**Unidad Nº 4:**

Gestión de proyectos de Software

* La gestión de un proyecto de software comienza con un conjunto de actividades que globalmente se denominan planificación del proyecto. Antes de que el proyecto comience, el gestor y el equipo de software deben realizar una estimación del trabajo a realizar, de recursos necesarios y del tiempo que transcurrirá desde el comienzo hasta el final de su realización.
* Siempre que estimamos, estamos mirando hacia el futuro y aceptamos cierto grado de incertidumbre.
* Existen técnicas útiles para la estimación del esfuerzo y del tiempo.
* Las métricas del proyecto y del proceso proporcionan una perspectiva histórica y una potente introducción para generar estimaciones cuantitativas. La experiencia anterior (de todas las personas involucradas) puede ayudar en gran medida al desarrollo y revisión de las estimaciones.
* La estimación de recursos, costos y planificación temporal de un esfuerzo en el desarrollo de software requiere experiencia, acceder a una buena información histórica y el coraje de confiar en predicciones (medidas) cuantitativas cuanto todo lo que existe son datos cualitativos.
* La estimación conlleva un riesgo inherente y es este riesgo el que lleva a la incertidumbre.

Complejidad del proyecto

* La complejidad del proyecto tiene un gran efecto en la incertidumbre, que es inherente en la planificación. Sin embargo, la complejidad es una medida relativa que se ve afectada por la familiaridad con esfuerzos anteriores.
* Se podría considerar una aplicación sofisticada de comercio electrónico como **excesivamente compleja** para un desarrollador que haya realizado su primera aplicación. Sin embargo, para un equipo de software que desarrolle su enésimo sitio web de comercio electrónico podría considerarse **sumamente fácil** (una de tantas).
* Se han propuesto una serie de medidas cuantitativas de la complejidad del software. Tales medidas se aplican en el nivel de diseño y de codificación, y por consiguiente son difíciles de utilizar durante la planificación del software (antes de que exista un diseño o un código).
* Sin embargo, al comienzo del proceso de planificación se pueden establecer otras valoraciones de complejidad más subjetivas (por ejemplo, los factores de ajuste de la complejidad del punto de función).

Tamaño del proyecto

* El tamaño del proyecto es otro factor importante que puede afectar a la precisión y a la eficiencia de las estimaciones.
* A medida que el tamaño aumenta, crece rápidamente la interdependencia entre varios elementos del software. El problema de la descomposición, un enfoque importante hacia la estimación, se hace más difícil porque los elementos descompuestos pueden ser todavía excesivamente grandes.
* Parafraseando la ley de Murphy: “lo que puede ir mal, irá mal”, y si hay más cosas que pueden fallar, más cosas fallarán.

Grado de incertidumbre estructural

* El grado de incertidumbre estructural tiene también efecto en el riesgo de la estimación.
* En este contexto, la estructura se refiere:
  + Al grado en el que los requisitos se han definido.
  + La facilidad con la que pueden subdividirse funciones
  + La naturaleza jerárquica de la información que debe procesarse.
* El riesgo se mide por el grado de incertidumbre en las estimaciones cuantitativas establecidas por recursos, coste y planificación temporal.
* Si no se entiende bien el ámbito de proyecto o los requisitos del proyecto están sujetos a cambios, la incertidumbre y el riesgo son peligrosamente altos. El planificador del software debería solicitar definiciones completas de rendimiento y de interfaz (dentro de una especificación del sistema).
* El planificador y, lo que es más importante, el cliente, deben tener presente que cualquier cambio en los requisitos del software significa inestabilidad en el coste y en la planificación temporal.

Fiabilidad de la estimación

* La complejidad del proyecto
* El tamaño del proyecto
* El grado de incertidumbre estructural

*Afectan la fiabilidad de la estimación.*

Factores críticos de éxito

* Soporte Ejecutivo.
* Involucramiento del cliente.
* Gerente de Proyecto experimentado.
* Objetivos de negocios claros.
* Alcances claros y acotados.
* Infraestructura tecnológica probada.
* Requerimientos consolidados.
* Proceso formal y maduro de desarrollo.
* Estimaciones realistas.

Estimaciones

* No son simples, pero se necesitan a menudo.
* Una organización solo mejora en sus estimaciones cuando las usa:

*Falacia: como estimo mal, no vale la pena hacerlo.*

* Creadas, usadas y refinadas durante:
  + Planeamiento estratégico.
  + Factibilidad/SOW/Propuesta.
  + Evaluación de Sub-Contratistas.
  + Planeamiento básico de proyecto (iterativo).
* Una “estimación exacta” es una contradicción.
* Integrar:
  + Factores discernibles que afectan la estimación: tamaño, complejidad, capacidades, experiencias anteriores.
* Ningún proyecto soporta ser estimado en el “peor caso”.
* Tampoco en el “mejor caso”.
* Ideal:
* En nuestra industria las estimaciones suelen estar corridas en .
* La calidad de las estimaciones obviamente hace a la calidad del proyecto, pero las estimaciones se deben hacer en un momento donde la cantidad y calidad de la información es limitada.
* La estimación es muy precisa al final, pero ahí no se necesita.
  + Es importante capturarla para el proyecto siguiente.
* Las mejores estimaciones se basan en la historia.
* **Sobre-estimación:**
  + Ley de Parkinson (los seres humanos tienden a ocupar todo el tiempo asignado para una tarea), Costo inviable.
* **Infra-estimación:** escasez de recursos.
* Cambios: tecnología, nuevas funciones.

Objetivo de la planificación

* Proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, costo y **planificación temporal**.
* Estas estimaciones se hacen dentro de un marco de tiempo limitado al comienzo de un proyecto de software, y deberían actualizarse regularmente a medida que progresa el proyecto.
* Las estimaciones deberían definir los escenarios del **mejor caso y peor caso** de forma que los resultados del proyecto puedan limitarse.
* Cuanto más sepa, mejor realizará la estimación. Por consiguiente, actualice sus estimaciones a medida que progresa el proyecto.

Ámbito del Software

* La primera actividad de la planificación del proyecto de software es determinar el ámbito del software. Se debe delimitar la declaración del ámbito del software.
* El ámbito del software describe el control y los datos a procesar, la función, el rendimiento, las restricciones, las interfaces y la fiabilidad.
* Para esto se trabaja con las técnicas ya vistas de recolección de información de los stakeholders.

Recursos

* La segunda tarea de la planificación del desarrollo de software es la estimación de los recursos requeridos para acometer el esfuerzo de desarrollo de software.



* Cada recurso queda especificado mediante cuatro características:
  + Descripción del recurso
  + Informe de disponibilidad
  + Fecha cronológica en la que se requiere el recurso
  + Tiempo durante el que será aplicado el recurso
* Las dos últimas características pueden verse como una ventana temporal. La disponibilidad del recurso para una ventana específica tiene que establecerse lo más pronto posible.

