**Unidad 1: Ingeniería del Software**

¿Qué es la Ingeniería?

La ingeniería es la aplicación de conocimiento científico para resolver problemas, de significado práctico inmediato, que tienen requerimientos y restricciones conflictivas.

DRAE

* Ingeniería: conjunto de conocimientos y técnicas cuya aplicación permite la utilización racional de los materiales y de los recursos, mediante invenciones, construcciones u otras realizaciones provechosas para el hombre.
* Ingeniero: persona que profesa o ejerce la ingeniería.

¿Qué es el Software?

“Programas de computadoras, procedimientos y posible documentación asociada y datos pertinentes a la operación de un sistema de computadora”

IEEE Standad Glossary of Software Engineering Terminology; IEEE Std 610.12-1990.

* Las instrucciones (programas) que proporcionan las características y funciones.
* Las estructuras de datos que le permiten manipular información.
* Los documentos que describen la operación y uso.

Software: la parte más compleja de los Sistemas de Información.

* Información = principal activo de las empresas.
* Desarrollo de SI <- fuertes presiones (calidad, productividad).
* Inicialmente se realizaba el desarrollo de manera artesanal. Ahora, es una Disciplina de ingeniería: calidad, herramientas, gestión de proyectos.

Atributos de un buen software

* Dependen obviamente de su aplicación, cada uno tendrá características específicas.
* Un software bien diseñado deberá tener alguno de los siguientes atributos: Mantenibilidad, Confiabilidad, Eficiencia, Usabilidad.

Mitos del Software

“En ausencia de normas significativas, una industria nueva como el software suele depender de las costumbres” (Tom DeMarco).

**Mitos del Administrador**:

* Nuestros procedimientos y estándares nos proporcionan todo el conocimiento necesario.
* Cuando el cronograma está retrasado, siempre es posible contratar nuevos programadores para terminar a tiempo.
* Si se decide subcontratar el proyecto de software me puedo relajar y esperar a que lo construya.

**Mitos del Cliente**:

* Un enunciado general de los objetivos es suficiente para comenzar a escribir programas, los detalles se pueden afinar más adelante.
* Los requerimientos de los sistemas cambian continuamente, pero el cambio puede ajustarse con facilidad porque el software es flexible.

**Mitos del Desarrollador**:

* Una vez que el programa fue escrito y puesto a funcionar, el trabajo está terminado.
* Mientras un programa no se está ejecutando, es imposible evaluar su calidad.
* El único producto del trabajo que puede entregarse para tener un proyecto exitoso es el programa en funcionamiento.
* La ingeniería del software obligará a generar una documentación voluminosa e innecesaria y de manera invariable hará más lento el proceso.

Cuestiones que se plantean

* ¿Por qué tarda tanto la obtención del software terminado?
* ¿Por qué son tan altos los costos de desarrollo de software?
* ¿Por qué es imposible encontrar todos los errores en el software antes de entregarlo al cliente?
* ¿Por qué lleva tanto tiempo y esfuerzo el mantenimiento de sistemas existentes?
* ¿Por qué es tan difícil medir el progreso al desarrollar y darle mantenimiento a los sistemas?

**Ingeniería de Software**

* Es una disciplina ingenieril, donde se aplican herramientas, métodos y procesos para generar un producto de software de calidad.
* Es el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales. Fritz Bauer 1969.
* La ingeniería de software es:

1) la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software.

2) El estudio de enfoques según el punto 1. IEEE 1993.

Herramientas

* Son instrumentos o sistemas automatizados que dan soporte a las actividades de producción de software.
* Se han desarrollado herramientas que soportan los diferentes métodos.
  + Herramientas CASE (Ingeniería del Software Asistida por Computadora): Upper CASE, Lower CASE.

Métodos

Indican cómo construir técnicamente el software.

* Descripción del modelo del sistema: descripciones de los modelos del sistema que se desarrollará y la notación empleada para definir estos modelos. Modelo de objetos, máquina de estados, etc.
* Reglas: restricciones que siempre aplican a los modelos de sistemas.
* Recomendaciones: heurísticas que caracterizan unas buenas prácticas de diseño en este método.
* Guías en el proceso: descripciones de las actividades que deben seguirse para desarrollar los modelos del sistema y la organización de estas actividades.

Procesos

* Un proceso de software es una secuencia de actividades para la producción de software.
* Definen:
  + La secuencia en la que se aplican los métodos.
  + Las entregas requeridas.
  + Los controles: ayudan a asegurar la calidad y coordinar los cambios.
  + Las directrices: ayudan a los gestores del software a evaluar el progreso.
* Son la combinación de los métodos y las herramientas que en forma conjunta dan un resultado particular.

Procesos de Software

Es un conjunto de actividades y resultados que producen un producto de software. Estas actividades las llevan a cabo los ingenieros de software y son: especificaciones del software, desarrollo del software, validación del software y evolución del software.

Breve Historia del Software

* **1º Generación del software** (¿? – 1965)
  + Hardware de propósito general
  + Software como algo añadido
  + Desarrollo a medida
  + Ninguna planificación
  + Orientación por lotes
  + Desarrollo a base de “prueba y error”
* **2º Generación del software** (1965 - 1975)
  + Sistemas multiusuario
  + Interactividad (tiempo real)
  + Almacenamiento y bases de datos
  + La industria del software ( “software house”)
  + Software de gran volumen
  + Mantenimiento
* **3º Generación del software** (1975 - 1985)
  + Microprocesadores, PCs
  + Sistemas distribuidos
  + Hardware de bajo costo
  + Aparecen las redes locales y redes globales
  + Industria planetaria
  + Complejidad en los Sistemas de Información
* **4º Generación del software** (1985 - 1995)
  + Tecnologías Orientadas a Objeto
  + Interfaces gráficas de usuario
  + Sistemas expertos, redes neuronales, IA
  + Proceso paralelo
  + Tecnologías de componentes
  + COTS (Commercial Off-The-Shelf)
  + Internet y Servicios Web
* **5º Generación del software** (1995 - continúa)
  + Aumenta la omnipresencia de la Web
  + Reutilización de la Información
  + Reutilización de componentes de Software
  + Sistemas móviles
  + Portabilidad
  + Información disponible en todo momento y en todo lugar

**¿Por qué surge la IS?**

Crisis del Software

* Entre la década del 60 al 80 se comienzan a producir los primeros problemas en el proceso de desarrollo de software.
* ¿Por qué se produce esta crisis?
* Creciente demanda, complejidad y desafíos. Misma fuerza de trabajo, métodos y herramientas.

Inversión en Desarrollo de Software

* 47 %: entregado, pero nunca usado.
* 29.7%: pagado, pero nunca entregado.
* 19%: usado, pero con trabajo extra o abandonado después.
* 3%: usado después de cambios.
* 2%: usado tal como se entregó.
* Año 1979. Total: $6.8 millones

Críticas a los desarrollos del software

* Retrasos no previstos
* Desbordamiento de costos
* Software no acorde con los requisitos
* Errores en los programas
* Sensibilidad a los errores humanos y a las averías físicas
* Dificultad de puesta en marcha
* Dificultad de evolución
* Mantenimiento desastroso

Caso emblemático: Aeropuerto Denver

* Nuevo aeropuerto de Denver (’90s)
* El sistema de manejo de equipaje subterráneo: casi 34 km. de cintas transportadoras y 4000 telecars independientes para 20 aerolíneas.
* Un sistema central de 100 PC en red, 5000 sensores eléctricos, 400 receptores de radio y 56 lectores de barra.
* Por nueve meses no estuvo en funcionamiento por errores en el sistema de control.
* Costo: más del doble de lo estimado (193 millones de dólares).
* Pospuso la apertura del aeropuerto por 8 meses. De octubre a mayo, el costo: 1,1 millón por día.

Otros ejemplos

* Therac-25: 6 personas expuestas a dosis masivas de radiación (15000 a 20000 rads en vez de 200), se cree que 3 murieron por esta causa.
* Arianne5: el cohete explotó en su despegue. 7B del programa, 500MM cohete más su carga (4 satélites). Bug por overflow no tratado al mover un número de punto flotante de 64 bits a un entero con signo de 16 bits. Ni siquiera debería haber estado corriendo…
* Problema en el Pentium (1994). Problema en un for loop causaba cálculo incorrecto de divisiones. 400 MM de costo.
* Misil Patriot: cálculo incorrecto del tiempo desde booteo por error de redondeo (0,34 segundos en 100 horas, más de 500 metros)
* Larga Distancia AT&T falla por 9 horas. Break en un switch de C.

Otro hito: No silver bullet

* Paper magistral de Brooks en Abril de 1987.
* Fred Brooks: Ingeniero en software. Trabajó en IBM, entre otros proyectos, en el desarrollo del OS/360. En 1964 abandona IBM para fundar el departamento de Computación científica en la universidad de Chapel Hill, que dirigió por 20 años.
* Dos tipos de problemas:
  + Esenciales: inherentes a la naturaleza del software.
  + Accidentales: relacionadas a la producción de software, pero que no son inherentes a él.

Números en 1991

* 52% de los proyectos se terminaron con éxito. (2% del 79).
* 31,1% se terminó con algunas modificaciones respecto de lo acordado inicialmente (3% del 79)
* 16,6% de los proyectos fue abandonado (76% del 79).

**La complejidad del Software**

*“La complejidad del software es una propiedad esencial, no una propiedad accidental” (Brooks)*

Motivos que llevan a que el software sea complejo

* Complejidad del dominio del problema
  + Imagen que tiene el cliente del dominio del problema.
  + Imagen que tiene el desarrollador del dominio del problema.
  + El dominio del problema en sí.
* La dificultad de la gestión del proceso de desarrollo.
* La flexibilidad del desarrollo de software
  + Necesidad de grandes labores de abstracción.
  + Falta de estándares.
* Problemas en la caracterización del comportamiento de sistemas discretos
  + Gran volumen de variables.
  + Interacciones entre las mismas.

Particularidades del Software

* El producto software es enteramente conceptual.
* No tiene propiedades físicas como peso, color o voltaje, y, en consecuencia, no está sujeto a leyes físicas o eléctricas.
* Su naturaleza conceptual crea una distancia intelectual entre el software y el problema que el software resuelve.
* Es difícil para una persona que entiende el problema entender el sistema software que lo resuelve.
* Para probar es necesario disponer de un sistema físico.
* El mantenimiento no es sólo una substitución de componentes.

La Industria del Software

El software es una industria, no es un arte.

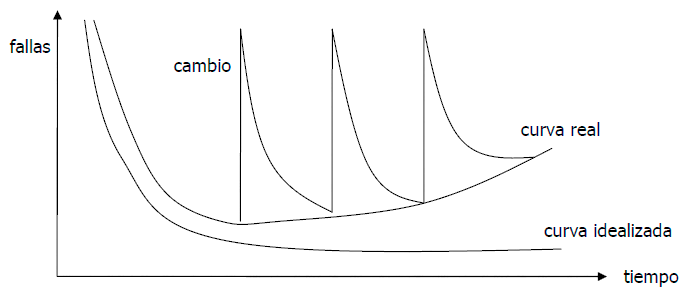
La importancia de la IS

* La economía de todos los países desarrollados depende del software, representando cada vez un mayor porcentaje de su PBI.
* Cada vez son más los sistemas controlados por software.
* Los costos del software llegan, en ocasiones, a dominar los costos de todo el sistema.

**Productos de Software**

* Productos Genéricos: productos que son producidos por una organización para ser vendidos al mercado.
* Productos hechos a medida: sistemas que son desarrollados bajo pedido a clientes específicos.
* La mayor parte del gasto del software es en productos genéricos, pero hay más esfuerzo en el desarrollo de los sistemas hechos a medida.

Características del Software

* El software se desarrolla, **no** se fabrica/manufactura. Los costos del software se concentran en la ingeniería.
* El software no se desgasta, pero sí se “deteriora”.
* La mayoría del software se construye a medida. La reutilización de componentes recién ha empezado.

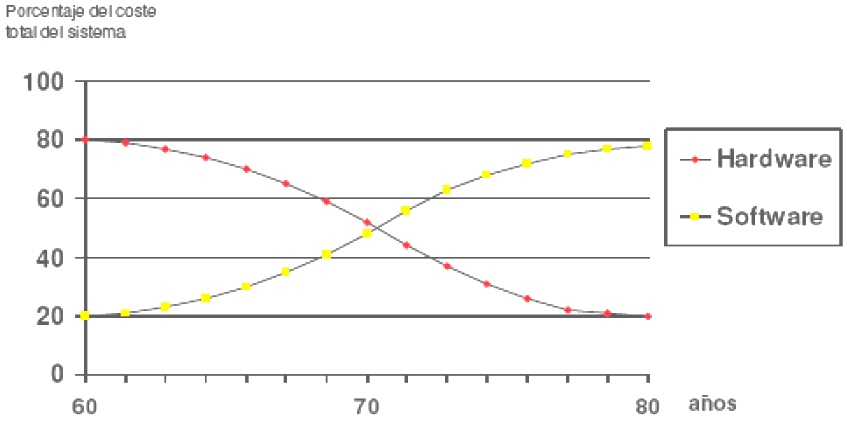
Características de los productos de Software

* **Mantenible**: debe ser posible que el software evolucione y que siga cumpliendo con sus especificaciones.
* **Confiable**: que el software funcione libre de fallas en un entorno determinado y durante un tiempo específico.
* **Eficiente**: el software no debe desperdiciar los recursos del sistema. Debe usarlos adecuadamente (memoria, tiempos de respuesta, etc.)
* **Interfaz adecuada**: el software debe contar con una interfaz de usuario adecuada y su documentación.

Tipos de Software (Pressman)

* Por función
  + Software de sistemas: SO.
  + Software de aplicación: ofimática, correo electrónico, etc.
  + Software científico y de ingeniería: autocad, geogebra, etc.
  + Software empotrado: software que está incluido en productos que no son normalmente de software: heladeras, autos, lavarropas, etc.
  + Software de línea de productos: software para la generación de líneas de productos.
  + Aplicaciones Web: desarrollo web.
  + Software de Inteligencia Artificial: robótica, sistemas expertos, redes neuronales, etc.
  + Juegos.
* Por origen
  + Software nuevo: aquel que se desarrolla para cumplir con una función.
  + Software heredado
  + Software de integración: software que comunica diferentes softwares.
* Por arquitectura
  + Monolíticos
  + Cliente-Servidor
  + Aplicaciones basadas en web
  + Etc.

Costos de Hardware y Software



Necesidad de un proceso

Un proceso de desarrollo de software:

* Organiza y estructura las actividades.
* Contribuye a la calidad del software y a la velocidad con que se desarrolla.
* Define el enfoque que se adopta mientras el software está en desarrollo.
* Es parte de la Ingeniería de Software.

**Proceso de desarrollo de SW**

* Define un marco de trabajo para realizar SW de alta calidad
* Define un conjunto de actividades y resultados asociados que producen un “producto de SW”.
* Nos brinda estabilidad, organización y control.
* Define las interacciones:
  + Usuarios – Diseñadores
  + Usuarios – Herramientas
  + Diseñadores – Herramientas

¿Por qué seguir un proceso?

* Un proceso es un conjunto de procedimientos, organizado para construir productos que satisfacen una serie de objetivos y estándares.
* Los procesos son importantes porque imponen **consistencia** y **estructura** en un conjunto de actividades.
* Sabemos cómo hacer algo bien y queremos forzar que otros lo hagan de la misma forma.

Requisitos nuevos o modificados Proceso de desarrollo de SW Sistema nuevo o modificado

Elementos del proceso



Proceso de SW

* Distintos procesos de software organizan las actividades de diferentes formas, y las describen con diferente nivel de detalle.
* El tiempo de cada actividad varía, así como los resultados.
* Organizaciones diferentes usan procesos diferentes para producir el mismo producto.
* Para algunos tipos de aplicación, algunos procesos son más convenientes que otros.

Características del proceso de desarrollo

* **Entendible**: ¿se encuentra el proceso bien definido y es entendible?
* **Visible**: ¿El proceso es visible al exterior?
* **Soportable**: ¿Puede el proceso ser soportado por herramientas CASE? ¿Puede ser implementado según las condiciones existentes?
* **Aceptable**: ¿El proceso es aceptado por los involucrados en él?
* **Confiable**: ¿Los errores del proceso son descubiertos antes de que se conviertan en errores del producto?
* **Mantenible**: ¿Puede el proceso evolucionar para cumplir con los objetivos organizacionales?
* **Rapidez**: ¿Cuánto más rápido puede producirse el producto (SFW)?

