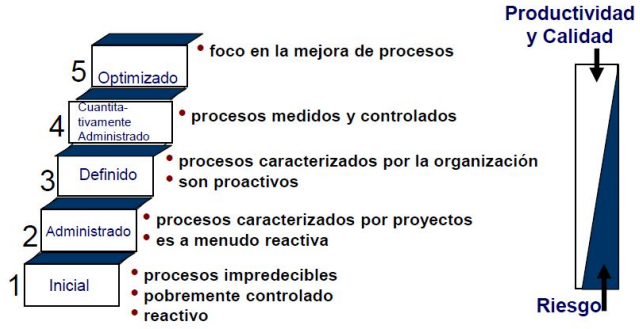
- Mejora de Procesos: mejora la comunicación, se reducen los defectos en forma temprana, reducción de re-trabajo, reducción costos de desarrollo y reducción de mantenimiento...
    - SPICE
    - IDEAL (Initiating, Diagnosing, Establishing, Acting, Learning)

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)

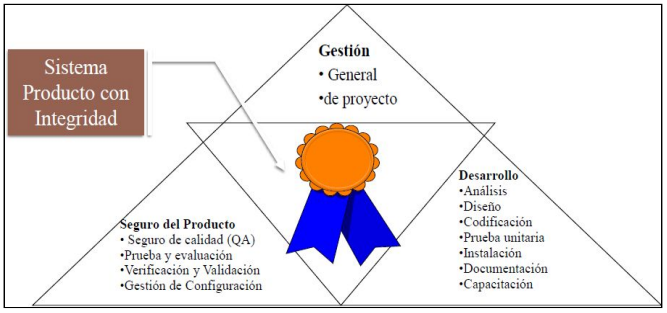
Su propósito es establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida del mismo.
Ventajas: aumenta la protección contra cambios innecesarios, mejora la visibilidad del estado del proyecto y sus componentes, disminuye el tiempo de desarrollo, aumenta 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

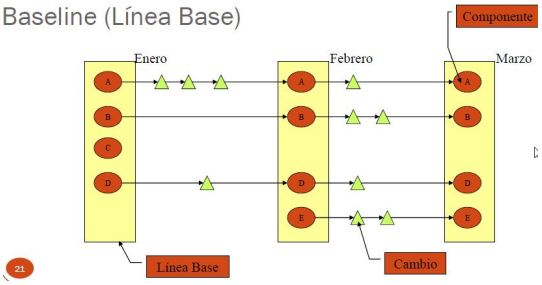
1 - Identificación

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

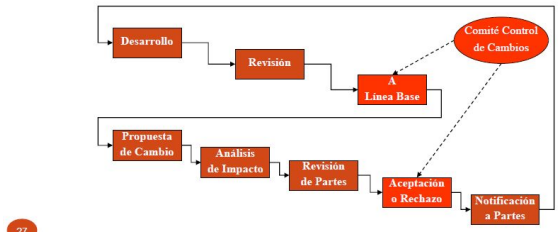
- Se logra mediante la definición de sus líneas base (baselines) y de los cambios que los mismos pueden sufrir durante la evolución del sistema.
- Las líneas base (baseline) y sus cambios especifican la evolución del sistema.

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

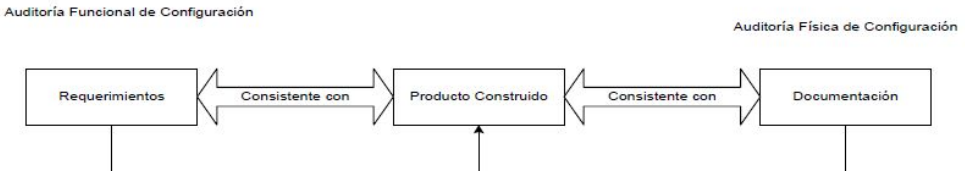
- Determinar la semejanza entre el estado actual del sistema y el establecido como línea base.
- Provee el mecanismo para establecer una línea base.

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.

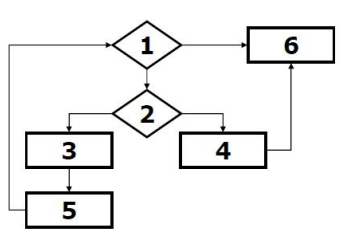
El testing de caja blanca se utiliza porque hay errores que pueden no ser detectados por testing de caja negra, como errores tipográficos, caminos lógicos que se cree que no se ejecutara, etc

- **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.