Recursos Humanos

* El número de personas requerido para un proyecto de software solo puede ser determinado después de hacer una estimación del esfuerzo de desarrollo (por ejemplo, personas-mes).
* Hay que especificar tanto la posición dentro de la organización (por ejemplo: gestor, ingeniero de software experimentado, etc.) como la especialidad (por ejemplo: telecomunicaciones, bases de datos, cliente/servidor).

Recursos de Software Reutilizables

* La ingeniería del software basada en componentes destaca la reutilización, esto es, la creación y la reutilización de bloques de construcción de software.
* **Componentes ya desarrollados**. El software existente se puede adquirir de una tercera parte o provenir de uno desarrollado internamente para un proyecto anterior.
* **Componentes ya experimentados**. Especificaciones, diseños, código o datos de prueba existentes desarrollados para proyectos anteriores que son similares al software que se va a construir para el proyecto actual.
* **Componentes con experiencia parcial**. Especificaciones, diseños, código o datos de prueba existentes desarrollados para proyectos anteriores que se relacionan con el software que se va a construir para el proyecto actual, pero que requerirán una modificación sustancial.
* **Componentes nuevos.** Los componentes de software que el equipo de software debe construir específicamente para las necesidades del proyecto actual.

Recursos de Entorno

* El entorno es donde se apoya el proyecto de software, llamado a menudo entorno de IS (EIS), incorpora hardware y software.
* Cuando se va a desarrollar un sistema basado en computadora (que incorpora hardware y software especializado), el equipo de software puede requerir acceso a los elementos en desarrollo por otros equipos de ingeniería.

Estimaciones Seguras

* Dejar la estimación para más adelante (obviamente, podemos realizar una estimación al cien por cien fiable tras haber terminado el proyecto).
* Basar las estimaciones en proyectos similares ya terminados.
* Utilizar técnicas de descomposición relativamente sencillas para generar las estimaciones de coste y de esfuerzo del proyecto.
* Utilizar uno o más modelos empíricos para la estimación del coste y esfuerzo del software.

*Desgraciadamente, la primera opción, aunque atractiva, no es práctica.*

Dejar las estimaciones para más adelante

* ¿Lo vemos como opción factible? ¿Cómo organizamos un proyecto si no estimamos?
* Debemos realizar las estimaciones, sabiendo que contamos con pocos datos, y un alto grado de incertidumbre en muchos casos.

Basar las estimaciones en proyectos similares

* Puede funcionar bien si el proyecto actual es bastante similar a los pasados.
* Normalmente es muy difícil encontrar otro proyecto similar, no solo en las características básicas, sino en cuanto al cliente, personal involucrado, etc.

Estimaciones seguras

* Las opciones restantes son métodos viables para la estimación del proyecto de software.
* Desde un punto de vista ideal, se deben aplicar conjuntamente las técnicas indicadas; usando cada una de ellas como comprobación de las otras.
* Las **técnicas de descomposición** utilizan un enfoque de **divide y vencerás** para la estimación del proyecto.
* Se pueden utilizar los **modelos empíricos** de estimación como complemento de las técnicas de descomposición, ofreciendo un enfoque de estimación potencialmente valioso por derecho propio.
* Cada modelo se basa en la experiencia (datos históricos) y toma como base:

Donde es uno de los valores estimados (por ejemplo, esfuerzo, coste, duración del proyecto) y los son determinados parámetros independientes (por ejemplo, LDC o PF estimados).

* Las **herramientas automáticas de estimación** implementan una o varias técnicas de descomposición o modelos empíricos.
* Cuando se combinan con una interfaz gráfica de usuario, las herramientas automáticas son una opción atractiva para la estimación. En sistemas de este tipo, se describen las características de la organización de desarrollo (por ejemplo, la experiencia, el entorno) y el software a desarrollar.
* De estos datos se obtienen las estimaciones de coste y de esfuerzo.
* Cada una de las opciones viables para la estimación de costes del software, solo será buena si los datos históricos que se utilizan como base de la estimación son buenos. Si no existen datos históricos, la estimación del coste descansará sobre una base muy inestable.

Técnicas de Descomposición

La precisión de una estimación del proyecto de software se predice basándose en una serie de cosas:

1. El grado en el que el planificador ha estimado adecuadamente el tamaño del producto a construir.
2. La habilidad para traducir la estimación del tamaño en esfuerzo humano, tiempo y dinero (una función de la disponibilidad de métricas fiables de software de proyectos anteriores).
3. El grado en el que el plan del proyecto refleja las habilidades del equipo de software.
4. La estabilidad de los requisitos del software y el entorno que soporta el esfuerzo de la ingeniería del software.

Tamaño del software

* El tamaño representa el primer reto importante del planificador de proyectos.
* En el contexto de la planificación de proyectos, el tamaño se refiere a una producción cuantificable del proyecto de software.
* Si se toma un enfoque directo, el tamaño se puede medir en LDC.
* Si se selecciona un enfoque indirecto, el tamaño se representa como PF.

Putnam y Myers sugieren cuatro enfoques diferentes del problema del tamaño:

* **Tamaño en lógica difusa**. Este enfoque utiliza las técnicas aproximadas de razonamiento que son la piedra angular de la lógica difusa. Para aplicar este enfoque, el planificador debe identificar el tipo de aplicación, establecer su magnitud en una escala cuantitativa y refinar la magnitud dentro del rango original.
* **Tamaño en punto de función.** El planificador desarrolla estimaciones de características del dominio de información.
* **Tamaño de componentes estándar.** El software se compone de un número de componentes estándar que son genéricos para un área en particular de la aplicación. Por ejemplo, los componentes estándar para un sistema de información son: subsistemas, módulos, pantallas, informes, programas interactivos, programas por lotes, archivos, LDC e instrucciones para objetos. El planificador de proyectos estima el número de incidencias de cada uno de los componentes estándar, y utiliza datos de proyectos históricos para determinar el tamaño de entrega por componente estándar.
* **Tamaño del cambio**. Este enfoque se utiliza cuando un proyecto comprende la utilización de software existente que se debe modificar de alguna manera como parte de un proyecto. El planificador estima el número y tipo (por ejemplo: reutilización, añadir código, cambiar código, suprimir código) de modificaciones que se deben llevar a cabo. Mediante una proporción de esfuerzo para cada tipo de cambio, se puede estimar el tamaño del cambio.