**Modelos del proceso – Ciclo de Vida**

* Se debe escoger una estrategia de desarrollo, llamada modelo de proceso o ciclo de vida
* Se selecciona de acuerdo a:
  + Naturaleza del proyecto y aplicación
  + Controles y entregas requeridas
  + Características del equipo u organización de desarrollo, entre otros.

Ninguno es mejor que otro, cada proyecto tiene el más apropiado.

Ciclo de Vida

Sucesión de etapas por las que atraviesa un producto de software a lo largo de su existencia.

“Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software.” – IEEE 1074

“Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.” – ISO 12207-1

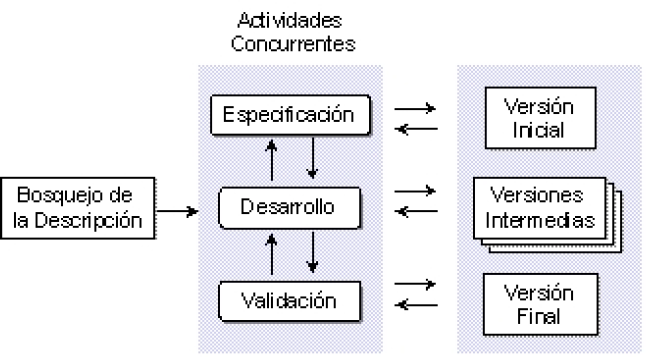
Modelos prescriptivos de procesos

Las principales diferencias entre distintos modelos de ciclo de vida están divididas en tres grandes visiones:

* El alcance del ciclo de vida (hasta dónde se desea llegar con el proyecto): sólo saber si es viable el desarrollo de un producto, el desarrollo completo o el desarrollo completo más actualizaciones y mantenimiento.
* La cualidad y cantidad de las etapas en que será divido el ciclo de vida: según el que sea adoptado y el proyecto para el cual sea adoptado.
* La estructura y la sucesión de las etapas, si hay realimentación entre ellas y si hay libertad para repetirlas (iteración).

Modelos incrementales

* Incrementos: proporcionan un subconjunto de funcionalidad: se entrega “algo de valor” al cliente con cierta frecuencia.
* El cliente se involucra más, se usan los incrementos como prototipos y generan experiencias de validación.
* Difícil de aplicar a sistemas transaccionales que tienden a ser integrados y a operar como un todo.
* Los errores en los requisitos se detectan tarde.



¿Cuándo usaríamos un modelo incremental?

* Cuando no se dispone del personal para una instalación completa.
* Cuando se pueden definir los incrementos acotados en tamaño, con funcionalidades bien definidas.
* Cuando no se está seguro de cumplir con plazos de tiempo o se tiene una fecha imposible de cambiar.
* **No es recomendable** cuando se tienen atributos de calidad críticos.

Modelos Evolutivos

* La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N versiones hasta que se desarrolle el sistema adecuado.
* Se adaptan más fácilmente a los cambios introducidos a lo largo del desarrollo.
* Iterativos
* En cada iteración se obtienen versiones más completas del software.

Construcción de Prototipos

* No están claros los requerimientos al inicio.
* Sistemas nuevos, poco conocimiento del dominio
* Útil cuando hay inseguridades del lado cliente y/o desarrollador.
* Reduce el riesgo.
* Especificación: desarrollo creciente.
* No modifica el flujo del ciclo de vida.
* Una vez identificados los requerimientos, se construye el producto.

**Métodos Ágiles**

* Surgen en 2000 como alternativa a los procesos de desarrollo de software tradicionales:
  + Caracterizados por ser rígidos y
  + Dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.
* Punto de partida: Manifiesto Ágil -> documento que resumen la “filosofía ágil”.

Principios fundamentales

* **Valorar al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.** La gente es el principal factor de éxito de un proyecto de software.
* **Desarrollar software que funciona más que conseguir una buena documentación**. No producir documentos a menos que sea necesarios de forma inmediata para tomar una decisión importante.
* **La colaboración con el cliente más que la negociación de un contrato.** Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo.
* **Responder a los cambios más que seguir estrictamente un plan.** La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo.

Tipos de Metodologías Ágiles

* Programación Extrema (XP – Extreme Programming)
  + Metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software.
  + Promueve el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores.
  + Realimentación continua entre el cliente y el equipo de desarrollo.
  + Comunicación fluida entre todos los participantes.
  + Simplicidad en las soluciones implementadas.
  + Flexibilidad para enfrentar los cambios.
  + **Utilidad:** requerimientos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.
* SCRUM
  + Define un marco para la gestión de proyectos.
  + Está especialmente indicada para proyectos con un rápido cambio de requisitos.
  + Características:
    - El desarrollo de software se realiza mediante iteraciones, denominadas **sprints**, con una duración máxima de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente.
    - Promueve las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

**Ingeniería del Software**

… según la IEEE (1990) …

“… La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado, cuantificable para el desarrollo, operación y mantenimiento de software…”

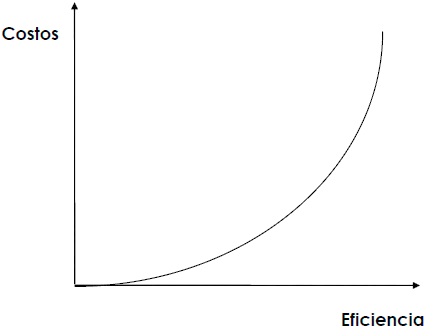
¿Por qué es difícil?

* El **software** es uno de los objetos de mayor complejidad hecho por humanos. Desarrollar software es resolver un juego de restricciones de naturaleza técnica, económica y humana.
* Una disciplina joven víctima de su propio éxito
  + La teoría sobre la que se debe apoyar la ingeniería no está terminada.
  + La tecnología y la capacidad de construir sistemas complejos crece rápidamente exigiendo más a una disciplina ingenieril que está madurando e introduciendo nuevas problemáticas.

Costos del Software

* Los costos del software a menudo dominan al costo del sistema. El costo del software en un PC es a menudo más caro que la PC.
* Cuesta más mantener el software que desarrollarlo. Para sistemas con una larga vida, este costo se multiplica.
* La Ingeniería de Software concierne a un desarrollo efectivo en cuanto a costos del software.

Costo de la eficiencia



Situación Actual

* Los cambios en hardware han sido enormes.
* Los cambios en software también:
  + Internet y aplicaciones relacionadas.
  + Enorme variedad de tecnologías para construir aplicaciones, que pueden ser desplegadas mucho más rápidamente que en el pasado.
* Sin embargo, más allá de las tecnologías, si miramos los procesos de ingeniería del software, muchas cosas permanecen igual.
* El modelo en cascada sigue siendo utilizado por más del 40% de las empresas a pesar de que sus serios problemas fueron identificados hace 20 años.
* Todavía muchos proyectos terminan tarde, exceden el presupuesto o no entregan el software que esperaban los clientes.
* En muchas áreas sigue sin existir un conjunto de estándares que se use ampliamente:
  + La disciplina no es todavía madura
  + Es necesario mayor esfuerzo en educación en IS.
* Tres problemas esenciales al comienzo del Siglo XXI (Sommerville):
  + El reto de lo heredado: mantener y actualizar software antiguo con funciones críticas, si parar el negocio, evitando costos excesivos.
  + El reto de la heterogeneidad: desarrollar sistemas flexibles, multiplataforma, etc.
  + El reto de la entrega: reducir tiempo de entrega sin reducir la calidad del sistema.

Demandas Comerciales

* Sistemas que nunca deben fallar y que siempre deben alcanzar sus plazos de entrega. (Ejemplo: sistemas de control de aviones).
* Sistemas que deben ser seguros, confiables, livianos y extensibles. (Ejemplo: Sistemas de Tarjetas de Crédito o compras on-line).
* Sistemas de redes abiertas de performance crítica, y que son muy costosos de apagar.

IS ¿Saber todo?

Resumen: existen muchos temas de la IS que pueden ser estudiados. Muchos. De verdad.

IS: Problemas Principales

* La planificación y la estimación de costos son frecuentemente imprecisas.
* La productividad no se corresponde con la demanda.
* La calidad del software no llega a ser a veces ni aceptable.

Conclusiones

* Trabajamos con algo muy complejo.
* Que trata con una problemática muy amplia.
* Que tiene dificultades esenciales.
* Y que además está evolucionando…
* Hay muchos temas por cubrir.

Nos vamos a concentrar en aquellos que más afectan la construcción del software.

…No existe un único enfoque para mejorar el mal del software. Sin embargo, embargo, mediante la combinación de:

* Métodos completos para todas las fases del desarrollo del software,
* Mejores herramientas para automatizar estos métodos,
* Bloques de construcción más potentes para la implementación del software,
* Mejores técnicas para garantizar la calidad del software,
* Y una filosofía predominante para coordinación, control y gestión,

podemos conseguir una disciplina para el desarrollo del software: “Ingeniería del Software”.

Problemas comunes del desarrollo de SW

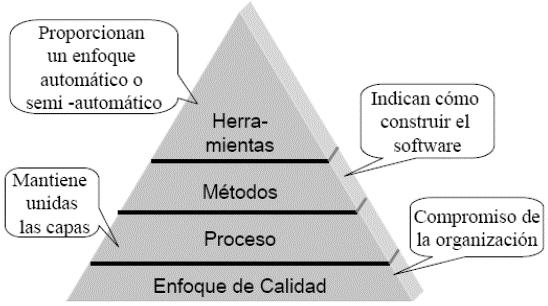
* Incapacidad para estimar tiempo, costos y esfuerzo para el desarrollo de un producto software.
* Falta de calidad del producto software.
* Avance del HW y necesidad de aplicaciones más complejas.
* Cambio en la relación entre el costo hardware/software.

Visión General de IS

Independientemente del producto al que se va a aplicar la ingeniería, se deben resolver las siguientes preguntas:

* ¿Cuál es el problema a resolver?
* ¿Cuáles son las características más relevantes del producto a cumplir?
* ¿Cómo se construirá el producto?
* ¿Qué enfoque se va a utilizar para evitar los riesgos?
* ¿Cómo se soportará el mantenimiento del producto?

IS: Proceso, métodos y herramientas



¿Por qué es importante definir un proceso?

Necesidad de un proceso

* Un proceso define quién está haciendo qué, cuándo y cómo lograr cierta meta.
* El Proceso de desarrollo de Software:
  + Organiza y estructura las actividades.
  + Contribuye a la calidad del software y a la velocidad con que se desarrolla.
  + Define el enfoque que se adopta mientras el software está en desarrollo.
  + Es parte de la Ingeniería de Software.

Siguiendo un proceso

* Un proceso es un conjunto de procedimientos (receta), organizado para construir productos que satisfacen una serie de objetivos y estándares.
* Los procesos son importantes porque imponen **consistencia** y **estructura** en un conjunto de actividades.
* Sabemos cómo hacer algo bien y queremos forzar que otros lo hagan de la misma forma.

Proceso de Desarrollo de Software

Requisitos, necesidades ¿Cómo se hace un proyecto software? Sistema software

Proceso de Software

* Conjunto estructurado de actividades requeridas (y resultados asociados) para desarrollar o producir un sistema software (producto).
  + Planificación
  + Especificación
  + Diseño o Desarrollo
  + Validación
  + Evolución
  + Gestión
* Las actividades varían dependiendo de la organización y del tipo de SW a ser desarrollado.

El proceso debe estar explícitamente definido para ser bien administrado.

**Estándares de la IS**

* **Estándar**: conjunto de **criterios aprobados, documentados y disponibles** para determinar la adecuación de una acción o conjunto de acciones (estándar de proceso) o de un objeto (estándar de producto).
* **Guía**: conjunto de **criterios bien definidos y documentados** que encaminan una actividad o tarea. Es más flexible que un estándar.

Utilidad de los Estándares

Según Sommerville, loes estándares son útiles porque:

* Agrupan lo mejor y más apropiado de las buenas prácticas y usos
* Engloban los “conocimientos” que son patrimonio de una organización
* Proporcionan un marco para implementar procedimientos de aseguramiento de la calidad.
* Proporcionan continuidad entre el trabajo de distintas personas.

Estándares en IS

* Tipos:
  + Para Datos. Ejemplo: SQL2, SQL1999, ODMG 2.0.
  + De Codificación
  + Estructurales: políticas de división del software en módulos
  + De documentación
  + De proceso
  + Para otras actividades: seguridad, auditoría, planificación, seguimiento, control de calidad, etc.
* Ejemplos:
  + IEEE Standards Collection Software Engineering – 1998 Edition
  + IEEE Std. 610.12-1990, Glossary of Software Engineering Terminology
  + IEEE Std. 830-1993, Recommended Practice for Software Requirements Specifications.
  + IEEE Std. 990-1987, Recommended Practice for Ada as a Program Design Language.
  + IEEE Std. 1062-1987, Recommended Practice for Software Acquisition
  + IEEE Std. 1063-1987, Standard for Software User Documentation
  + IEEE Std. 1219-1992, Standard for Software Maintenance

Estándares relacionados con el proceso

* Necesidad establecer un enfoque disciplinado y sistemático
* Objetivos:
  + Conseguir un **marco común** para “hablar el mismo lenguaje” en el desarrollo y gestión de SW.
  + Definir los procesos de desarrollo, mantenimiento y gestión de software de forma genérica y abstracta.

Estándares

* SEI CMM-CMMi
* Familia ISO 9000 CALIDAD
* IEEE 1074 – IEEE Standard for Developing Software Life Cycle Processes
* ISO/IEC 12207 – Information technology – Software life cycle processes (posteriormente adoptado por IEEE/EIA)
  + *IEEE – Institute of Electrical and Electronics Engineering.*
  + *ISO – International Organization for Standarization.*
  + *IEC – International Electrotechnical Commission.*

**SEI’s CMM/CMMi (Capability Maturity Model)**

* Modelo de Evaluación de Procesos
* El enfoque SEI proporciona una **medida de la eficacia global de las prácticas de IS** de una compañía.
* Define 5 niveles que se obtienen como consecuencia de las respuestas a un cuestionario de evaluación.
* Los resultados son numéricos y proporcionan una indicación de la madurez del proceso en la organización.
* Niveles
  + **1 – Inicial**

El éxito depende de esfuerzos personales más que de procesos adecuadamente definidos y planificados.

* + **2 – Repetible/Gestionado**

Se establecen políticas y procedimientos para llevar a cabo un proyecto Se planifica y se revisa para evaluar objetivos. Se obtienen niveles de calidad parecidos a proyectos anteriores.

* + **3 – Definido**

Se adopta un proceso de software estándar y se adapta a cada proyecto. Se ajusta a las políticas de la organización.

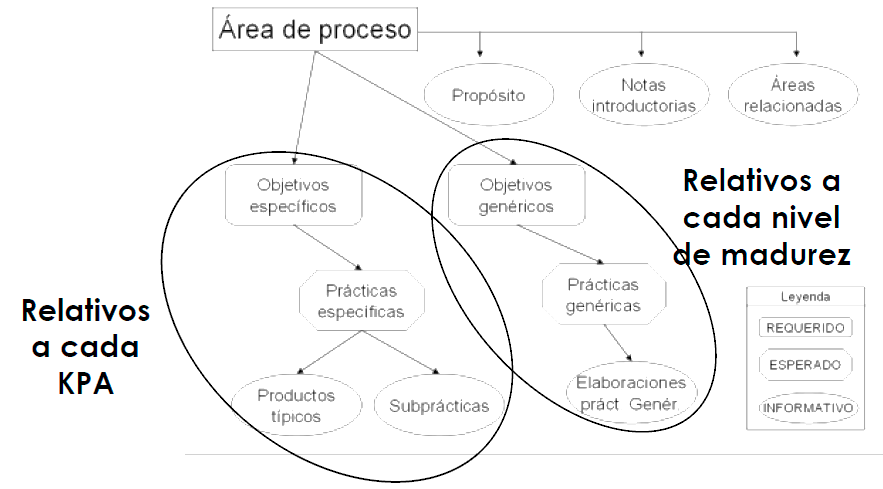
* + **4 – Gestionado/Cuantitativamente**

La calidad del producto y del proceso es evaluada, predecible y cuantificable. Se pueden usar dichas medidas (“métricas del software”) para detectar situaciones excepcionales y corregirlas.

* + **5 – Optimizado**

El proceso es continuamente mejorado usando las medidas obtenidas de procesos anteriores.