Estimación basada en el problema

Los datos de LDC y PF se utilizan de dos formas durante la estimación del proyecto de software:

* Como una variable de estimación que se utiliza para **dimensionar** cada elemento del software.
* Como métricas de línea base recopiladas de proyectos anteriores y utilizadas junto con variables de estimación para desarrollar proyecciones de coste y de esfuerzo.
* Las métricas de productividad de línea base (por ejemplo: LDC/pm o PF/pm [por mes]) se aplican entonces para la variable de estimación adecuada y se extrae el coste o el esfuerzo de la función.
* Las estimaciones de función se combinan para producir una estimación global del proyecto entero.
* Los proyectos se deberían agrupar por tamaño de equipo, área de aplicación, complejidad y otros parámetros relevantes. Entonces se deberían calcular las medias del dominio local.
* Cuando se estima un proyecto nuevo, primero se debería asignar a un dominio, y a continuación utilizar la media del dominio adecuado para la productividad al generar la estimación.
* Las técnicas de estimación de LDC y PF difieren en el nivel de detalle que se requiere para la descomposición y el objetivo de la partición.
* Con independencia de la variable de estimación que se utilice, el planificador del proyecto comienza por estimar un rango de valores para cada función o valor del dominio de información.
* Mediante los datos históricos o (cuando todo lo demás falla) la intuición, el planificador estima un valor de tamaño optimista, más probable y pesimista para cada función, o cuenta para cada valor del dominio de información. Al especificar un rango de valores, se proporciona una indicación implícita de grado de incertidumbre.

Valor esperado para el tamaño del software

Entonces se calcula un valor de tres puntos o esperado.

* El valor esperado de la variable de estimación (tamaño), , se puede calcular como una media ponderada de las estimaciones optimistas (), las más probables () y las pesimistas ().
* Se asume que existe una probabilidad muy pequeña en donde el resultado del tamaño real quedará fuera de los valores pesimistas y optimista
* Una vez que se ha determinado el valor esperado de la variable de estimación, se aplican datos históricos de productividad de LDC y PF.
* ¿Son correctas las estimaciones?
* La única respuesta razonable a esto es: **no podemos estar seguros**. Cualquier técnica de estimación, no importa lo sofisticada que sea, se debe volver a comprobar con otro enfoque incluso entonces, va a prevalecer el sentido común y la experiencia.

Estimación basada en el proceso

* La técnica más común para estimar un proyecto es basar la estimación en el proceso que se va a utilizar.
* Es decir, el proceso se descompone en un conjunto relativamente pequeño de actividades o tareas, y en el esfuerzo requerido para llevar a cabo la estimación de cada tarea.
* Al igual que las técnicas basadas en problemas, la estimación basada en el proceso comienza con un esbozo de las funciones del software obtenidas a partir del ámbito del proyecto. Para cada función se debe llevar a cabo una serie de actividades del proceso de software.
* Si la estimación basada en el proceso se realiza independientemente de la estimación de LDC o PF, ahora tendremos dos o tres estimaciones del coste y del esfuerzo que se pueden comparar y evaluar.
* Si ambos tipos de estimaciones muestran una concordancia razonable, hay una buena razón para creer que las estimaciones son fiables.
* Si, por otro lado, los resultados estas técnicas de descomposición muestran poca concordancia, se debe realizar más investigación y análisis.

Muchas divergencias entre estimaciones se deben a una de dos causas:

* No se entiende adecuadamente el ámbito del proyecto o el planificador lo ha mal interpretado.
* Los datos de productividad usados para técnicas de estimación basados en problemas no son apropiados para la aplicación, están obsoletos (ya no reflejan con precisión la organización de la ing. de software), o se han aplicado erróneamente.
* El planificador debe determinar la causa de la divergencia y conciliar las estimaciones.

Modelos Empíricos de Estimación

* Un modelo de estimación para el software de computadora utiliza fórmulas derivadas empíricamente para predecir el esfuerzo como una función de LDC o PF.
* Los datos empíricos que soportan la mayoría de los modelos de estimación se obtienen de una muestra limitada de proyectos. Por esta razón, el modelo de estimación no es adecuado para todas las clases de software y en todos los entornos de desarrollo. Por consiguiente, dos resultados obtenidos de dichos modelos se deben utilizar con prudencia.

Estructura de los modelos de estimación

* Un modelo común de estimación se extrae utilizando el análisis de regresión sobre los datos recopilados de proyectos de software anteriores.
* La estructura global de tales modelos adquiere la forma de:

donde , y son constantes obtenidas empíricamente, es el esfuerzo en personas-mes, y es la variable de estimación (de LDC o PF).

* La mayoría de los modelos de estimación tienen algún componente de ajuste del proyecto que permite ajustar E por otras características del proyecto (por ejemplo: complejidad del problema, experiencia del personal, entorno de desarrollo).
* Entre los muchos modelos de estimación orientados a LDC propuestos, se encuentran los siguientes:
  + Modelo de Walston-Felix
  + Modelo de Bailey-Basili
* También se han propuesto los modelos orientados a PF. Entre estos modelos se incluyen:
  + Modelo de Albretch y Gaffney
  + Modelo de Kemerer
  + Modelo de Matson, Bamett y Mellichamp
* Un rápido examen de los modelos listados anteriormente indica que cada uno producirá un resultado diferente para el mismo valor de LDC y PF.

**Cocomo**

El modelo COCOMO original se ha convertido en uno de los modelos de estimación de coste del software más utilizados y estudiados en la industria. Ha evolucionado a un modelo de estimación más completo llamado COCOMO II.

Desarrollar para comprar

En muchas áreas de aplicación del software, a menudo es más rentable adquirir el software de computadora que desarrollarlo. Los gestores de ingeniería del software se enfrentan con una decisión desarrollar o comprar que se puede complicar aún más con las opciones de adquisición:

* El software se puede comprar (con licencia) ya desarrollado;
* Se pueden adquirir componentes de software totalmente experimentado o parcialmente experimentado.
* El software puede ser construido de forma personalizada por una empresa externa para cumplir las especificaciones del comprador.

Para productos de software más caros, se pueden aplicar las directrices siguientes:

* Desarrollo de una especificación para la función y rendimiento del software deseado. Definición de las características medibles, siempre que sea posible.
* Estimación del coste interno de desarrollo y la fecha de entrega.
  + Selección de tres o cuatro aplicaciones candidatos que cumplan mejor las especificaciones.
  + Selección de componentes de software reutilizables que ayudarán en la construcción de la aplicación requerida.
* Desarrollo de una matriz de comparación que presente la comparación una a una de las funciones clave. Alternativamente, realizar el seguimiento de las pruebas de evaluación para comparar el software candidato.
* Evaluación de cada paquete de software o componente según la calidad de productos anteriores, soporte del vendedor, dirección del producto, reputación, etc.
* Contacto con otros usuarios de dicho software y petición de opiniones.

Outsourcing

* La decisión de contratar fuentes externas puede ser estratégica o táctica.
* En el nivel estratégico, los gestores tienen en consideración si una parte importante de todo el trabajo del software puede ser contratado a otros.
* En el nivel táctico, un jefe de proyecto determina si algunas partes o todo el proyecto es aconsejable realizarlo mediante subcontratación.
* Como **ventajas**:
  + Los ahorros de coste se pueden lograr reduciendo el número de personas y las instalaciones (por ejemplo: computadoras, infraestructura) que los apoyan.
* Y como **desventajas:**
  + Una compañía pierde control sobre el software que necesita.
  + Como el software es una tecnología que diferencia sus sistemas, servicios, y productos, una compañía corre el riesgo de poner su destino en manos de un tercero.

Herramientas Automáticas de Estimación

Las técnicas de descomposición y los modelos empíricos de estimación descritos en las secciones anteriores se pueden implementar con software.

Todos realizan las seis funciones genéricas mostradas a continuación:

* Dimensionamiento de las entregas del proyecto. Se estima el tamaño de uno o más productos de software.
* Selección de las actividades del proyecto. Se selecciona el marco de trabajo del proceso adecuado.
* Predicción de los niveles de la plantilla. Se especifica el número de personas disponibles para realizar el trabajo.
* Predicción del esfuerzo del software.
* Predicción del coste del software.
* Predicción de la planificación del software.

**Unidad Nº 5: Gestión de la calidad**

**Calidad**

La calidad está asociada con la satisfacción de alguien.

Problemas en los proyectos

* Atrasos en las entregas
* Falta cumplimiento de los compromisos
* No están claros los requerimientos
* El software/sistema no hace lo que tiene que hacer
* Trabajo fuera de hora
* ¿A dónde está esa rutina?

Preguntas acerca de la calidad

* ¿Hace lo que el usuario quiere?
* ¿Le soluciona el problema que intenta resolver?
* ¿Lo hace como él quiere?
* ¿Es factible de construir?, ¿de corregir, de expandir, de mejorar?
* ¿Se lo puede construir rápido, barato, y en forma segura?
* ¿Se puede todo a la vez?
* Lo modifiqué, ¿Sigue funcionando bien?
* ¿Me gusta?

Gestión de la configuración del SW

* La gestión de configuración del software (GCS) es una **actividad de autoprotección.**
* Como el cambio se puede producir en cualquier momento, las actividades de GCS sirven para:
  + Identificar el cambio.
  + Controlar el cambio.
  + Garantizar que el cambio se implementa adecuadamente.
  + Informar del cambio a todos aquellos que puedan estar interesados.

No hay calidad cuando

* El sistema cancela.
* No hace algo que el usuario pidió.
* Hace mal algo.
* Hace algo que el usuario aún no pidió.
* Es lento.
* Es difícil de usar.
* Es fácil cometer errores con él.
* Es “sensible”.
* Cuesta mucho hacerlo.
* Cada cambio es difícil de introducir.
* Es feo.
* No sé cómo probar que funciona bien.
* Sólo yo puedo arreglarlo.
* Reinventa la rueda (empezar de nuevo).
* Es más complejo que lo necesario.

SW de calidad

Es aquel que satisface con:

* Las expectativas del Cliente y del Usuario.
* Las necesidades del equipo de desarrollo y mantenimiento.
* Otros interesados en el producto.
* Otros interesados en el proceso.

La calidad es para el cliente, el desarrollador, el usuario, el operador y el líder del proyecto.

Definición de calidad

* Cumplir con los requerimientos (requisitos) (Crosby).
* Cumplir con los requerimientos de alguna persona (Weinberg).
  + Calidad es valor para alguna persona.
  + Valor es aquello que está dispuesto a pagar para obtener sus requerimientos.
* Adecuación al uso (Juran).
  + Satisfacción de las necesidades del cliente.
  + Ausencia de deficiencias.
* La totalidad de aspectos y características de un producto o servicio que se sustentan en su capacidad de cumplir las necesidades especificadas o implícitas (ISO 8402-1986)

Visiones de la calidad

* Trascendente
  + Ve la calidad como algo que se puede reconocer, pero no definir
* De Usuario
  + Adecuación para su uso:
    - Más concreta.
    - Basada en las características del producto que satisfacen las necesidades del usuario.
    - Depende de un contexto de uso.
    - Mejor calidad para alguna persona puede ser peor para otra.
  + Confiabilidad, performance, facilidad de uso.
* De Construcción
  + Mira la calidad del producto mientras se lo construye y luego de entregarlo
    - ¿Se construye bien desde el inicio?
    - Ataca los defectos y el retrabajo
    - Cumplimiento del proceso
      * ISO 9000
      * CMM
    - ¿Buen proceso ⇒ buen producto? No siempre. Es difícil tener un buen producto con un mal proceso.
* De Producto
  + Características inherentes al producto
    - Propiedades internas
      * Facilidad de Mantenimiento: ¿Puedo corregirlo?
      * Flexibilidad: ¿Puedo cambiarlo?
      * Facilidad de Prueba: ¿Puedo probarlo?
      * Corrección: ¿Hace lo que quiero?
  + Definición de McCall
    - Usabilidad.
    - Integridad.
    - Eficiencia.
    - Correctitud.
    - Confiabilidad.
    - Facilidad de Mantenimiento.
    - Facilidad de Prueba.
    - Flexibilidad.
    - Facilidad de Reúso.
    - Portabilidad.
    - Interoperabilidad.
  + Definición de Boehm
    - Portabilidad.
    - Eficiencia.
    - Ingeniería persona máquina.
    - Facilidad de prueba.
    - Facilidad de aprendizaje.
    - Facilidad ante modificaciones.
    - Confiabilidad.
* De Valor Agregado

**Aseguramiento de la calidad**

* Es el diseño y planificación de acciones sistemáticas que se requieren para “ASEGURAR” la calidad del Software.
* Responsables
  + Ingenieros de software.
  + Gerentes y Líderes de Proyecto.
  + Clientes.
  + Comerciales.
  + Grupo de SQA.

Propiedades

* Las visiones permiten definir modelos de Calidad (conjunto de características).
* Estos modelos permiten definir propiedades del producto y del proceso.
* En base a estas propiedades definimos métricas (indicadores).
* En base a las métricas realizamos la Gestión de Calidad para Asegurarla.

Características

* Funcionalidad.
* Confiabilidad.
* Usabilidad.
* Eficiencia.
* Mantenibilidad.
* Portabilidad.

Principios

* La calidad no se ‘inyecta’ ni compra.
* Es un esfuerzo (y el negocio) de todos.
* La administración de los recursos humanos es muy importante.
* Se necesita soporte gerencial
  + Pero se puede empezar por uno.
* Se debe liderar con el ejemplo.
* No se puedo controlar lo que no se mide.
* Simplicidad.

Aseguramiento de la calidad



* El propósito de SQA es suministrar a la gerencia de una visibilidad dentro del proceso que está siendo usado por el proyecto y de los productos que se construyen. (CMM)
* SQA implica la revisión y auditoría de los productos y actividades para verificar que cumplen con los procedimientos y estándares aplicables y suministrando a los gerentes de proyecto y otros del resultado de estas revisiones y auditorías.
* ¿Es sólo esto?
  + SQA software quality assurance

**Proceso**

La calidad del software está fuertemente determinada por la calidad del proceso usado para desarrollarlo y mantenerlo.

Proceso de desarrollo del SW

* Es la aplicación de un método para el desarrollo de software por parte de las personas con el entrenamiento adecuado.
* No debemos confundir la definición de un proceso con el proceso real:
  + La definición del proceso es la expresión de cómo deseamos trabajar.
  + El proceso real es el modo en que efectivamente hacemos el trabajo.