* En 2000 surge CMMi
  + Integra cuatro disciplinas: Software, Ingeniería de Sistemas, Desarrollo integrado de procesos y productos, Gestión de proveedores.
  + Propone cuatro categorías para agrupar las áreas clave del proceso
    - Administración de Procesos.
    - Administración de Proyectos.
    - Ingeniería.
    - Soporte.



* KPA: conjunto de prácticas relacionadas que son ejecutadas de forma conjunta para conseguir un conjunto de objetivos.
* CMMi – énfasis en:
  + Métricas
  + Trazabilidad requerimientos producto final
  + Cambia nombres de niveles 2 y 4 (Repetible Gestionado, Gestionado gestionado cuantitativamente)
* Dos representaciones:
  + Por niveles: nivel de madurez considerando todas las KPAs (define un nivel de madurez organizacional).
  + Continua: nivel de madurez en cada KPA independientemente de los restantes.

**ISO 9000**

* Familia de estándares para la gestión de la calidad de cualquier proceso de producción.
* La organización debe tener un proceso de calidad que supervise todas las fases de la producción y entrega del producto.
* Para cada proyecto se define un plan de calidad.
* Se auditan los proyectos para asegurar que se cumplen los controles de calidad.
* Fomenta la producción de documentación e informes.

Objetivos

* Estandarizar las actividades del personal por medio de la documentación
* Incrementar la satisfacción del cliente
* Medir y monitorear el desempeño de los procesos
* Incrementar la eficacia y/o eficiencia de la organización en el logro de sus objetivos.
* Mejorar continuamente en los procesos, productos, servicios, eficacia, etc.
* Surgen en el año 1994.

ISO 9001: 2008

* Especifica los requisitos para un **buen sistema de gestión de la calidad**. Puede utilizarse para su aplicación interna por las organizaciones, para certificación o con fines contractuales.
* Los estándares de calidad pueden ser propios de cada organización, las ISO enseñan a gestionarlos y aplicarlos. Si no existen, pueden tomarse los definidos en la ISO 9001.

ISO 9000

* ISO 9001: Quality Systems – Model for Quality Assurance in Design, Development, Production, Installation and Servicing. Describe el sistema de calidad utilizado para desarrollar un producto con procesos que cubren su ciclo de vida.
* ISO 9000-3: Guidelines for Application of ISO 9001 to the Development, Supply and Maintainance of Software. Documento específico que interpreta el estándar ISO 9001 para el desarrollador de software.
* ISO 9004-2: Quality Management and Quality System Elements (Part 2). Este documento proporciona las directrices para las actividades de soporte al desarrollo de software bajo un sistema de calidad.

**IEEE 1074 - Standard for Developing Software Life Cycle Processes**

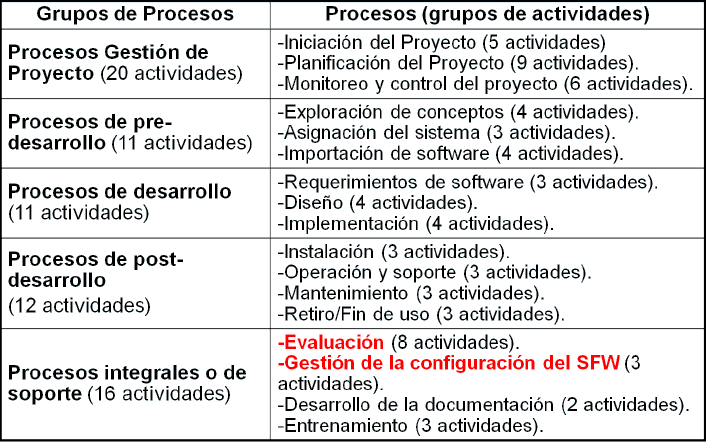
* Define
  + Las actividades que constituyen los **procesos** necesarios **para el desarrollo y el mantenimiento de software** (parte de un sistema o autónomo)
  + Los **procesos de gestión y soporte** a lo largo de toda la vida del software.
* Primera versión: 1074-1991. Última versión: 1074-2006.
* Ciclo de vida “una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento de software”.
* Es estándar requiere la definición de un ciclo de vida. Pero no implica ninguno determinado.
* Cada organización debe asociar las actividades definidas en el estándar a su propio ciclo de vida de software. Si no lo ha definido, debe hacerlo.
* El seguimiento del estándar no implica el uso de ningún método específico, ni la creación de determinados documentos. Prescribe los **procesos** del ciclo de vida, no los **productos** del mismo.

Implementación Estándar IEEE 1074

1. **Establecer los requerimientos para el proceso de ciclo de vida de un proyecto de software.**
   1. Los stakeholders (interesados) del proceso de ciclo de vida deben ser seleccionados.
   2. Sus necesidades y expectativas son transformados en un conjunto factible de requerimientos aceptables.
2. **Seleccionar el modelo de ciclo de vida para el proyecto**
   1. Identificar todos los modelos de ciclo de vida disponibles
   2. Identificar cuestiones propias del sistema de software y del ambiente que deban ser consideradas
   3. Identificar cualquier restricción que deba ser impuesta en la selección del modelo de ciclo de vida
   4. Evaluar todos los modelos de ciclo de vida aplicables, considerando experiencia previa y capacidades organizacionales.
   5. Seleccionar el modelo de ciclo de vida que mejor satisfaga los atributos del proyecto y las restricciones.
3. **Desarrollar el ciclo de vida del proyecto de software**
   1. Considerando los procesos y actividades definidos en el estándar
   2. Pueden incorporarse actividades nuevas, pero **DEBEN** existir las marcadas como “required”.
   3. Pasos:
      1. Ubicar las actividades en una secuencia ejecutable.
      2. Identificar y justificar actividades no utilizadas.
      3. Verificar el mapa de actividades (muchas utilizan como entrada salidas de otras actividades y esto debe ser respetado)
4. **Establecer el proceso de ciclo de vida del proyecto de software**

Integrar el modelo de ciclo de vida desarrollado en el punto anterior con los OPAs (organizational process assets) artefactos que definen el entorno organizacional en un proyecto de software (políticas, estándares, procedimientos, métricas, herramientas, metodologías, etc.). Debe ser seleccionados y adaptados para el proyecto en particular.

1. **Validar el proceso de ciclo de vida del proyecto de software**



* Procesos divididos en actividades (obligatorias y opcionales)
  + Información de entrada
  + Descripción
  + Información de salida
* Antes de empezar un proyecto, revisar las actividades para ver si son aplicables y establecer un orden.
* **Conformidad con el estándar**: implica la realización de **todas** las actividades obligatorias.

**Estándar ISO/IEC 12207: Information Technology / Software Life Cycle Processes**

* Establece un marco común para los procesos de ciclo de vida
* Emplea términos bien definidos
* Considera todo el ciclo de vida:
  + Desde la definición de requisitos hasta el fin de uso
  + Contiene procesos para adquirir y suministrar productos y servicios de software.
* Primera versión: 122077-1995. Última versión: 12207-2008
* Indica los procesos, actividades y tareas que se necesitan durante la adquisición de:
  + un sistema que contiene software,
  + un producto software autónomo,
  + un servicio software,
* y durante el suministro, desarrollo, operación y mantenimiento de productos software.

“Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso”

* Proceso: conjunto de actividades.
* Actividad: conjunto de tareas.
* Tarea: acción que transforma entradas en salidas.

Objetivos principales:

* Proporcionar una estructura de trabajo para que TODOS los involucrados en el desarrollo de software usen un **lenguaje común** se establece en forma de procesos bien definidos.
  + **Modularidad**: se pretende conseguir procesos con un mínimo acoplamiento y una máxima cohesión.
  + **Responsabilidad**: se busca establecer un responsable para cada proceso, facilitando la aplicación del estándar en proyectos en los que pueden existir distintas personas u organizaciones involucradas.

Características:

* Proporciona procesos para **definir, controlar y mejorar** los procesos de ciclo de vida software.
* El marco descrito por el estándar está diseñado para ser **adaptado** a cada organización y proyecto.
* El **proceso de adaptación** consiste en la eliminación de procesos, actividades y tareas no aplicables (también se pueden añadir).

**Procesos**:

* Procesos Principales:
  + Adquisición
  + Suministro
  + Desarrollo
  + Explotación
  + Mantenimiento
* Procesos de Soporte
  + Documentación
  + **Gestión de configuración**
  + **Aseguramiento de calidad**
  + **Verificación**
  + **Validación**
  + Revisión conjunta
  + Auditoría
  + Resolución de problemas
* Procesos de la Organización
  + Gestión
  + Infraestructura
  + Mejora
  + Formación
* Proceso de Adaptación

17 Procesos organizan 74 Actividades, que implementan 224 tareas, que interactúan con políticas y procedimientos, planes, registros, documentos y auditorías y revisiones.

* Procesos principales:
  + Útiles a las personas involucradas en la adquisición, suministro, desarrollo, explotación o mantenimiento del software. Compradores, proveedores, personal de desarrollo, operadores y personal de mantenimiento del software.
* Procesos de soporte:
  + Sirven de apoyo al resto.
  + Contribuyen al éxito y calidad del proyecto software.
  + Se aplican en cualquier momento del ciclo de vida.
* Procesos de la organización (procesos generales):
  + Objetivo: gestionar y mejorar la organización (gestión, formación del personal, mejora del proceso, etc.)
  + Se realizan fuera de proyectos específicos, a nivel organizativo.
* Proceso de adaptación:
  + Permite adaptar el estándar a cada proyecto y organización.
  + Factores que influencian la forma de adquirir, desarrollar, explotar o mantener un sistema:
    - Contexto organizacional,
    - Tamaño y complejidad del proyecto,
    - Requerimientos regulatorios,
    - Culturas corporativas, etc.

**Procesos principales:**

* **Proceso de adquisición**.
  + Actividades y tareas que el comprador, el cliente o el usuario realizan para adquirir un sistema o producto o servicio de software.
    - Preparación y publicación de una solicitud de ofertas.
    - Selección del proveedor del software.
    - Gestión de los procesos desde la adquisición hasta la aceptación del producto.
* **Proceso de suministro/provisión**
  + Actividades y tareas que realiza el proveedor
    - Se inicia al preparar una propuesta para atender una petición de un comprador, o por la firma de un contrato con el comprador para proporcionarle un producto software.
    - Identificación de procedimientos y recursos para gestionar bien el proyecto, Desarrollo de los planes del proyecto, Ejecución de los planes del proyecto hasta la entrega del producto software al comprador.
* **Proceso de Desarrollo**
  + Contiene las actividades y tareas realizadas por el desarrollador.
  + Integra las siguientes actividades:
    - Implementación del proceso.
    - Análisis de requisitos del sistema.
    - Diseño de la arquitectura del sistema.
    - Análisis de los requisitos del software.
    - Diseño detallado del software.
    - Codificación y prueba del software.
    - Instalación.
    - Apoyo a la aceptación.
  + Implementación del proceso.
    - Si no está especificado en el contrato, el desarrollador definirá un modelo de ciclo de vida.
    - Apropiado al ámbito, magnitud y complejidad del proyecto.
    - Las actividades y tareas del proceso de desarrollo serán seleccionadas y relacionadas con el modelo de ciclo de vida.
    - Si no están indicados en el contrato, el desarrollador deberá seleccionar, adaptar y utilizar aquellos estándares, métodos, herramientas y lenguajes de programación que son apropiados (y están documentados).
  + Análisis de requisitos del sistema.

Los requisitos del sistema incluyen:

* + - Hardware
    - Funciones y capacidades
    - Requisitos de seguridad
    - Requisitos de interacción hombre-máquina
    - Interfaces del sistema
    - Restricciones aplicables al diseño
    - Requisitos de aceptación
  + Diseño de la arquitectura del sistema.
    - Se identifica la arquitectura de alto nivel del sistema:
    - Se determinan los principales componentes hardware y software.
    - Se asignan responsabilidades y requisitos a dichos componentes.
  + Análisis de requisitos del software.
    - Se identifican y documentan los requisitos del software:
      * Especificaciones funcionales y de capacidad.
      * Interfaces externas.
      * Seguridad y protección (de información, daños personales, ...).
      * Datos que se van a manejar y requisitos de la BD.
      * Requisitos de instalación y de aceptación.
      * Requisitos de mantenimiento.
    - Varios estándares definidos para esta fase:
      * IEEE 830-1998. Recommended Practice for Software Requirements Specifications.
      * DI-IPSC-81433. Software Requirements Specification (estándar del DoD)
  + Diseño de la arquitectura del software.
    - Componentes principales del software.
    - Diseño de alto nivel de la BD.
    - Versiones preliminares de los manuales de usuario.
    - Requisitos de las pruebas y planificación de la integración del software.
  + Diseño detallado de software.
    - Diseño detallado de cada componente de software.
    - Diseño detallado de las interfaces.
    - Diseño detallado de la BD
    - Actualizar manuales de usuario.
    - Definir y documentar los requerimientos de prueba.
    - Actualizar requisitos de prueba para la integración del software.
    - Evaluar todo lo anterior.
    - Reuniones de revisión.
  + Codificación y prueba del software.
    - Se desarrollan los componentes software y las bases de datos.
    - Se prueban los componentes (prueba de unidad).
    - Se actualizan los manuales de usuario.
  + Actividades Finales.
    - Integración del software
    - Prueba del software
    - Integración del sistema
    - Prueba del sistema
    - Instalación del software
    - Soporte del proceso de aceptación del software.
* **Proceso de Explotación u Operación**
  + Actividades y tareas sistema completo (HW + SW)
  + El sistema debe ser operado de acuerdo con la documentación de usuario, en su entorno previsto.
  + Entre otras actividades, el operador deberá:
  + Desarrollar un plan para llevar a cabo las actividades y tareas de este proceso.
  + Procedimientos para comprobar el producto software en su entorno de operación, enviando informes de problemas y peticiones de modificación al proceso de mantenimiento.
  + El operador debe proporcionar asistencia a los usuarios.
* **Proceso de Mantenimiento**.
  + El software o la documentación necesita ser modificado, debido a problemas, necesidades de mejora o adaptación
    - nuevos errores detectados, cambios en la legislación, cambios en el entorno, necesidad de mejoras, migración a un nuevo entorno operativo, se va a terminar con su uso …

“Modificar el software existente manteniendo su consistencia”

* + Comprende las siguientes actividades:
    - Implementación del proceso de mantenimiento.
    - Análisis del problema y de la modificación.
    - Implementación de la modificación.
    - Revisión y aceptación del mantenimiento.
    - Migración.
    - Fin de uso del software.