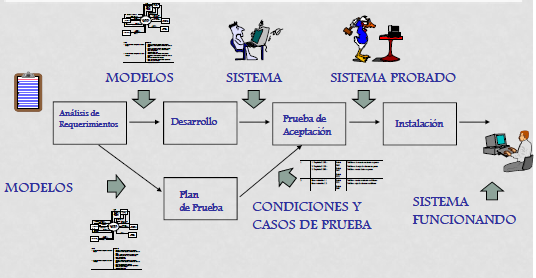
Proceso

* Conjunto de herramientas, métodos, y prácticas que usamos para construir un producto [Humphrey, Managing the Software Process].
* Secuencia de pasos encadenados con el fin de construir un producto u ofrecer un servicio.

Cuando no hay proceso

* Las actividades del proceso no están definidas:
  + por lo tanto, no podremos planearlas.
  + por lo tanto, no podremos controlarlas.
  + por lo tanto, no podremos asegurar los resultados.
* ‘Los participantes harán lo que puedan para resolverlo’

Cómo es un proceso para un proyecto



Soporte al proceso

Ingeniería de Requerimientos

Desarrollo

Prueba de Aceptación

Instalación

Plan de Prueba

MODELOS

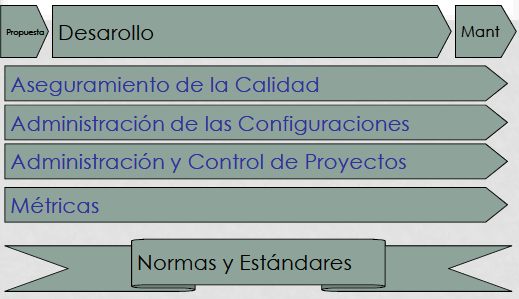
SISTEMA

SISTEMA PROBADO

SISTEMA FUNCIONANDO

CONDICIONES Y CASOS DE PRUEBA

* Administración y control de proyectos
* Administración de configuraciones
* Aseguramiento de la calidad

Estructura del aseguramiento de la calidad

Se lleva a cabo durante todo el ciclo de desarrollo

Costo de no tener calidad

* Del usuario
  + Mayor costo de administración.
  + No poder hacer ciertas cosas.
  + No poder llegar a tiempo al mercado.
  + Perder imagen con el cliente.
  + No poder confiar en la herramienta.
* De sistemas
  + Mantenimiento fuera de hora.
  + Nunca cosas nuevas.
  + Estar sobrecargado.
  + No poder cumplir plazos.
  + Recibir quejas del usuario.
  + Perder credibilidad.
  + Horas extras.

Economía de la calidad

* Un producto de más calidad
  + tiene menos errores.
  + tardo menos en hacer que funcione.
  + tiene menos fallas, tardo menos en arreglarlo.
  + el usuario está más contento.
  + por lo tanto, le dedico menos tiempo y esfuerzo.
* **Cuesta menos y tardo menos.**
* **Recuerde que un proyecto ‘normal’ tiene más del 43% del esfuerzo gastado en retrabajo.**

SW de calidad

* Aquel que cumpla con los requisitos.
* Es decir que al ser usado no tenga fallas.

Error:

* La diferencia entre un valor o condición computada, observada, o medida y el valor o condición verdadera, especificada, o teóricamente correcta.
* Para conocer los errores debemos conocer los resultados correctos o esperados.

Definiciones de error

* Equivocación
  + Una acción humana que produce un resultado incorrecto.
* Defecto
  + Paso, proceso, o definición de dato incorrecto.
  + Ausencia de cierta característica.
* Falla
  + Resultado de ejecución incorrecto. Es el producido por el software distinto al resultado esperado.

V & V

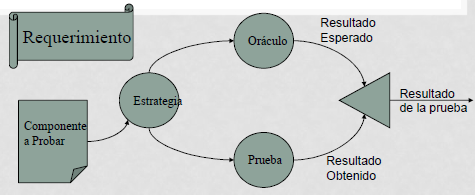
* Validación
  + ¿Estamos haciendo el producto correcto? Prueba el cliente.
* Verificación
  + ¿Estamos haciendo el producto correctamente? Test, casos de prueba.
* Algunas Reflexiones
  + ¿Qué acciones haría para verificar?
  + ¿Qué acciones haría para validar? El documento de requerimientos se valida con el cliente.
  + ¿Las pruebas son validación o verificación?

**Gestión de la calidad**

Pruebas de Software

* Probar es ejecutar un programa con el objetivo de encontrar defectos en él.
* La dura realidad de las pruebas
  + “Las pruebas sólo pueden mostrar la presencia de defectos, no su ausencia”.
  + Una prueba es exitosa si encuentra defectos.
  + Las pruebas encuentran Fallas, no defectos.

**Proceso de prueba**



Las pruebas siempre deben realizarse contra un resultado esperado.

Los resultados deben ser revisados minuciosamente.

Economía de la prueba

* Se puede invertir mucho esfuerzo en probar
* Probar es el proceso de establecer confianza en que un programa hace lo que se supone que tiene que hacer
  + Ya que nunca voy a poder demostrar que un programa es correcto.
  + Continuar probando es una decisión económica.

Muchas organizaciones concentran sus esfuerzos en arreglar los errores en vez de prevenirlos.

**Unidad Nº 6: Gestión de la Configuración**

*“El arte de coordinar el desarrollo de software para minimizar... la confusión, se denomina* ***gestión de la configuración****. La gestión de configuración es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo de programación. La meta es maximizar la productividad minimizando los errores.”* (Babich, 1986)

**Problemas actuales**

* El software moderno es crecientemente complejo.
* Aplicaciones enormes y heterogéneas.
* Muchos grupos de trabajo trabajando concurrentemente.
* Aplicaciones con ciclos de vida muy largos.
* La misma aplicación núcleo satisface múltiples objetivos.
* Ambiente con múltiples instancias configurables.
  + Lenguajes, Industria, Clientes, Versiones Premium, Licencias.
* Cambios realizados por distintos programadores.
* ¿“Qué” versión hay que instalar en el cliente?
* Errores corregidos que reaparecen.
* Implementación de algún cambio sin estar confirmado.
* ¿Qué fuentes se corresponden con un ejecutable?
* ¿Cuál es la última versión del manual de usuario?
* ¿Está la documentación actualizada?
* Cambios se suceden continuamente afectando el software y su documentación.
* Programación y testing concurrente, corrección de errores, sustitución de fuentes.
* Composición del sistema en un momento dado: documentación, código fuente y ejecutables.
* Construcción de familias de software: distintas plataformas, funcionalidades, etc.

**Qué es GCS**

* La gestión de configuración del software (GCS) es una **actividad de autoprotección.**
* Como el cambio se puede producir en cualquier momento, las actividades de GCS sirven para:
  + Identificar el cambio.
  + Controlar el cambio.
  + Garantizar que el cambio se implementa adecuadamente.
  + Informar del cambio a todos aquellos que puedan estar interesados.