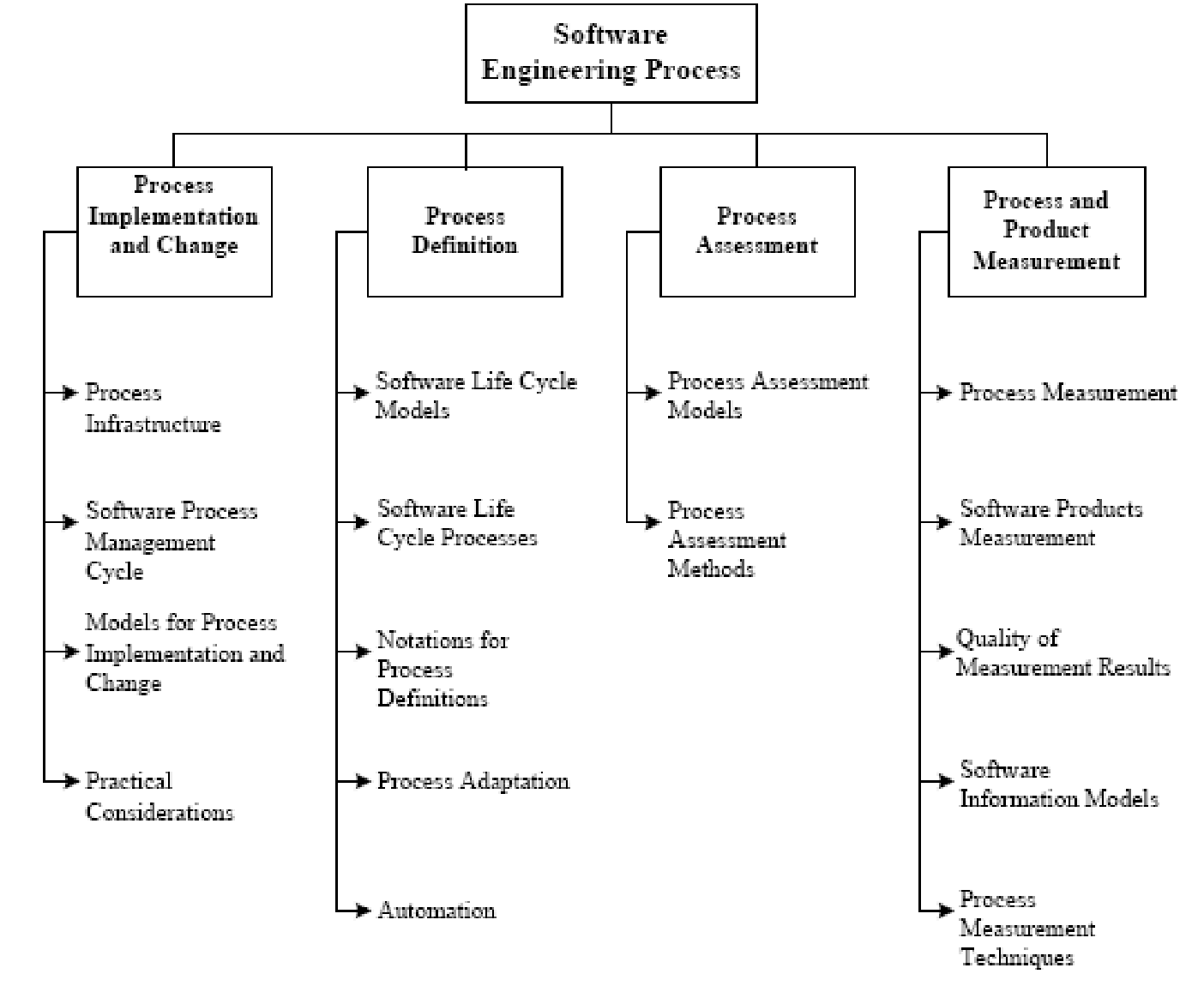
**Procesos de Soporte**:

* Sirven de apoyo al resto de procesos.
* Se aplican en cualquier momento del ciclo de vida:
  + Documentación.
  + Gestión de la configuración.
  + Aseguramiento de la calidad.
  + Verificación.
  + Validación.
  + Revisión conjunta.
  + Auditoría.
  + Resolución de problemas.
* **Proceso de documentación**.
  + Registrar la información producida por cualquier proceso o actividad del ciclo de vida.
    - Diseño y desarrollo,
    - Producción y
    - Mantenimiento
  + de los documentos necesarios para todas las personas involucradas en el proceso software.
* **Proceso de Configuración del software**.
  + Se encarga de gestionar:
    - las modificaciones de los elementos de configuración del software de un sistema…: “la modificación X al programa Y fue hecha por la persona Z”
    - y las versiones de los elementos …: “la última versión del programa X es la 1.4”
  + Se encarga de:
    - Registrar e informar sobre el estado de los elementos y las peticiones de modificación.
    - Asegurar la completitud, consistencia y corrección de los elementos.
    - Controlar el almacenamiento y la manipulación de los elementos.
* **Proceso de Aseguramiento de la calidad**.
  + Aporta confianza en que los procesos y los productos de software del ciclo de vida cumplen con los requisitos especificados y se ajustan a los planes establecidos.
  + Usa resultados de otros procesos de apoyo: verificación, validación, auditorías, etc.
* **Proceso de Verificación**. ¿Estamos construyendo correctamente el producto? Se verifica:
  + Si los requisitos de un sistema o del software están bien recogidos en cada modelo **verificación horizontal**
  + Si los productos software de cada fase del ciclo de vida cumplen los requisitos impuestos sobre ellos en las fases previas **verificación vertical**
* **Proceso de Validación**. ¿Estamos haciendo el producto correcto?
  + Indica si el sistema o software final cumple con las necesidades del usuario.
  + También se puede validar una especificación.
  + Puede ser realizado por una organización independiente (proceso de validación independiente).
* **Proceso de Revisión Conjunta**.
  + Evaluar el estado del software y sus productos en una actividad del ciclo de vida o fase del proyecto.
  + Se realiza durante todo el ciclo de vida.
* **Proceso de Auditoría**.
  + El objetivo de una auditoría es realizar una evaluación exhaustiva y producir un documento de recomendaciones para enmendar o mejorar los aspectos débiles que se detecten.
    - Si el software refleja la documentación de diseño.
    - Si los datos para las pruebas cumplen con la especificación.
    - Si el software ha sido probado.
    - Si la documentación de usuario cumple con las normas.
    - Si los costos y plazos se adhieren a los planes establecidos, etc.
* **Proceso de Resolución de Problemas**.
  + Analizar y eliminar los problemas (diferencias con el contrato o los requisitos) descubiertos durante el desarrollo, el mantenimiento, u otro proceso.
  + Toma como entrada resultados de otros procesos (validación, verificación, revisión, auditoría, etc.)
  + Objetivo: todos los problemas descubiertos se analizan y eliminan.

**Procesos Generales**:

* Ayudan a establecer, implementar y mejorar la gestión consiguiendo una organización más efectiva.
* Se llevan a cabo a nivel organizativo, fuera del ámbito de proyectos y contratos específicos.
  + Proceso de gestión.
  + Proceso de infraestructura.
  + Proceso de mejora.
  + Proceso de formación.
* Proceso de Gestión.
  + Se incluye en cualquier organización que tenga que gestionar sus procesos.
  + Implica planificación, seguimiento y control, revisión y evaluación.
* Proceso de Infraestructura.
  + Establece la infraestructura necesaria para el resto de procesos (para el desarrollo, la explotación o el mantenimiento): hardware, software, herramientas, normas, instalaciones.
* Proceso de Mejora.
  + Sirve para establecer, valorar, medir, controlar y mejorar los procesos del ciclo de vida del software.
  + Propone buenas prácticas para tener en cuenta: Personal Software Process (PSP), Quality Improvement Paradigm (QIP), etc.
  + Gestión de la calidad total.
* Proceso de Formación.
  + Sirve para mantener el personal formado, desarrollando un plan de formación.

**Proceso de IS en SWEBoK (IEEE, 2004)**

* El proceso de IS puede ser analizado en dos niveles.
* Primer nivel abarca las actividades técnicas y de gestión dentro de los procesos del ciclo de vida del software que son implementados durante la adquisición, desarrollo, mantenimiento y retiro.
* Segundo nivel es un meta-nivel que concierne la definición, implementación, evaluación, gestión, cambio y mejora de los procesos de ciclo de vida del software.
* Proceso de IS: importante no sólo para grandes organizaciones existen experiencias exitosas en organizaciones pequeñas, equipos y experiencias individuales.
* Involucra considerar e integrar otras áreas de conocimiento (por ejemplo, gestión de recursos humanos).

**Unidad 2: Ingeniería de Requerimientos**

Introducción a la Ingeniería de Requerimientos

**La Ingeniería de Requerimientos (IR) se relaciona con la búsqueda de la situación futura y el cambio asociado**.

Está relacionada con **encontrar (capturar) información y considerar** posibles **opciones**, y con la identificación de lo que debería ser diseñado en orden a satisfacer una necesidad futura percibida.

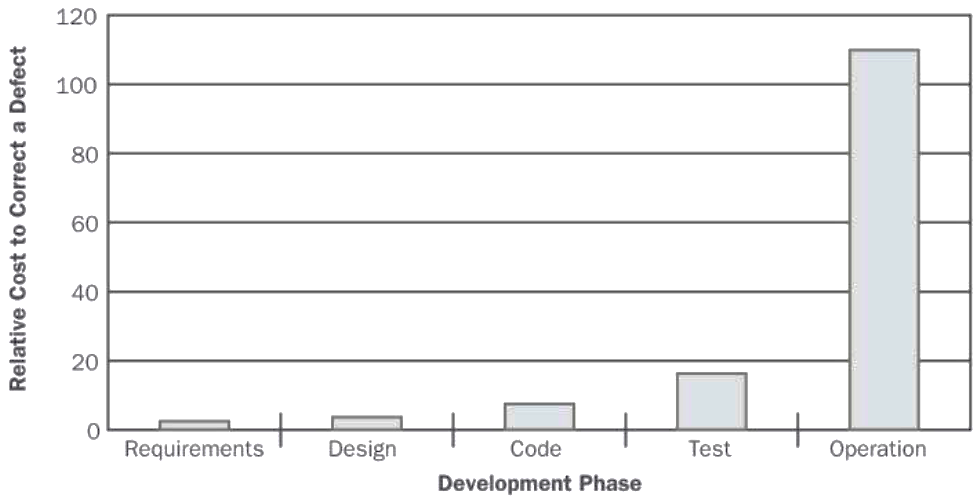
“La parte más difícil de construir un sistema de software es decidir precisamente qué construir. **Ninguna otra parte** del trabajo conceptual **es tan dificultosa** como establecer los requerimientos técnicos detallados, incluyendo todas las interfaces hacia las personas máquinas y otros sistemas de software. Ninguna otra parte del trabajo **paraliza el sistema si es mal hecha**. Ninguna otra parte **es tan difícil de rectificar después**.”

Las 3 ideas fundamentales a considerar:

* La parte más difícil del trabajo: al inicio.
* El impacto perjudicial del error.
* La dificultad de rectificar posteriormente.

Costo de reparar errores

El costo de reparar un defecto de software varía de acuerdo al momento en que se esté dentro del ciclo de vida del software.



Problemas del diseño de Software

Entre el 40% y el 60% de los defectos encontrados pertenecen a la etapa de requerimientos.

Esencialmente la diferencia es “entre lo que los desarrolladores piensan que tienen que construir y lo que los usuarios o clientes piensan que van a obtener”.

Problemática de la IR

* El desarrollo de software es una cadena de valor.
* Los defectos inyectados acarrean parte de ese valor.
* Existe la gran falacia de la velocidad. En realidad, cuanto menos tardo, más tardo.
* Progresar sobre requerimientos no consolidados es la principal razón de problemas en el desarrollo.

Gestión de Requerimientos

1995, Chaos Report

* Solo el 16% de todos los proyectos se completaron en calendario y presupuesto original.
* 8 problemas concentran 80% de las fallas (Pareto):
  + **Requerimientos incompletos (13,1%)**
  + **Falta de involucración del cliente (12,4%)**
  + Falta de recursos (10,6%)
  + Expectativas poco realistas (9,9%)
  + Falta de soporte ejecutivo (9,3%)
  + **Requerimientos cambiantes/volátiles (8,7%)**
  + **Falta de planeamiento/Planeamiento realista (7,5%)**
  + **Carencia/Ausencia de Necesidad (7,5%)**

Orientación de la IR

Trabajo presente del usuario y Opciones Tecnológicas Diseño del sistema Sistema Futuro

Ingeniería de Requerimientos: relacionada a encontrar la situación futura y el cambio asociado.

**Ingeniería de Requerimientos**

La ingeniería en requerimientos entra como una **subtarea** de la Ingeniería de Software; proporciona métodos, técnicas y herramientas que faciliten el trabajo de definición de lo que se quiere que haga el software.

Ingeniería de requerimientos: Rol fundamental

* **Definición del problema**
* **Estudio de factibilidad**
* **Análisis**
* Diseño del sistema
* Diseño detallado
* Implementación
* Mantenimiento

Debido a esta relación se deriva que entre los demandantes del software y la ingeniería de requerimientos se da una muy fuerte interacción.

¿Por qué es necesaria una etapa de requerimientos?

Desde el punto de vista del cliente una etapa de requerimientos es necesaria porque le ayuda a entender y expresar las nuevas necesidades y a identificar cómo puede satisfacerlas. Necesidad de Requerimientos.

Determinación de Necesidades

Compra o Desarrollo

Instalación

Introducción y entrenamiento

Uso limitado

Uso Total

Evaluación y decisión de etapa siguiente

Evaluación de limitaciones del producto

Requerimiento

* Simplemente… cualquier cosa que un cliente necesite.
* Desde el punto de vista del diseñador, cualquier cosa que necesite ser diseñada.

Requerimientos

* ’94 Jones: “Los requerimientos son las sentencias de necesidades de un usuario que lanzan el desarrollo de un programa o sistema”.
* ’93 Alan Davis: “una necesidad de un usuario o una facilidad necesaria, función o atributo de un sistema que puede ser censado desde una posición externa al sistema”
* En general, el énfasis está en: **lo que irá en el producto, sus características. No cómo el producto se diseñará o construirá.**
* ’97 Sommerville and Sawyer: “Requerimientos son una especificación de lo que debería ser implementado. Son una descripción de cómo el sistema debería comportarse o de una propiedad o atributo del sistema. Puede ser una restricción sobre el proceso de desarrollo del sistema”

**Lo que realmente necesitamos es asegurarnos que todos los stakeholders del proyecto llegan a un entendimiento compartido de los términos usados para describir los requerimientos**. Los errores en los requerimientos se detectan tarde.

La más notable de las definiciones la de la IEEE:

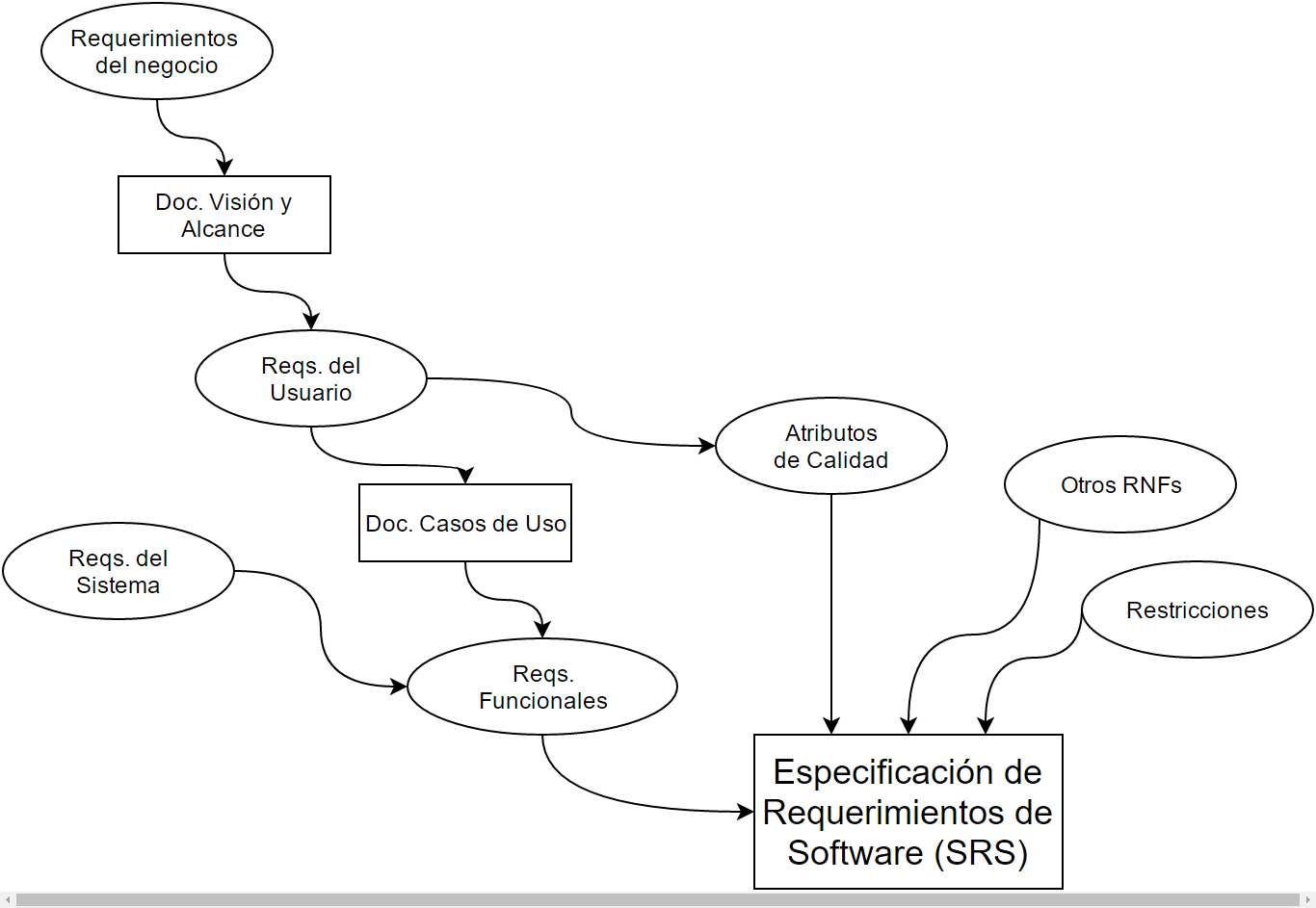
* Una condición o capacidad necesaria para un usuario para resolver un problema o alcanzar un objetivo
* Una condición o capacidad que debe ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación, u otro documento formalmente impuesto
* Una representación documentada de una condición o capacidad dada en 1 o 2.