Mantenimiento y GCS

**Mantenimiento**: Es un conjunto de actividades de ingeniería del software que se producen *después* de que el software se haya entregado al cliente y esté en funcionamiento.

**Gestión de configuración del software**: Es un conjunto de actividades de seguimiento y control que comienzan cuando se inicia el proyecto de ingeniería del software y termina sólo cuando el software queda fuera de la circulación. Son actividades de seguimiento y control desde el *nacimiento* del SW hasta su *muerte*.

Configuración del SW

* Los elementos que componen *toda* la información producida como parte del proceso de ingeniería del software se denominan colectivamente ***configuración del software***.
* Información producida:
  + **Programas** de computadora (tanto en forma de código fuente como ejecutable).
  + **Documentos** que describen los programas de computadora (tanto técnicos como de usuario).
  + **Datos** (contenidos en el programa o externos a él).

El Cambio

* El cambio se puede producir en cualquier momento y por cualquier razón.
* Primera Ley de la Ingeniería de Sistemas:
  + “Sin importar en qué momento del ciclo de vida del sistema nos encontremos, el sistema cambiará y el deseo de cambiarlo persistirá a lo largo de todo el ciclo de vida.”
* La GCS es un conjunto de actividades desarrolladas para gestionar los cambios a lo largo del ciclo de vida del software de computadora.
* Fuentes del Cambio:
  + Nuevas condiciones en el negocio.
  + Nuevas necesidades del cliente.
  + La reorganización, crecimiento o reducción del negocio.
  + Restricciones presupuestarias.

Elementos de configuración del SW (ECS)

Toda información creada como parte del proceso de ingeniería de software es un elemento.

* En el extremo:
  + Cada sección individual de una gran especificación.
  + Cada caso de prueba de un gran conjunto de pruebas.
* Realidad:
  + Un documento.
  + Un conjunto completo de casos de pruebas.
  + Un componente.
  + Etc.
* Elementos componentes
  + Herramientas que permiten el acceso y gestión de cada ítem de CS.
* Elementos de proceso
  + Procedimientos y tareas para gestionar el cambio.
* Elementos de construcción
  + Herramientas que automatizan la construcción del software.
* Elementos humanos
  + Conjunto de herramientas y procesos utilizados por el equipo de trabajo.

Los elementos se organizan como *objetos de configuración* catalogados con un nombre único.

Un objeto de configuración tiene un *nombre* y *atributos* y está conectado a otros objetos mediante *relaciones*.

Ítem de configuración

* *Una agregación de hardware, software o ambos, que es designado para gestión de la configuración y tratado como una entidad simple en el proceso GCS* (IEEE 610.12-90).
* Cada uno de los elementos que están bajo Gestión de Configuración.
* No todas las entidades necesitan ser controladas todo el tiempo.
* Debe decidirse *cuándo* un elemento pasa a estar bajo control
  + Demasiado temprano → Burocracia (costo y tiempo).
  + Demasiado tarde → Caos.

**Línea Base (PARCIAL!)**

* *Una especificación o producto que se ha revisado formalmente y sobre los que se ha llegado a un acuerdo, y que de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambios* (IEEE 610.12-90).
* Un elemento de configuración se convierte en línea base si fue revisado y aprobado.
* Un **cambio** es el paso de una línea base a la siguiente.

Líneas bases típicas

* Especificación del sistema.
* Plan del Proyecto de Software.
* Especificación de Requisitos del Software.
* Especificaciones de diseño
  + Descripciones del Diseño de los Módulos.
  + Descripciones del Diseño de las Interfaces.
* Código fuente.
* Planes / Procedimientos.
* Datos de prueba.
* Sistema en funcionamiento.

**Conceptos básicos**

* Sistema
  + Colección de componentes organizados para obtener una función específica o conjunto de funciones. [IEEE 610.12-90].
* Visibilidad
  + Dada la naturaleza abstracta del software, es fácil ignorar qué contiene al evolucionar desde los requisitos al código.
  + La **visibilidad**se consigue mediante la identificación (unívoca) de todos los productos software y sus relaciones.
* Trazabilidad
  + Todas las representaciones de un producto de software a lo largo de su ciclo de vida deben ser consistentes.
  + **Traza**: dependencia que indica relación de proceso o histórica entre dos elementos que representan el mismo concepto, sin reglas para derivar el uno del otro.
* Versión (revisión)
  + “Versión” es la forma particular que adopta un objeto en un contexto dado.
  + Desde el punto de vista de evolución, es la forma particular de un objeto en un instante dado. Se suele denominar “revisión”.



* Configuración
  + Un sistema software comprende distintos componentes (hardware y software), que evolucionan individualmente.
  + Hay que garantizar la consistencia del *conjunto* del sistema.
  + Una “*configuración*” *es una combinación de versiones particulares de los componentes que forman un sistema consistente.*
  + Desde el punto de vista de evolución, *es el conjunto de las versiones de los objetos componentes en un instante dado*.



* Gestión de la Configuración (GC) (Configuration Management (CM))
  + Es una disciplina
    - *para la identificación de la configuración de un sistema* 
      * En distintos puntos del tiempo con el objetivo de controlar sistemáticamente los cambios a la configuración y mantener la integridad y trazabilidad de la configuración durante el ciclo de vida del sistema.
    - *que aplica conceptos técnicos y administrativos para*:
      * identificar y documentar características funcionales y físicas de un ítem de configuración, controlar cambios en las características, registrar y reportar procesamiento de cambios y estado de la implementación, verificar adecuación con requerimientos especificados. [IEEE 610.12-90]
* GCS o SCM (Software Configuration Management)
  + Es un *proceso* de soporte al ciclo de vida del software.
  + Es la Gestión de la Configuración aplicada específicamente al software.
  + Permite sistematizar el control de cambios de la configuración y mantener la integridad y rastreabilidad de la configuración a través del ciclo de vida del sistema de software.
  + Es una disciplina para gestionar la evolución de los productos de software, tanto desde etapas tempranas del desarrollo como durante el mantenimiento.
  + *Está relacionado con el versionamiento, pero es mucho más que eso, es la sistematización de los cambios a través de un proceso*.

Gestión de la Configuración del Software

La Gestión de Configuración del Software es un elemento importante de garantía de calidad del software. Su responsabilidad principal es el *control de cambios*.

La GCS plantea respuestas a preguntas complejas:

* ¿Cómo identifica los elementos discretos de una configuración de software?
* ¿Cómo gestiona una organización las diferentes versiones existentes de un programa (y su documentación) de forma que se puedan introducir cambios eficientemente?
* ¿Cómo controla la organización los cambios antes y después de que el software sea distribuido al cliente?
* ¿Quién tiene la responsabilidad de aprobar y de asignar prioridades a los cambios?
* ¿Cómo podemos garantizar que los cambios se han llevado a cabo adecuadamente?
* ¿Qué mecanismos se usan para avisar a otros de los cambios realizados?

Cinco tareas bases de la GCS:

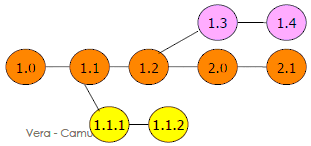
* Identificación.
* Control de versiones.
* Control de Cambios.
* Auditorías de configuración.
* Generación de informes.

Identificación

* ¿Qué elementos se van a controlar?
  + Se controlan todos los artefactos que son necesarios para el futuro mantenimiento del sistema.
  + → ***Items de configuración.***
  + Dos tipos de objetos: **básicos** y **agregados.**
  + *Objetos básicos*: unidad de información.
    - Sección de una especificación de requisitos, parte de un modelo de diseño, conjunto de casos de prueba.
  + *Objetos agregados*: colección de objetos básicos y otros objetos agregados.
* Ejemplos de elementos que se controlan:
  + Plan de proyecto.
  + Especificación de requerimientos de software.
  + Documento de arquitectura.
  + Diseño de la BD.
  + Componentes del sistema (Código).
  + Casos de prueba.
  + Manual de usuario.
  + Especificaciones de despliegue.
  + Hardware requerido.
  + Etc.
* Cada elemento consta de:
  + Nombre: Se debe asignar un único nombre a cada elemento que esté bajo la administración de la configuración. Texto sin ambigüedad.
    - “Plan de proyecto”
    - “Plan de Gestión de Configuración”
    - “Documento de definición de requerimientos”
    - “Estándares de análisis, diseño, codificación, pruebas, y auditoría”
    - El Código fuente del programa docentes.java | reportar\_faltas.php.
    - Código ejecutable. docentes.class | reportar\_faltas.php |.
    - Documentos de diseño de base de datos.
    - “Manual de usuario.
  + Descripción: lista de elementos de datos que identifican:
    - Tipo de ECS: documento, programa, datos, etc.
    - Identificador del Proyecto al que pertenece.
    - Creador del elemento.
    - Información del cambio o la versión.
    - Versión: Es una instancia de un elemento de Configuración. Ej.: CtrlCliente.java versión 1.0.
  + Lista de recursos: entidades que proporciona, procesa, referencia o son, de alguna forma, requeridas por el objeto.
    - Los tipos de datos.
    - Las funciones específicas.
    - Los nombres de las variables.
  + Relaciones: relaciones entre diferentes objetos nombrados.
    - Diagrama de Clased <*parte -de*> Modelo de Análisis.
    - Modelo de Análisis <*parte-de*> Especificación de Requisitos.
    - Especificación de Pruebas <*requiere*> Especificación de Diseño.
    - Etc.

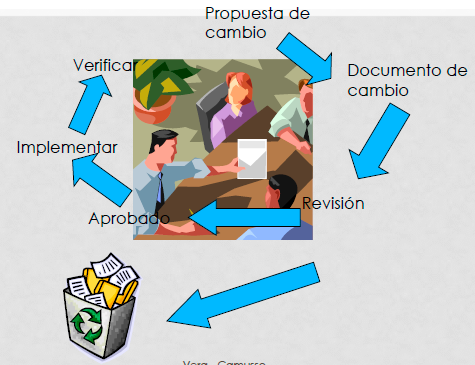
Control de versiones

* Combina *procedimientos* y *herramientas* para gestionar las *versiones* de los *elementos de configuración* creados durante el proceso de ingeniería de software.
* Sistema de Control de Versiones
  + Base de datos del proyecto.
  + Gestión de la versión de cada elemento de configuración.
  + Facilitad de construir una versión específica del software.
  + Facilidad de seguimiento de conflictos (seguimiento de bugs).
* Cada nodo del grafo de evolución es un ECS compuesto (una colección de ECSs) o una versión completa del software.
* Comúnmente se asocia un número a cada versión.
* Cada versión del SW es una colección de ECSs (fuente, documentos, datos) y cada versión puede estar compuesta por distintas *variantes*.



**Control de cambios**

* Combinación de procedimientos humanos y herramientas automatizadas para proporcionar un mecanismo para identificar, controlar el acceso y manejar cambios tanto al producto como a la línea base del proyecto.
* Objetivos
  + Establecer las versiones básicas del producto y del proyecto.
  + Proveer formas de controlar los requerimientos de cambio internos y externos que afecten las versiones básicas.
  + Asegurar que los cambios son hechos a las versiones básicas de los productos una vez que se han aprobado.
* Un proceso de administración de cambios asegura que los costos y beneficios del cambio sean adecuados y que el cambio se implemente correctamente.
* Proceso:
  + Petición del cambio.
  + Evaluación del cambio: esfuerzo técnico, efectos secundarios, impacto sobre otros componentes, costos.
  + Informes de cambios (Resultados de la evaluación) a la Autoridad de Control de Cambio (ACC) (Aprueba o Deniega)
  + Se genera una Orden de Cambio de Ingeniería (OCI) para cada cambio: qué se cambiará, restricciones, criterios de revisión y auditoría.
  + ‘Salida’ elementos de configuración de la base de datos del proyecto.
  + Realización del cambio.
  + Revisión del cambio (auditoría).



* “*El control del cambio es vital. Pero las fuerzas que lo hacen necesario también lo tornan irritante. Nos preocupamos por los cambios porque una pequeña perturbación en código puede crear una gran falla en el producto. Pero también puede resolver una gran falla o permitir maravillosas nuevas capacidades. Nos preocupamos por los cambios porque un solo desarrollador solitario podría hundir el proyecto; aunque en las mentes de dichos solitarios se originan ideas brillantes, y un proceso de control del cambio gravoso podría desalentarlos efectivamente de realizar trabajo creativo.”* (James Bach, 1998)

Auditoría de la configuración

* Verificar que, en un momento dado, el sistema en desarrollo es una colección de productos consistente y bien definida.
* Determinar que todos los elementos de configuración están presentes en la línea base del Software, estableciendo la correctitud de la versión de cada elemento de configuración.
* Prevenir problemas.

Tipos:

* Auditoría Física: verifica que el software y su documentación son internamente consistentes y están listos para ser entregados al cliente.
  + ¿Están correctamente identificados y nombrados los archivos para cada ítem de configuración?
  + ¿Han sido ubicados los ítems de configuración en el repositorio correcto?
  + ¿Existen todas las líneas bases definidas para el proyecto?
  + ¿Se han realizado los respaldos al repositorio según lo definido para el proyecto?
  + ¿Los ítems de configuración que componen la línea base han sido revisados de acuerdo al ciclo de revisión definido?
* Auditoría Funcional: evaluación independiente de los ítems de configuración para verificar que la funcionalidad y la performance sean consistentes con la ERS.
  + ¿Hay un conjunto de requerimientos asignados al release bien definidos y documentados?
  + ¿Todos los requerimientos asignados al release fueron probados o revisados?
  + ¿Todos los problemas reportados luego de las pruebas o revisiones fueron tratados debidamente?
  + ¿El sistema probado refleja el paquete del release acordado con el cliente?
* Auditoría de Rastreabilidad: verifica si a lo largo del proceso de desarrollo se mantuvo la rastreabilidad entre los componentes.