Clasificación de los requerimientos

En cuanto al contenido:

* **Funcionales**: describen las entradas, salidas y funciones de transformación del sistema
* **No funcionales**: definen atributos como confiabilidad, performance, entre otros. Atributos de calidad, normalmente.
* **Inversos**: definen cómo el sistema de software nunca se debe comportar.

En cuanto al origen:

* **Del negocio**: objetivos de alto nivel de la organización. Documentados en la visión o alcance del proyecto.
* **Del usuario**: tareas que los usuarios deben hacer con el producto. Documentados en los casos de uso o escenarios.
* **Funcionales**: las funcionalidades del software a construir.
* **No funcionales**: que responden a estándares, regulaciones, contratos, interfaces, performance, restricciones y atributos de calidad.

Requerimientos y Calidad

* Los requerimientos están indisolublemente vinculados con la calidad del sistema.
* Calidad según IEEE:
  + El grado en que un sistema, componente o proceso satisface los requerimientos especificados.
  + El grado en el que un sistema, componente o proceso satisface necesidades y expectativas de un cliente o usuario.

Beneficios

Los beneficios de **buenos requerimientos** incluyen:

* Acuerdo entre los stakeholders (en relación al comportamiento esperado y al criterio de aceptación).
* Una base para estimaciones de recursos y tiempos.
* Mejoras en la usabilidad, mantenibilidad, etc.
* Reducción del esfuerzo: menos retrabajo y omisiones.

**Las ventajas crecen con el tamaño y complejidad del sistema.**

Rol de los Requerimientos

* Acuerdo desarrolladores-clientes-usuarios finales.
  + Aspecto contractual.
  + Multipartes (comunicación – conflictos – cambios de visiones).
* Base para el diseño de software.
  + Minimizar defectos tanto como sea posible.
  + Técnicamente factible.
* Soporte para verificación y validación.
* Soporte a la evolución del sistema.
  + Evolución del sistema: cambio del sistema viejo al sistema nuevo y cambio de requerimientos.

Ingeniería de Requerimientos

La IR se define como el proceso sistemático de desarrollo de los requerimientos a través de un proceso cooperativo e iterativo de análisis del problema, documentando las observaciones resultantes en una variedad de representaciones y chequeando la certeza del conocimiento ganado.

IR – Puntos a tener en cuenta

* Es una parte integral del proceso conectado a las otras partes por continuas retroalimentaciones.
* La naturaleza de los requerimientos es tal que los cambios ocurrirán necesariamente.
* Con las especificaciones de los requerimientos, el ciclo de desarrollo del software se hará con estos en mente.
* Hay varias fuentes de requerimientos que deben ser exploradas
* Habrá una necesidad continua de relacionar los requerimientos con las fuentes; también de relacionar el diseño y las decisiones de implementación a los requerimientos.
* Siempre habrá requerimientos conflictivos y situaciones decisivas que deberán acomodarse.

Alcance de la IR

Tiene que ver con:

* Los procesos que concluyen en la formulación de requerimientos.
* La generación de productos relacionados a esos procesos.
* La administración de los requerimientos en el desarrollo, operación y mantenimiento de software.

IR: El rol de las técnicas y las fallas

La mayoría de los problemas en el desarrollo de sistemas tienen su raíz no tanto en cuestiones técnicas (software) sino en cuestiones sociales, económicas, organizacionales y gerenciales (Bubenko, 1995).

* La falla o éxito de un proyecto siempre será definido con relación a un grupo particular con sus objetivos, intereses, roles y expectativas.
* Esos intereses y objetivos están definidos en el contexto global de una organización y su medioambiente político y social y no sólo con relación a la tecnología.

Tipos de fallas en el Desarrollo de Software

* **Fallas de correspondencia**: cuando los objetivos del proyecto no han sido alcanzados.
* **Fallas de procesos**: se relacionan a los procesos involucrados en el desarrollo de sistemas de información en los que el presupuesto, tiempo u otro recurso han sobrepasado el punto en donde se esperaba algún beneficio respecto del sistema propuesto, o donde los recursos gastados no resultaron en un sistema implementable.
* **Fallas de interacción**: es el argumento para el cual un bajo nivel de uso de un sistema puede ser interpretado como una falla.
* **Fallas de expectativas**: el sistema ha fallado en dar con las expectativas de al menos un grupo de interés sobre el sistema.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tipo de Falla \ Causa Posible | Falta de proceso sistemático | Comunicación pobre entre la gente | Falta de conocimiento apropiado o comprensión compartida | Documentación inapropiada, incompleta o precisa | Gestión pobre de gente o recursos |
| **Del proceso**  Beneficio negativo, recursos mal estimados | ✔ | ✔ |  |  | ✔ |
| **De interacción**  Bajo nivel de uso del sistema |  | ✔ | ✔ | ✔ |  |
| **De expectativas**  No alcanzó las expectativas de algún stakeholder |  | ✔ | ✔ | ✔ |  |

La gestión de requerimientos, como proceso, debe minimizar la ocurrencia de fallas.

IR como proceso

**El proceso de IR**: conjunto de actividades que resultan en el desarrollo de productos intermedios o finales.

**Objetivo:** estandarizar el proceso y los productos para:

* Permitir el seguimiento y control de cada actividad.
* Medir la efectividad del procedimiento y formas de mejorarlo a través de la evaluación de los productos parciales.
* Posibilitar el uso de herramientas automáticas aumentar la productividad y reducir los costos administrativos.

**Proceso de la IR**

IR: Subcomponentes

* Ingeniería de Requerimientos
  + Desarrollo de Requerimientos
    - Elicitación
    - Análisis
    - Especificación
    - Validación
  + Gestión de requerimientos

IR: Tareas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Desarrollo** | | | | **Gestión** |
| **Elicitación** | **Modelado** | **Análisis** | **Comunicación** |
| Identificación de Fuentes Inform. | Representación | Verificación | Documento de Requerimientos | Identificación de cambios |
| Recolección de hechos | Organización | Validación | Técnicas de comunicación | Análisis de cambios |
| Comunicación | Almacenamiento (registración) | Negociación |  | Realización de cambios |

Documento de Requerimientos

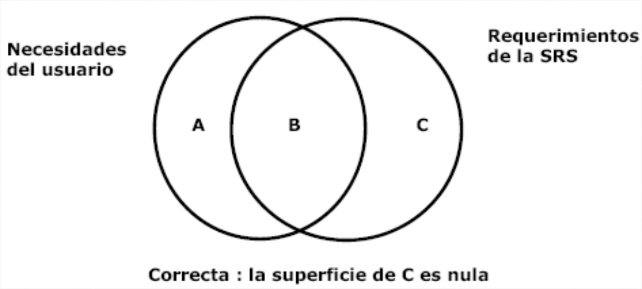
* Un Documento de Requerimientos (DR) es una especificación de lo que se requiere que haga un sistema de información y no de cómo hacerlo.
* Un DR puede ser evaluado por su efectividad como un medio de comunicación, por su contenido como una medida de chequeo (checklist), y por la calidad de su contenido.

IEEE: Guía de especificación de requerimientos

* Funcionalidad: ¿Qué se supone que hará el software?
* Interfaces externas: ¿Cómo interactúa el software con las personas, hardware y otras aplicaciones?
* Performance: ¿Cuál es la velocidad, disponibilidad, tiempo de respuesta, tiempo de recuperación, etc., de las funciones del software?
* Atributos: ¿Cuáles son las consideraciones sobre portabilidad, correctitud, mantenibilidad, seguridad, etc.?
* Restricciones: ¿Existe algún estándar vigente, lenguaje de implementación, políticas de integridad de BD, limitaciones de recursos, entornos de operación, etc.?

Características de una buena definición de requerimientos

Una especificación de requerimientos de software (ERS/SRS) debe ser:

* Correcta
* No ambigua
* Completa
* Consistente
* Ranqueable (en importancia y/o estabilidad)
* Verificable
* Modificable
* Trazable
* Usable
* **Correcta**
  + Si y solo si todo requerimiento formulado en la ERS debe ser satisfecho por el software.
  + Puede ser correcta pero no abarcar todas las necesidades planteadas.
  + Todo lo que está es lo que el cliente necesita. No tiene nada que no necesite.
* **No ambigua**: interpretación única.
  + Los requerimientos, comúnmente, están escritos en lenguaje natural, donde las oraciones pueden tener más de un significado.
  + Los lenguajes de requerimientos formales ayudan a reducir la ambigüedad, porque los procesadores de lenguaje detectan automáticamente cualquier error semántico, de léxico o sintáctico.
* **Completa**: la ERS es completa si incluye:
  + Todos los requerimientos significativos, donde se relacional la funcionalidad, performance, restricciones de diseño, atributos o interfaces externas y los estándares de la compañía.
  + Todas las respuestas del software a todas las entradas posibles
  + Todas las etiquetas y referencias a las figuras, tablas y diagramas de la ERS y las **definiciones de los términos.**

**Principio de redacción: el software hará sólo lo que dice la ERS.**

* **Consistente**: pueden ocurrir tres tipos de conflictos:
  + Diferentes términos usados para el mismo objeto.
  + Características de objetos conflictivas (contradictorias)
  + Fallas temporales o lógicas.
* **Ranqueable**: cada requerimiento tiene una identificación que indica su importancia y/o estabilidad. Esto ayuda a que:
  + Los clientes consideren mejor los requerimientos.
  + Los desarrolladores toman mejores consideraciones de diseño y asignan esfuerzos más adecuados.
* **Verificable**: una ERS es verificable si y solo si todo requerimiento es verificable.
  + Debe existir un proceso con el que un humano o máquina pueda chequear que el producto de software cumple con el requerimiento.
  + Los requerimientos ambiguos no son verificables.
* **Modificable**: la ERS deberá estar organizada en forma coherente y fácil de usar, con una tabla de contenidos, un índice y referencias cruzadas explícitas. Las sentencias de requerimientos no deben ser redundantes, en lo posible.
  + Cualquier cambio debería poder hacerse
    - Fácilmente
    - Manteniendo la completitud
    - Consistentemente
  + Requiere
    - Organización coherente y fácil de usar
    - No ser redundante
* **Trazable**: el origen de cada requerimiento debe ser claro, facilitando el rastreo de decisiones anteriores, y la búsqueda de todos los documentos que se referencian en la ERS.
* **Usable**: la ERS debe ser diseñada de modo que sea una referencia y si es necesario se modifique durante la vida del producto.
  + Es usado en las fases de operación y mantenimiento.
  + Tabla de contenido, índice y apéndice.

Beneficios de una buena ERS

Según la IEEE (1998), una buena ERS debe brindar diversos beneficios al proceso:

* Establecer la base de acuerdo entre los clientes y los proveedores sobre las características del producto de software.
* Reducir el esfuerzo de desarrollo.
* Provee una base para estimar costos y cronogramas.
* Provee una línea base para la validación y verificación.
* Facilita la transferencia (nuevos usuarios, máquinas, etc.).
* Sirve como base para la mejora posterior del producto.

Ingeniería de Requerimientos

* Los requerimientos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas.
* El proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones, se denomina **ingeniería de requerimientos**.

Requerimientos: Niveles de Abstracción

* **Los requerimientos de usuario:** son declaraciones, en lenguaje natural y en diagramas, de los servicios que se espera que el sistema proporcione y de las restricciones bajo las cuales debe funcionar.
* **Los requerimientos del sistema:** establecen con detalle las funciones, servicios y restricciones operativas del sistema. El documento de requerimientos del sistema (algunas veces denominado especificación funcional) debe ser preciso. Debe definir lo qué es lo que se va a implementar. Puede ser parte del contrato entre el comprador del sistema y los desarrolladores de software.

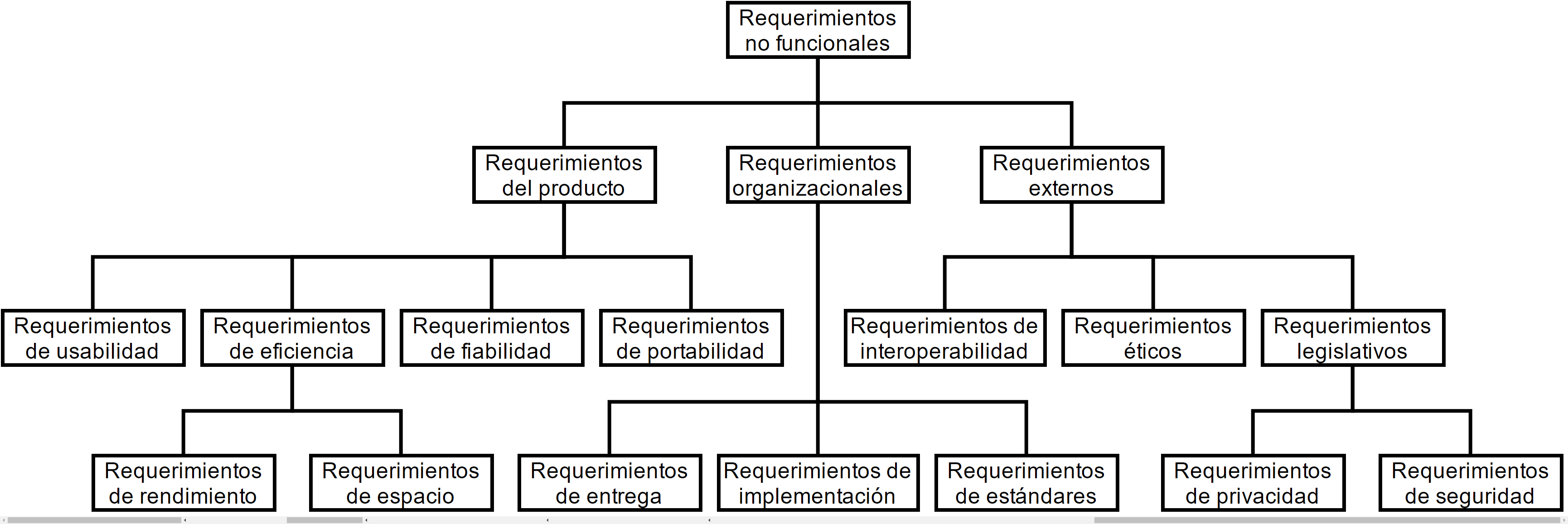
|  |  |  |
| --- | --- | --- |
| **Tipo Requerimiento** | **Descripción** | **Lectores** |
| Requerimientos de Usuario | El sistema deberá permitir que todos los suscriptos puedan subir, bajar y consultar las tesis de posgrado publicadas. | Administradores clientes  Usuarios finales  Ing. Clientes |
| Requerimiento de Sistema | 1. Al suscribirse, los usuarios deberán completar un formulario, con sus datos personales y antecedentes como investigadores. 2. El administrador autorizará diariamente los usuarios cuyos antecedentes sean suficientes. 3. Una vez autorizado, se enviará un mail, donde se informará de la aceptación de suscripción … | Usuarios finales  Desarrolladores del Sistema |

**Requerimientos Funcionales**

* Describen lo que el sistema debe hacer. Es importante que se describa el Qué y no el Cómo. Estos requerimientos al tiempo que avanza el proyecto de software, se convierten en algoritmos, la lógica y gran parte del código del sistema.
* Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer.
* En principio, la especificación de requerimientos funcionales de un sistema debe estar **completa** y ser **consistente.**
  + La completitud significa que todos los servicios solicitados por el usuario deben estar definidos.
  + La consistencia significa que los requerimientos no deben tener definiciones contradictorias.
* En la práctica, para sistemas grandes y complejos, es prácticamente imposible alcanzar los requerimientos de consistencia y completitud.

**Requerimientos No Funcionales**

* Son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento.
* Pueden especificar el rendimiento del sistema, la protección, la disponibilidad, y otras propiedades emergentes. Esto significa que a menudo son más críticos que los requerimientos funcionales particulares.
* Algunos pueden restringir el proceso que se debe utilizar para desarrollar el sistema.