**Generación de informes**

Los informes intentan responder las siguientes preguntas:

* ¿Qué pasó?
* ¿Quién lo hizo?
* ¿Cuándo?
* ¿Qué más se vio afectado?

Informe de estado de la configuración (IEC)

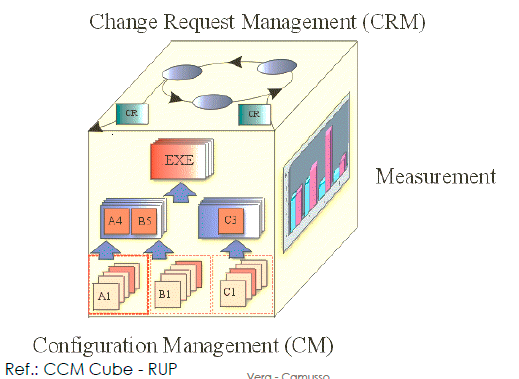
Registro y reporte de la información necesaria para gestionar eficientemente la configuración.

* Estado de cada cambio propuesto.
* Momento en que se incorpora un cambio de configuración del software.
* Deficiencias detectadas durante la auditoria de configuración del software.
* Información descriptiva de cada cambio propuesto.

**Ventajas de GCS**

* Como una herramienta de control permite:
  + Mantener la integridad de los elementos.
  + Evaluar y ejecutar los cambios en un ambiente controlado.
* Como una herramienta de visibilidad permite:
  + Evidencia objetiva y concreta de la creación y evolución del producto.
  + Que las inspecciones y auditorías de la configuración establezcan el estado de avance real del proyecto.
* Como una herramienta de reducción de costos permite:
  + La reducción de los costos de desarrollo, ayudando a mantener el “orden” en el proyecto.
  + La reducción de los costos de mantenimiento, asegurando la integridad del software en la operación y la actualización y consistencia de toda la documentación.
* Como una herramienta de apoyo a la administración del proyecto:
  + Incrementa la eficiencia y la efectividad de la administración.
  + Incrementar la efectividad de sus disciplinas.

Resumen



Ref.: CCM Cube - RUP

* La administración de configuración es una etapa de soporte en el proceso de producción de software que cubre a todas las demás áreas.
* Ayuda en la identificación de las versiones de los artefactos que componen una versión específica del sistema.
* Establece los ítems de configuración y líneas base que servirán como base para la evolución controlada del sistema.
* La administración de los cambios a los ítems de configuración se implementa a través de un procedimiento formal donde se evalúan los cambios por un Comité.
* Las herramientas automatizadas y la base de datos dan soporte al equipo de SCM para realizar sus tareas.
* Es esencial para la certificación de calidad ISO 9000, y los estándares CMM y CMMI.

**Gestión de versiones**

La gestión de versiones (VM, por las siglas de version management) es el proceso de hacer un seguimiento de las diferentes versiones de los componentes de software o ítems de configuración, y los sistemas donde se usan dichos componentes.

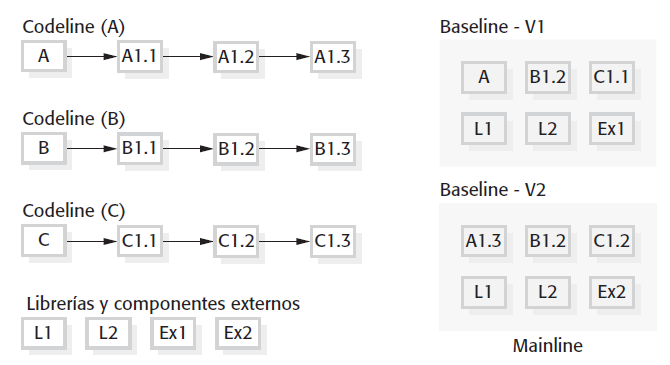
También incluye asegurar que los cambios hechos a dichas versiones por los diferentes desarrolladores no interfieran unos con otros. Por lo tanto, se puede considerar a la gestión de versiones como el proceso de administrar líneas de código y líneas base.

Configuraciones alternativas – Variantes.

Un sistema de software puede adoptar distintas formas dependiendo del lugar donde se instale.

Una *variante* es una versión de un componente.

Modelo ortogonal de versiones



En la figura se observa que diferentes líneas base usan distintas versiones de los componentes de cada línea de código. En el diagrama se sombrearon los recuadros que representan componentes en la definición línea base para indicar que en realidad son referencias a componentes en una línea de código. La línea principal es una secuencia de versiones del sistema desarrolladas a partir de una línea base original.

* Codeline: secuencia de versiones de código fuente con versiones posteriores derivadas de versiones anteriores. Aplican regularmente a componentes de sistemas, de manera que existen diferentes versiones de cada componente
* Línea base: es una definición de un sistema específico. Incluye las versiones de los componentes más las librerías utilizadas, archivos de configuración, etc.

Las líneas base son importantes porque muchas veces es necesario volver a crear una versión específica de un sistema completo. Por ejemplo, una línea de producto puede ejemplificarse de modo que existan versiones de sistema individuales para diferentes clientes. Posiblemente se tenga que volver a crear la versión entregada a un cliente específico si, por ejemplo, dicho cliente reporta bugs en su sistema que deban repararse.

* Línea principal: Secuencia de versiones de sistema desarrolladas desde una línea base original.

**Identificación de versiones**

Para crear una versión particular de un sistema se tiene que especificar la versiones de cada uno de los componentes que de incluir. Esto debe definirse una forma no ambigua. Numeración de versiones: al nombre de componente o del sistema se le añade un número de versión.

Convenciones de nomenclatura

* <id> <9>. <9>. <9>. <9> Ejemplo: Id 1.2.3.4
  + 1: reléase: se incrementa cuando se produce una modificación importante del producto.
  + 2: nivel: se incrementa cuando se produce una modificación leve en el producto.
  + 3: versión: se incrementa cuando se corrige un fallo o problema.
  + 4: instalación: se incrementa cuando se realiza una modificación debido al entorno donde se instalará.
* Dos números - MAYOR.MENOR
  + Se aumenta el número mayor cuando hay grandes cambios y, el menor, cuando ocurre un pequeño cambio. (4.3)
* También se puede agregar otro número que es el MICRO y la fase de desarrollo en que se encuentra el software (alpha, beta, etc.). Ejemplo: 1.2.1, 1.2.2-Beta
  + Se aumenta el número micro cuando se aplica una corrección de software y a su vez sufre pocos o ningún cambio.
* También se puede agregar el micro.build
  + Se cambia cuando se produce una revisión del código fuente sin cambios sustanciales.

Sin embargo, es importante aclarar que cada organización define su nomenclatura de cómo va a definir el número de la versión.

* Ubuntu utiliza como número el año y el mes de lanzamiento del software. Ejemplo: 4.10 (fue lanzado el 20 de octubre de 2004).
* Apache usa MAJOR.MINOR.PATCH.