Requerimientos del Producto

* Especifican o restringen el comportamiento del software.
* Indican requerimientos de eficiencia (rendimiento o espacio), de seguridad, de usabilidad y de contabilidad.

Requerimientos de la organización

* Son requerimientos derivados de políticas y procedimiento en la organización tanto del cliente como del desarrollador.
* Incluyen requerimientos operacionales que definen como se usará el sistema, requerimientos del proceso de desarrollo que especifican cuestiones específicas del desarrollo del software y requerimientos ambientales que definen el entorno de operación del sistema.

Requerimientos Externos

* Cubre todos los requerimientos derivados de factores externos al sistema y a su proceso de desarrollo.
* En ellos se incluyen requerimientos regulatorios que establecen lo que debe hacer el sistema para ser aprobado en su uso por un ente regulador. Requerimientos legislativos que deben seguirse para que un sistema funcione acorde con la ley y requerimientos éticos que garanticen que el sistema será aceptable para sus usuarios y público en general.

Requerimientos No Funcionales

* A menudo son difíciles de verificar.
* Para esto se definen métricas para poder “medir” el cumplimiento de los requerimientos funcionales:
  + Rapidez: cantidad de transacciones por segundo, tiempo de respuesta al usuario o a eventos, tiempo de actualización de pantalla.
  + Tamaño: Kbits, RAM, etc.
  + Facilidad de uso: Tiempo de formación, cantidad de cuadros de ayuda.
  + Fiabilidad: tiempo medio entre fallas, probabilidad de no disponibilidad, tasa de ocurrencia de fallos.
  + Robustez: Tiempo de reinicio después de fallos, probabilidad de corrupción de los datos después de fallos.
* Los requerimientos no funcionales tales como la fiabilidad, la seguridad y confidencialidad son particularmente importantes para los sistemas críticos.

Reflexionemos

¿Por qué muchas veces los requerimientos no funcionales entran en conflicto con los requerimientos funcionales? Ejemplos:

* Restricciones de memoria.
* Restricciones del lenguaje.
* Restricciones en la metodología.

**Requerimientos y sus características**

Son las propiedades principales. Un conjunto de requerimientos en estado de madurez, deben presentar una serie de características tanto individualmente como en grupo:

* Necesario
* Conciso
* Completo
* Consistente
* No ambiguo
* Verificable

Características de los requerimientos

* **Necesario**
  + Un requerimiento es necesario si su omisión provoca una deficiencia en el sistema a construir, y además su capacidad, características físicas o factor de calidad no pueden ser reemplazados por otras capacidades del producto o del proceso.
* **Conciso**
  + Un requerimiento es conciso si es fácil de leer y entender. Su redacción debe ser simple y clara para aquellos que vayan a consultarlo en un futuro.
* **Completo**
  + Un requerimiento está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
* **Consistente**
  + Un requerimiento es consistente si no es contradictorio con otro requerimiento.
* **No** **ambiguo**
  + Un requerimiento no es ambiguo cuando tiene una sola interpretación. El lenguaje usado en su definición, no debe causar confusiones al lector.
* **Verificable**:
  + Un requerimiento es verificable cuando puede ser cuantificado de manera que permita hacer uso de los siguientes métodos de verificación: inspección, análisis, demostración o pruebas.

Procesos de IR

La IR tiene que ver con:

* Los procesos que concluyen en la formulación de requerimientos.
* La generación de productos relacionados a esos procesos.
* La administración de los requerimientos en el desarrollo, operación y mantenimiento de software.

Los procesos de IR incluyen 4 actividades de alto nivel:

* Valorar si el sistema es útil para la empresa (estudio de factibilidad).
* Descubrir requerimientos (adquisición y análisis).
* Convertir estos requerimientos en alguna forma estándar (especificación).
* Comprobar que los requerimientos definan realmente el sistema que requiere el cliente (Validación).

Adquisición y Análisis de Requerimientos

* Luego de un estudio de factibilidad inicial, se realiza la adquisición y análisis de requerimientos.
* Se trabaja en conjunto con el cliente y usuarios finales.
* Las actividades de este proceso son:
  + Descubrimiento de los requerimientos.
  + Clasificación y organización de los requerimientos.
  + Priorización y negociación de requerimientos.
  + Especificación de los requerimientos.

**Adquisición de Requerimientos**

* Es el proceso de recopilar información sobre el sistema requerido y otros sistemas existentes, como así también a partir de esta información, separar los requerimientos de usuario de los de sistema.
* Las fuentes de información durante la fase de descubrimiento de requerimientos incluyen: documentación, stakeholders, y especificaciones de sistemas similares.
* En el estudio de todo sistema, es primordial la recolección de datos para poder realizar su estudio.
* De esta recolección, dependerán en cierta medida, las siguientes etapas del sistema.
* No existe un único procedimiento para esta etapa, ya que cada sistema es único, pero las técnicas más utilizadas son: material escrito, entrevistas, cuestionarios, observación y muestreo.

Entrevistas

* Las entrevistas con los interesados del sistema son una parte de los procesos de ingeniería de requerimientos.
* Es una conversación entre dos personas, es un proceso continuo utilizado por el analista para obtener un modelo del sistema, y para obtener conocimientos sobre los problemas del sistema.
* De acuerdo al momento en el que se realicen las entrevistas en el proceso de análisis se clasifican en:
  + Iniciales o preliminares: se realizan a modo de presentación sobre el trabajo a realizar.
  + Recolección de datos: son las centrales y las que involucran el mayor tiempo de trabajo.
  + Seguimiento: luego de la recolección de datos es necesario aclarar dudas, llenar espacios, etc.
* Entrevistas Estructuradas vs no Estructuradas
  + Una entrevista estructurada es aquella en la cual las preguntas están pre planeadas, y no se permiten desviaciones. No están indicadas para el flujo libre de ideas, se parecen a un cuestionario hablado. Ej.: entrevista de trabajo.
  + Una entrevista no estructurada puede variar considerablemente del plan original. Previamente se establecen los puntos principales.
* A quien entrevistar
  + En toda organización se debe entrevistar a diferentes personas, en distintos niveles de la organización, para obtener un panorama completo del entorno.
    - Nivel de decisión.
    - Nivel Ejecutivo.
    - Nivel operativo.
  + Siempre es aconsejable, comenzar desde el nivel más alto de la pirámide, para mejorar la accesibilidad a los niveles inferiores y obtener una visión general del sector.

Cuestionarios

* Consiste en una serie de preguntas escritas, que se responden por escrito también.
* Se utilizan cuando se busca la misma información en usuarios distintos.
* Normalmente se utilizan como complemento de otras técnicas.
* Debe ser: claro, breve, atractivo, probado.

Observación Directa

* Es un punto clave, ya que observar la dinámica del sistema, es la pieza que más elementos de juicio le va a dar al Analista para realizar un buen análisis.
* Es una técnica de observación que se usa para entender los procesos operacionales y ayudar a derivar requerimientos de apoyo para dichos procesos.
* Es información de primera mano, sin ningún tipo de “ruidos”.

Muestreo

* Consta en reunir solo un conjunto representativo de los datos.
* Es apropiada cuando existe un gran volumen de datos.

Material Escrito

* En muchos casos podemos encontrarnos con muy poco material escrito (organizaciones informales), o que este se encuentre desactualizado y disperso por toda la organización.
* Estos materiales pueden brindarnos una idea de cómo deberían operar los sistemas, pero no como operan en realidad.
* Se debe prestar especial atención a: Manuales de procedimientos, de usuarios, de operaciones, Formularios, y Descripción de trabajos.

Otras Técnicas: Escenarios

* Son un bosquejo de la interacción del sistema.
* Implica trabajar en forma interactiva con los usuarios y participantes del sistema.
* Un escenario debe incluir:
  + Una descripción de lo que esperan cuando se inicia el escenario.
  + Una descripción del flujo normal de los eventos.
  + Una descripción de lo que puede salir mal y cómo manejarlo.
  + Una descripción del estado del sistema al finalizar el escenario.

Otras Técnicas: Casos de Uso

* Se introdujo por primera vez en 1993.
* Es una característica fundamental del modelado de lenguaje unificado.
* Un caso de uso identifica actores e interacciones.
* Los casos de uso se documentan con un diagrama de casos de uso.
* El conjunto de casos de uso representa todas las interacciones posibles que se describirán en los requerimientos del sistema.
* No hay una distinción tajante entre escenarios y casos de uso.
* Los escenarios y casos de uso son técnicas para detectar requerimientos de usuarios directos del sistema.

Validación de Requerimientos

* Es el proceso de verificar que los requerimientos definan realmente el sistema que el cliente quiere.
* Durante el proceso de validación de requerimientos se realizan las siguientes comprobaciones:
  + Comprobación de validez.
  + Comprobación de consistencia.
  + Comprobación de totalidad.
  + Comprobación de realismo.
  + Verificabilidad.

Técnicas de Validación de Requerimientos

* Revisión de los requerimientos: se analizan utilizando un equipo de revisores que verifican errores e inconsistencias.
* Creación de prototipos: se muestra un modelo ejecutable a los usuarios finales y clientes, así pueden experimentar con este modelo para constatar si cubre las necesidades reales.
* Generación de casos de prueba: Los requerimientos deben ser comprobables. El desarrollo de pruebas a partir de los requerimientos del usuario antes de escribir cualquier código es una pieza integral de la programación extrema.

Administración de Requerimientos

* Los requerimientos para los sistemas siempre cambian.
* Los requerimientos del sistema también deben evolucionar para reflejar esa visión cambiante del sistema.
* Una vez que se instala un sistema y comienza su uso, surgirán nuevos requerimientos indefectiblemente.
* La administración de los requerimientos es el proceso de comprender y controlar los cambios en los requerimientos del sistema.
* Es necesario seguir la pista (trazabilidad) de cada requerimiento, y mantener los vínculos entre requerimientos dependientes.

Planeación de la Administración de Requerimientos

* Es una primera etapa esencial la planeación para poder administrar los requerimientos.
* Se debe hacer:
  + Identificación de los requerimientos.
  + Proceso de administración del cambio. (efectos y costo del cambio).
  + Políticas de seguimiento.
  + Herramientas de apoyo.
* La administración de requerimientos necesita herramientas automatizadas que deben seleccionarse en la etapa de planificación.

Herramientas Automatizadas

Se necesitan herramientas de apoyo para:

* Almacenamiento de los requerimientos.
* Administración del cambio.
* Administración del seguimiento.

Administración del Cambio en los Requerimientos

* Debe aplicarse a todos los cambios propuestos a los requerimientos del sistema.
* La administración del cambio es esencial, ya que es fundamental conocer los costos de un cambio, para saber si debe llevarse a cabo o no.
* 3 Etapas fundamentales para la administración del cambio:
  + Análisis del problema y especificación del cambio.
  + Análisis del cambio y estimación del costo.
  + Implementación del cambio.

**Unidad 3: Medidas e indicadores de software**

Introducción

* La medición es fundamental en cualquier ámbito de la ingeniería, y la IS no es una excepción.
* La medición nos permite tener una visión más objetiva, y profunda proporcionando información para una evaluación fundamentada.
* Cuando se puede medir y pasar a números lo que se está haciendo, se conoce algo más. Cuando no se puede medir, el conocimiento es precario y deficiente.

Métricas y mediciones

* Las métricas del software se refieren a un amplio elenco de mediciones para el software de computadora.
* La medición se puede aplicar al proceso del software con el intento de mejorarlo sobre una base continua.
* Se puede utilizar en el proyecto del software para ayudar en la estimación, el control de calidad, la evaluación de productividad y el control de proyectos.
* Finalmente, el ingeniero de software puede utilizar la medición para ayudar a evaluar la calidad de los resultados de trabajos técnicos y para ayudar en la toma de decisiones táctica a medida que el proyecto evoluciona.

Razones para medir

Razones para medir los procesos de software, los productos y los recursos:

* Caracterizar
* Evaluar
* Predecir
* Mejorar

Medir para...

* Caracterizamos para comprender mejor los procesos, los productos, los recursos y los entornos y para establecer las líneas base para las comparaciones con evaluaciones futuras.
* Evaluamos para determinar el estado con respecto al diseño. Las medidas utilizadas son los sensores que nos permiten conocer cuándo nuestros proyectos y nuestros procesos están perdiendo la pista, de modo que podamos ponerlos bajo control. También evaluamos para valorar la consecución de los objetivos de calidad y para evaluar el impacto de la tecnología y las mejoras del proceso en los productos y procesos.
* Predecimos para poder planificar. Realizar mediciones para la predicción implica aumentar la compresión de las relaciones entre los procesos y los productos y la construcción de modelos de estas relaciones. Hacemos esto porque queremos establecer objetivos alcanzables para el coste, planificación, y calidad de manera que se puedan aplicar los recursos apropiados. Evaluamos para determinar el estado con respecto al diseño. Las medidas utilizadas son los sensores que nos permiten conocer cuándo nuestros proyectos y nuestros procesos están perdiendo la pista, de modo que podamos ponerlos bajo control. También evaluamos para valorar la consecución de los objetivos de calidad y para evaluar el impacto de la tecnología y las mejoras del proceso en los productos y procesos.
* Medimos para mejorar cuando recogemos la información cuantitativa que nos ayuda a identificar obstáculos, problemas de raíz, ineficiencias y otras oportunidades para mejorar la calidad del producto y el rendimiento del proceso.

Definiciones

* Dentro del contexto de la ingeniería del software, una **medida** proporciona una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto. La medición es el acto de determinar una medida. El IEEE Stadad Glossary of Software Engineering Terms [IEEE93] define métrica como “*una* ***medida*** *cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado*”.
* Una **métrica** del software relata de alguna forma las medidas individuales sobre algún aspecto (ej.: número medio de errores encontrados por revisión o el número medio de errores encontrados por persona y hora en revisiones)
* Un **indicador** es una métrica o una combinación de métricas que proporcionan una visión profunda del proceso del software, del proyecto de software o del producto en sí.

¿Qué medir?

* La medición es algo común en el mundo de la ingeniería. Se mide el consumo de energía, el peso, las dimensiones físicas, la temperatura, el voltaje, etc.
* La medición es mucho menos común en el mundo de la ingeniería de software. Existen problemas para ponerse de acuerdo sobre qué medir y las medidas de evaluación de problemas recopilados.

Indicadores del proyecto

Los indicadores de proyecto permiten al gesto de proyectos del software:

* Evaluar el estado del proyecto en curso
* Seguir la pista de los riesgos potenciales
* Detectar las áreas de problemas antes de que se conviertan en críticas
* Ajustar el flujo y las tareas del trabajo
* Evaluar la habilidad del equipo del proyecto en controlar la calidad de los productos de trabajo del software

Calidad del software

* El **Aseguramiento de Calidad (AC)** del software es el conjunto de actividades planificadas y sistemáticas necesarias para controlar que el **producto** (proceso, …) satisface los requisitos de **Calidad**.
* El AC debe estar presente en:
  + Métodos y herramientas de Análisis, Diseño, Programación y Testeo.
  + Control de la Documentación
  + Registro de Auditorías e Informes
  + Métodos para Medición y Evaluación

**Estándar ISO/IEC 12207**

El **Aseguramiento de Calidad** es un proceso de soporte del Estándar 12207.

Calidad del software

* **Medición de software:** necesidad de obtener datos cuantitativos que ayuden a mejorar la calidad. A partir de **Métricas**.
* **Creación de Modelos de Calidad:** útiles para discutir, planificar y obtener indicadores de calidad.
* **Aplicación de Estándares de calidad:** directrices (prescripciones) para el aseguramiento externo e interno de la calidad. ISO 9126, etc.

Medición de Software (Pressman)

La medición se utiliza a lo largo de un proyecto de software como apoyo en:

* Estimación
* Control de calidad
* Valoración de la productividad
* Control del proyecto

La medición es aplicada por los ingenieros de software para auxiliar la evaluación de la calidad de los productos de trabajo y para apoyar la toma de decisiones.

Ejemplos de métricas, indicadores, medidas

* Medida: un programa tiene 10.000 LDC (líneas de código).
* Métrica: la productividad del proyecto X fue de 250 (LDC/persona mes).
* Indicador: la productividad media de nuestra empresa es de 500(LDC/pm) y en el último proyecto ha sido de 250(LDC/pm).

Mediciones de Software

Mejora objetiva: midiendo atributos.

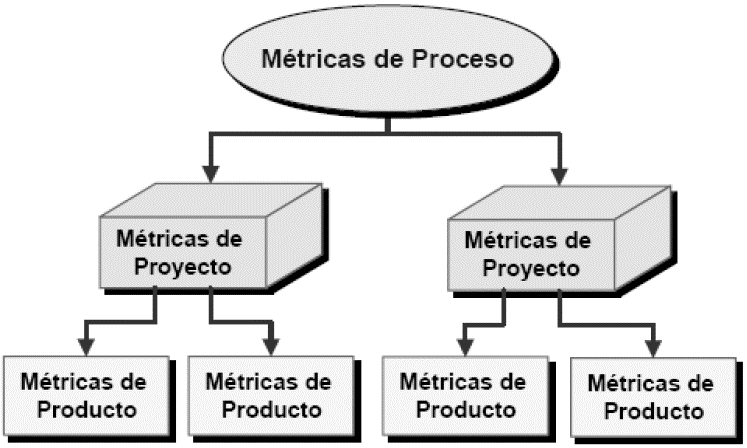
* Identificar atributos.
* Desarrollar métricas para esos atributos.
* Utilizar las métricas para obtener indicadores.
* Planificar e implementar el proceso de mejora.

Métricas: Beneficios

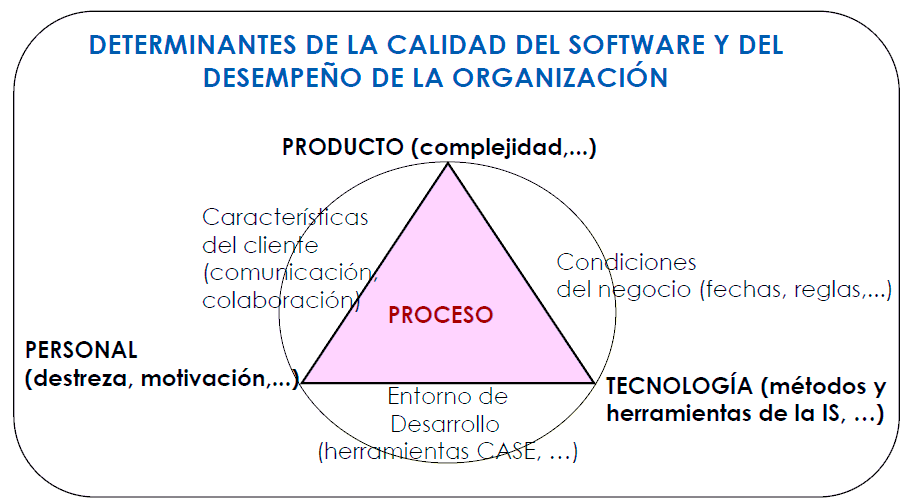
* Comunicación efectiva basada en hechos y datos y no en opiniones o cuestiones subjetivas.
* Identificación temprana de problemas y riesgos.
* Seguimiento y control de los objetivos.
* Soporte a las decisiones.

Se debe garantizar el buen uso de las métricas, de lo contrario, pueden ocasionar más problemas que beneficios. A veces, las métricas son usadas como medidas de persecución personal.

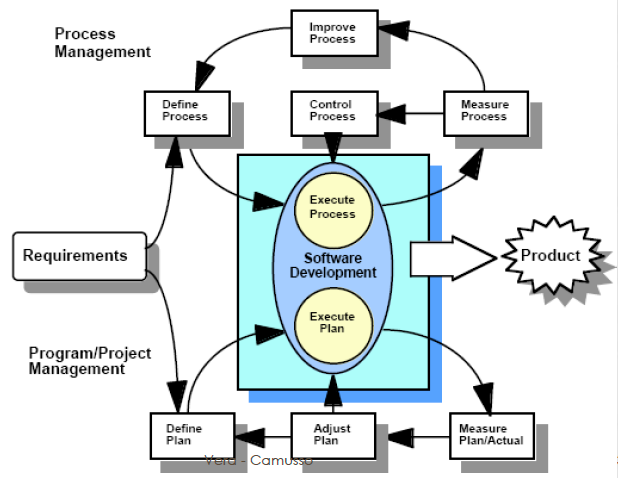
**Métricas: sus dimensiones**



* **Métricas del proceso:**
  + Para evaluar si los procesos de la organización se están llevando adelante según lo esperado.
  + Para asegurar que los procesos definidos están siendo considerados.
  + Para implementar mejoras de manera de lograr los objetivos de negocio (por ejemplo, minimizar riesgos y generar productos de calidad).



* + Mediciones del Proceso: única forma racional de mejorar procesos: medir sus atributos específicos, desarrollando y empleando métricas.
  + Métricas del Proceso:
    - Proporcionan Indicadores de Proceso
    - Se recopilan de varios proyectos
    - Se utilizan a largo plazo
    - Permiten tener visión de la eficiencia de un proceso existente (actividades, productos de trabajo, hitos, esfuerzos, etc.)
  + Las métricas del proceso pueden ser muy útiles, pero hay que saber interpretarlas.
  + Normas básicas de interpretación:
    - Utilizar el sentido común al interpretar los datos.
    - Proporcionar una realimentación regular a personas y equipos.
    - No utilizar métricas para evaluar a personas.
    - Establecer métricas claras y objetivos para alcanzarlas.
    - No utilizar métricas para amenazar a particulares o equipos.
    - Si una métrica identifica un área problemática, no se debería considerar como negativa.
    - Hay que interpretar todas las métricas en su conjunto, y no hacer prevalecer una en particular.
* **Métricas del Proyecto**
  + Planificar objetivamente
  + Conocer el estado y el progreso de los productos.
  + Se focalizan en cuestiones clave que comprometen el proyecto (tiempo, tamaño, esfuerzo, costo, calidad, estado).
  + Se utilizan para adaptar el flujo de trabajo del proyecto y las actividades técnicas.
  + Métricas del proyecto Indicadores del proyecto
    - Evaluar estado del proyecto actual
    - Analizar riesgos potenciales
    - Detectar áreas o cuestiones problemáticas
    - Ajustar flujos y tareas de trabajo
    - Evaluar habilidad del equipo en mantener bajo control la calidad de los productos.
  + Primera aplicación de las métricas del proyecto ¿Cuándo? Al principio, desde la planificación.
  + A medida que avanza el proyecto, las medidas del esfuerzo y el tiempo se comparan con las de la planificación.
  + El líder de proyecto utilizar estos datos para supervisar y controlar el avance.
  + Ejemplos:
    - Tiempo y esfuerzo medios de corrección de errores.
    - Errores detectados antes de la entrega de software.
    - Defectos detectados e informados por los usuarios finales, productos de trabajo entregados, etc.
  + Utilización fundamental de las métricas del proyecto:
    - *Optimizar la planificación*, guiando los ajustes necesarios que eviten retrasos y mitiguen problemas y riesgos potenciales.
    - *Evaluar la calidad de los productos en el momento actual*, modificando el enfoque técnico para mejorar la calidad, si es necesario.
* **Métricas del producto**
  + Asegurar la aceptación del cliente
  + Cuestiones de mayor peso: principalmente, las relativas a los atributos físicos y dinámicos del producto (arquitectura, productividad, confiabilidad, usabilidad, estabilidad, performance, etc.).
  + Mediciones del proceso y proyecto



Métricas del Proceso vs. Métricas del Proyecto

* Técnicamente no existe gran diferencia entre las métricas del proyecto y del proceso.
* Las métricas del proceso pueden concebirse como recopilaciones de métricas del proyecto.
* Las métricas del proceso son *estratégicas*: determinan el curso del *proceso de desarrollo de software*.
* Las métricas del proyecto son *tácticas*: determinan el curso del *proyecto actual*.

Muchas métricas se utilizan en ambos dominios.

* Las métricas de proyecto se consolidan con el fin de crear métricas del proceso que sean públicas para la organización como un todo.
* Ejemplo: un equipo A encontró 342 errores durante el proceso de software. El equipo B encontró 184 errores. Si todas las demás medidas se mantienen iguales, ¿qué equipo es más eficiente? No tenemos información sobre el tamaño ni la complejidad de los proyectos. Es necesario normalizar medidas para poder comparar.

**Métricas Orientadas al Tamaño**

* Líneas de código (LDC) valor de normalización.
* Métricas simples orientadas al tamaño:
  + Errores por KLDC, Defectos por KLDC, Costo por KLDC, Páginas de documentación por KLDC, ....
* Otras:
  + Errores por persona/mes, KLDC por persona/mes, costo por página de documentación.

**Métricas Orientadas a la Función**

* Valor de normalización: funcionalidad entregada por la aplicación.
* Métrica más utilizada: Punto de Función (PF).
* PF se calcula en base a características del dominio y de la complejidad esperada del software.
* PF es independiente del lenguaje de aplicación.
* PF puede conocerse en etapas tempranas del proyecto.

**Métricas Orientadas al tamaño y a la función**

* Las métricas basadas en PF y LDC son indicadoras del esfuerzo y costo del desarrollo de software.
* Existe relación entre ambas medidas
* La estimación de la funcionalidad de la aplicación está relacionada con las LDC y con el esfuerzo de desarrollo necesario.

**Métricas Orientadas a Objetos**

* Nº de guiones de escenario: directamente relacionado con el tamaño de la aplicación y con el número de casos de prueba necesarios.
* Nº de clases clave: “componentes independientes”, centrales en el dominio del problema. Indican la cantidad de esfuerzo necesario para desarrollar el software.
* Nº de clases de apoyo: no están relacionadas con el dominio. Son necesarias en la implementación.
* Nº de subsistemas: agregados de clase que apoyan una función visible para el usuario final.

**Métricas Orientadas a Casos de Uso**

* El CU se define en etapas tempranas del proceso útil en estimación
* Describen funciones y características visibles al usuario que son requerimientos básicos.
* Es independiente del lenguaje de programación.
* Directamente proporcional al tamaño de la aplicación en LDC y al número de casos de prueba necesarios para probar completamente la aplicación.
* Problema: grado de abstracción la importancia como medida de normalización es sospechosa.

**Métricas de proyectos de Ingeniería Web**

* Proyectos Web: énfasis en contenido funcionalidad.
* Métricas para evaluación de complejidad relativa y esfuerzo de desarrollo:
  + Nº de páginas web estáticas
  + Nº de páginas web dinámicas
  + Nº de vínculos internos de páginas
  + Nº de objetos de datos persistentes
  + Nº de sistemas externos
  + Nº de objetos de contenido estático y dinámico
  + Nº de funciones ejecutables
* Índice de personalización:

¿Por qué medir?

* Si no se mide, no existe forma real de determinar si se están haciendo bien las cosas (proyecto) o si se está mejorando (proceso).
* Sin embargo, la mayoría de los desarrolladores de software todavía no miden.
* **Medición: sirve para establecer una línea base a partir de la cual se evalúan mejoras.**

**Mediciones del Software**

* Las mediciones del mundo físico se pueden caracterizar de dos formas: medidas directas (la longitud de un tornillo) e indirectas (la calidad de un tornillo). Las métricas del software se pueden caracterizar de igual forma.
* Las medicas directas del proceso de ingeniería del software incluyen el costo y el esfuerzo aplicados.
* Pensemos ejemplos de métricas directas que vimos:
* LCD.
* Defectos informados, etc.
* ¿Y métricas indirectas? …
* Funcionalidad, calidad, eficiencia, facilidad de mantenimiento, etc.
* El costo y el esfuerzo para producir software, LCD, y otras medidas directas son relativamente fáciles de reunir. Las formas de evaluarlas se establecen más adelante.
* Sin embargo, la calidad, eficiencia, funcionalidad son más difíciles de evaluar y solo pueden ser medidas de forma indirecta.

**Métricas del Software**

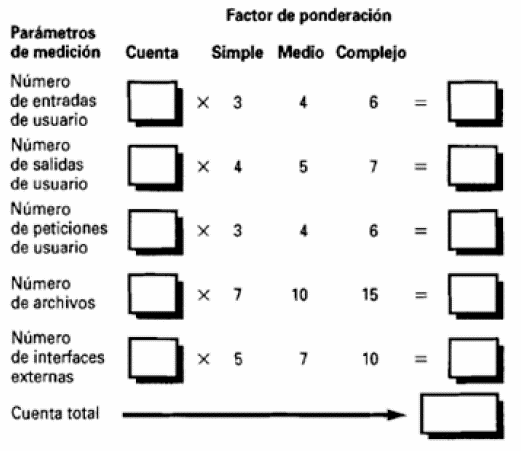
* El dominio de las métricas del software se divide en métricas del proceso, del proyecto y del producto.
* Las métricas del producto, que son **privadas** para un individuo, se combinan para desarrollar métricas del proyecto que son **públicas** para un equipo de software.
* Las métricas del proyecto se consolidan para crear métricas del proceso que sean **públicas** para toda la organización del software.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Proyecto** | **LCD** | **Esfuerzo** | **Coste (000)** | **Pag. Doc.** | **Errores** | **Defectos** | **Personas** |
| Alfa | 12.100 | 24 | 168 | 365 | 134 | 29 | 3 |
| Beta | 27.200 | 62 | 440 | 1224 | 321 | 86 | 5 |
| Gamma | 20.200 | 43 | 314 | 1050 | 256 | 64 | 6 |

* ¿Recuerdan la diferencia entre error y defecto?
* A partir de estos datos obtenidos por una organización del software se pueden desarrollar para cada proyecto métricas simples orientadas al tamaño.
* Errores por KLDC.
* Defectos por KLDC.
* Páginas de documentación por KLDC

**Métricas Orientadas a la Función**

* Las métricas del software orientadas a la función utilizan una medida de la funcionalidad entregada por la aplicación como un valor de normalización. Ya que la funcionalidad no se puede medir directamente, se debe derivar indirectamente mediante otras medidas.
* Se utiliza una medida llamada punto de función.
* Se determinan cinco características de dominios de información y se proporcionan las cuentas en la posición apropiada de la tabla.



Los valores de los dominios de información se definen de la forma siguiente:

* ***Número de entradas de usuario***. Se cuenta cada entrada de usuario que proporciona diferentes datos orientados a la aplicación.
* ***Número de salidas de usuario***. Se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En este contexto la salida se refiere a informes, pantallas, mensajes de error, etc.
* ***Número de peticiones de usuario***. Una petición se define como una entrada interactiva que produce la generación de alguna respuesta del software inmediata en forma de salida interactiva.
* ***Número de archivos***. Se cuenta cada archivo maestro lógico (esto es, un grupo lógico de datos que puede ser una parte de una gran base de datos o un archivo independiente).
* ***Número de interfaces externas***. Se cuentan todas las interfaces legibles por la máquina (por ejemplo: archivos de datos de cinta o disco) que se utilizan para transmitir información a otro sistema.

Una vez que se han recopilado los datos anteriores, a la cuenta se asocia un valor de complejidad. Las organizaciones que utilizan métodos de puntos de función desarrollan criterios para determinar si una entrada en particular es simple, media o compleja. No obstante, la determinación de la complejidad es algo subjetiva.

Características de los Puntos de Función (PF)

Responder a cada una de las siguientes catorce preguntas y asignarles un valor entre 0 y 5, donde 0 es no influencia, 1 es incidental, 2 es moderado, 3 es medio, 4 es significativo y 5 es esencial.

1. ¿Requiere el sistema copias de seguridad y de recuperación fiables?
2. ¿Requiere comunicación de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es crítico el rendimiento?
5. ¿Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado?
6. ¿Requiere entrada de datos interactiva?
7. ¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?
8. ¿Se actualizan los archivos maestros de forma interactiva?
9. ¿Son complejas las entradas, las salidas, los archivos o las peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Están incluidas en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

*Sumar los puntos asignados a cada respuesta y obtener un total F que indica un valor de ajuste de complejidad.*

Cálculo de PF

* El punto de función PF se calcula con la siguiente ecuación:
* Donde T es la suma de todos los puntos de Función no ajustados.
* F es la sumatoria de las características de los puntos de función. (o Grado Total de Influencia)
* A partir de esto, obtenemos los Puntos de Función ajustados.
* En esta fórmula, llamamos factor de ajuste a:
  + Factor de Ajuste

Ejercicio

Calcule el valor del punto de función de un proyecto con las siguientes características del dominio de información:

* Número de entradas de usuario: 32
* Número de salidas de usuario: 60
* Número de peticiones de usuario: 24
* Número de archivos: 8
* Número de interfaces externos: 2
* Asuma que todos los valores de ajuste de complejidad están en la media.

**Técnica de PF**

La técnica de puntos de función fue introducida por Albrecht (1979) y su propósito es medir el software cualificando la funcionalidad que proporciona externamente, basándose en el diseño lógico del sistema. Los objetivos de los puntos de función son:

* Medir lo que el usuario pide y lo que el usuario recibe.
* Medir independientemente la tecnología utilizada en la implantación del sistema.
* Proporcionar una métrica de tamaño que dé soporte al análisis de la calidad y la productividad.
* Proporcionar un medio para la estimación del software.
* Proporcionar un factor de normalización para la comparación de distintos softwares.

Métricas Ampliadas de PF

* Los puntos de función fueron diseñados originalmente para sistemas de información y por eso les da más importancia a los datos y menos a los aspectos de control y funcionales. Para subsanar esto, se han propuesto los puntos de característica y los puntos de función 3D.
* **Puntos de característica:** Se calcula igual que el punto de función y solo agrega la cuenta de algoritmos. En este contexto se define un algoritmo como un problema de cálculo limitado que se incluye dentro de un programa de computadora específico. Invertir una matriz, decodificar un *string* o manejar una interrupción son ejemplos de algoritmos.

Puntos de función 3D

Los puntos de función 3D se calculan llenando la siguiente tabla:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Elemento de Medición** | **Pesos de Complejidad** | | | **Subtotal** |
| **Bajo** | **Medio** | **Alto** |
| Estructuras internas de datos | \*7 | \*10 | \*15 |  |
| Datos externos | \*5 | \*7 | \*10 |  |
| Nº de entradas de usuarios | \*3 | \*4 | \*6 |  |
| Nº de salidas de usuarios | \*4 | \*5 | \*7 |  |
| Nº de peticiones de usuarios | \*3 | \*4 | \*6 |  |
| Transformaciones | \*7 | \*10 | \*15 |  |
| Transiciones |  |  |  |  |
| **Total Puntos de Función 3D** | | | |  |

* Estructuras internas de datos. Son arreglos, listas ligadas, pilas, colas, etc.
* Datos externos. Equivale a la suma de los archivos y las interfaces externas tal y como están definidos para el punto de función.
* Entradas de usuario. Definidas igual que para el punto de función.
* Salidas de usuario. Definidas igual que para el punto de función.
* Peticiones de usuario. Definidas igual que para el punto de función.
* Transformaciones. Son las operaciones internas requeridas para transformar datos de entrada en datos de salida. Multiplicar dos matrices cuenta como una transformación. Leer datos de un archivo y guardarlos en memoria no.
* Transiciones. Ocurre cuando el software pasa de un estado a otro como resultado de algún suceso. En un sistema de altas, bajas y cambios, al tomar la opción de altas, pasa del estado "menú principal" al estado "procesa altas" y puede ser que en ese momento pida datos para dar el alta.

Ejercicio

Calcule el valor del punto de función de un sistema empotrado con las características siguientes:

* Estructuras de datos interna: 6
* Estructuras de datos externa: 3
* Número de entradas de usuario: 12
* Número de salidas de usuario: 60
* Número de peticiones de usuario: 9
* Número de interfaces externos: 3
* Transformaciones: 36
* Transiciones: 24

Asuma que la complejidad de las cuentas anteriores se divide de igual manera entre bajo, medio y alto.

Luego que se realizaron los anteriores cálculos, se realizan los mismos pasos que para puntos de función.

El punto de función (y sus extensiones), como la medida LDC, es controvertido. Los partidarios afirman que PF es independiente del lenguaje de programación, lo cual es ideal para aplicaciones que utilizan lenguajes convencionales y no procedimentales; esto se basa en los datos que probablemente se conocen al principio de la evolución de un proyecto, y así el PF es más atractivo como enfoque de estimación.

Reconciliación de las diferentes métricas

La relación entre las líneas de código y los puntos de función depende del lenguaje de programación que se utilice para implementar el software, y de la calidad del diseño.

La tabla siguiente proporciona estimaciones informales del número medio de líneas de código que se requiere para construir un punto de función en varios lenguajes de programación:

|  |  |
| --- | --- |
| **Lenguaje** | **LCD/PF** |
| Ensamblador | 320 |
| C | 150 |
| Cobol | 105 |
| Pascal | 91 |
| Prolog/LISP | 64 |
| C++ | 64 |
| FoxPro | 31 |
| Visual Basic | 30 |
| Delphi | 29 |

Ejercicio

El software utilizado para controlar una fotocopiadora avanzada requiere 32.000 líneas de C y 4.200 líneas de C++.

Estime el número de puntos de función del software de la fotocopiadora.

(Resultados: 180 pf)

**Métricas para la calidad**

* Aunque hay muchas medidas de la calidad de software**, la corrección, facilidad de mantenimiento, integridad, y facilidad de uso** proporcionan indicadores útiles para el equipo del proyecto.
* **Corrección**. Un programa debe operar correctamente o proporcionará poco valor a sus usuarios. La corrección es el grado en el que el software lleva a cabo su función requerida. La medida más común de corrección es defectos por KLDC, en donde un defecto se define como una falta verificada de conformidad con los requisitos.
* **Facilidad de mantenimiento**. El mantenimiento del software cuenta con más esfuerzo que cualquier otra actividad de ingeniería del software. La facilidad de mantenimiento es la facilidad con la que se puede corregir un programa si se encuentra un error, se puede adaptar si su entorno cambia, o mejorar si el cliente desea un cambio de requisitos. No hay forma de medir directamente la facilidad de mantenimiento; por consiguiente, se deben utilizar medidas indirectas. Una simple métrica orientada al tiempo es el **tiempo medio de cambio (TMC)**, es decir el tiempo que se tarda en analizar la petición de cambio, hasta aplicar.
* **Integridad**. En esta época de hackers y firewalls, la integridad del software ha llegado a tener mucha importancia. Este atributo mide la capacidad de un sistema para resistir ataques (tanto accidentales como intencionados) contra su seguridad. El ataque se puede realizar en cualquiera de los tres componentes del software: programas, datos y documentos. **Amenaza** es la probabilidad (que se puede estimar o deducir de la evidencia empírica) de que un ataque de un tipo determinado ocurra en un tiempo determinado. La **seguridad** es la probabilidad (que se puede estimar o deducir de la evidencia empírica) de que se pueda repeler el ataque de un tipo determinado. La integridad del sistema se puede definir como:
* **Facilidad de uso**. El calificativo amigable con el usuario se ha convertido en omnipresente en las discusiones sobre productos de software. Si un programa no es amigable con el usuario frecuentemente está abocado al fracaso, incluso aunque las funciones que realice sean valiosas. La facilidad de uso es un intento de cuantificar lo amigable que puede ser con el usuario y se puede medir en función de cuatro características:
  + habilidad intelectual y/o física requerida para aprender el sistema;
  + el tiempo requerido para llegar a ser moderadamente eficiente en el uso del sistema;
  + aumento neto en productividad (sobre el enfoque que el sistema reemplaza) medida cuando alguien utiliza el sistema moderadamente y eficientemente;
  + valoración subjetiva (a veces obtenida mediante un cuestionario) de la disposición de los usuarios hacia el sistema.

**Métrica de calidad: EED**

* Una métrica de la calidad que proporciona beneficios tanto a nivel del proyecto como del proceso, es la eficacia de la eliminación de defectos (EED).
* EED es una medida de la habilidad de filtrar las actividades de la garantía de calidad y de control al aplicarse a todas las actividades del marco de trabajo del proceso.
* El valor ideal de EED es 1. Esto es, no se han encontrado defectos en ei software. De forma realista, D será mayor que cero, pero el valor de EED todavía se puede aproximar a 1. Cuando E aumenta (para un valor de D dado), el valor total de EED empieza a aproximarse a 1. De hecho, a medida que E aumenta, es probable que el valor final de D disminuya.

**Métricas del modelo de análisis: calidad de la especificación**

* Propuestas por Alan Davis en 1993.
* Corresponden a la calidad de la especificación de requerimientos.
* Incluyen las siguientes características:
  + Especificidad, compleción, corrección, comprensión, verificación, consistencia, logro, concisión, trazabilidad, modificación, exactitud, reutilización.
* Cualitativas, pero algunas pueden medirse.
* Sea
* donde

**Especificidad (ausencia de ambigüedad)**

* donde

**Grado de Validación de Requerimientos**

* donde

**Métricas de mantenimiento**

En el estándar IEEE 982.1-1998, se propone un Índice de Madurez del Software (IMS).

* Proporciona una indicación de la estabilidad del producto de software, basada en los cambios de cada versión del producto.
* donde

**Métricas de documentación**

Índice de Fog (está pensado para el idioma inglés):

* Para evaluar la dificultad de lectura de un texto.
* Medida de la longitud promedio de las palabras y declaraciones en los documentos.
* Cuanto mayor sea el índice de Fog, más difícil será comprender el documento.

Si Texto no comprensible

**Cocomo**

* Es un modelo de estimación de costes.
* Creado por Barry W. Boehm.
* Incluye 3 submodelos con un nivel de detalle cada vez mayor.
* Está basado en modelos de estimaciones matemáticas.
* Está orientado al producto final, no a fases intermedias.
* Se basa en la cantidad de líneas de código del proyecto.

Inconvenientes del modelo

* Comentarios en líneas de código.
* Estimaciones sobre un nº de líneas de código variable.
* No se le da importancia a la productividad, referente a los hábitos de trabajo
* Dificultad para contemplar costes de revisiones, reuniones.

Modelos de estimación

* **Modelo básico.**
* **Modelo intermedio.**
* **Modelo avanzado.**

Modos

* **Orgánico**: proyectos relativamente sencillos, menores de 50.000 líneas de código. Se tiene experiencia en proyectos similares y se encuentra en un entorno estable.
* **Semiorgánico**: proyectos intermedios en complejidad y tamaño. La experiencia en este tipo de proyectos es variable, y las restricciones intermedias.
* **Empotrado**: proyectos bastante complejos, en los que apenas se tiene experiencia y en un entorno de gran innovación técnica. Se trabaja con unos requisitos muy restrictivos y de gran volatilidad.

Estimación por modelo Cocomo

* Para estimar costos, esfuerzo y tiempo requerido para desarrollar el proyecto, se utilizará al modelo de COCOMO (COnstructive COst MOdel).
* Para calcular el Esfuerzo, necesitaremos hallar la variable KDLC (Kilo-líneas de código), donde necesitaremos conocer las líneas por cada punto de función según el lenguaje utilizado.

|  |  |
| --- | --- |
| **Lenguaje** | **LCD/PF** |
| Ensamblador | 320 |
| C | 150 |
| Cobol | 105 |
| Pascal | 91 |
| Prolog/LISP | 64 |
| C++ | 64 |
| FoxPro | 31 |
| Visual Basic | 30 |
| Delphi | 29 |

Las formulas a aplicar son

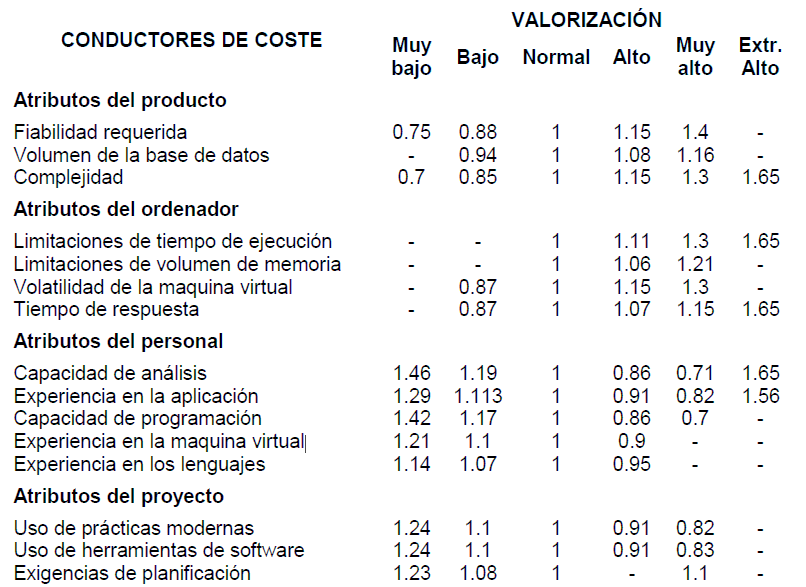
* E = Esfuerzo = a KLDC e \* FAE (persona x mes)
* T = Tiempo de duración del desarrollo = c Esfuerzo d (meses)
* P= Personal = E/T (personas)
* Si el desarrollo se realiza en FoxPro, y la cantidad de PFA es de 2350, entonces el KLDC seria: (31 \* 2350) / 1000.

Este proyecto será considerado como software semiorgánico, ya que posee más de 50.000 líneas de código, y además el mismo presenta una complejidad y tamaño intermedio.

Coeficientes a utilizar por tipo de proyecto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Proyecto de sofware** | **A** | **E** | **C** | **D** |
| ***Orgánico*** | 3.2 | 1.05 | 2.5 | 0.38 |
| ***Semi – acoplado*** | 3.0 | 1.12 | 2.5 | 0.35 |
| ***Empotrado*** | 2.8 | 1.2 | 2.5 | 0.32 |

También, se requiere conocer el valor de la variable FAE (Factor de ajuste de esfuerzo) que se obtiene mediante la multiplicación de los valores evaluados en los 15 conductores de coste que se observan en la tabla siguiente.



FAE – Justificación

Se deben justificar cada uno de los valores seleccionados, para cada atributo de coste.

Ejemplo:

* **Fiabilidad requerida del software:** si se produce un fallo ocasionarían pérdidas a los clientes que utilicen el software (Valoración Normal).
* **Tamaño de la base de datos:** la base de datos del sistema será relativamente grande (Valoración Normal).
* **Complejidad del producto:** la aplicación va a realizar cálculos sencillos (valoración baja).

Cálculo de esfuerzo

Con todos los datos que hemos obtenido, ya podemos calcular el esfuerzo:

* (persona x mes)

Cálculo de la duración del desarrollo

* (meses)
* aproximadamente 14 meses.

Cálculo de personal por mes

* (personas)
* aproximadamente 9 personas x mes, para cumplir el proyecto en 14 meses. Esto debe ajustarse según los tiempos planificados.

Cálculo de productividad

* **Esto da una estimación de la cantidad de líneas de código por persona por mes a generar.**

Costo estimado de la codificación

* Costo estimado de codificación =
* Costo estimado de codificación = $ 38400
* Este costo solo hace referencia a la etapa de codificación, no se tienen en cuenta las demás etapas.

**Modelo Intermedio**

* Añade al modelo básico 15 factores de ajuste o guías de coste.
* Logramos mayor precisión en la estimación gracias a los nuevos factores.
* La fórmula es la misma que la del modelo básico, pero con el añadido del factor (multiplicando).

Atributos del modelo

* Software
  + RELY: Indica las consecuencias para el usuario si falla el producto.
  + DATA: Relación Tamaño de la BD / Líneas de código.
  + CPLX: Complejidad del producto.
* Hardware
  + TIME: Limitaciones en el porcentaje del uso de la CPU.
  + STOR: Limitaciones en el porcentaje del uso de la memoria.
  + VIRT: Volatilidad de la máquina virtual.
  + TURN: Tiempo de respuesta.
* Personal
  + ACAP: calificación de los analistas.
  + AEXP: experiencia del personal.
  + PCAP: calificación de los programadores.
  + VEXP: experiencia del personal en la máquina virtual.
  + LEXP: experiencia en el lenguaje.
* Proyecto
  + MODP: uso de prácticas modernas de programación.
  + TOOL: uso de herramientas de desarrollo de software.
  + SCED: limitaciones en el cumplimiento de la planificación.

Ejemplo de estimación

Debemos desarrollar un software de no muy elevada dificultad, con las siguientes restricciones:

* 3 meses para el desarrollo del proyecto software.
* Debe estar implementado en el lenguaje Visual Basic.
* Cálculo del esfuerzo: necesitamos hallar la variable KDLC.

|  |  |
| --- | --- |
| **LENGUAJE** | **LDC/PF** |
| Ensamblador | 320 |
| **C** | 150 |
| **COBOL** | 105 |
| **Pascal** | 91 |
| **Prolog/LISP** | 64 |
| **C++** | 64 |
| **Visual Basic** | 32 |
| **SQL** | 12 |

Usaremos el tipo Orgánico ya que nuestro proyecto no supera las 50 KLDC, y es el mas a apropiado en este caso.

Coeficientes a usar:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PROYECTO SOFTWARE** | **a** | **b** | **c** | **d** |
| **Orgánico** | 3,2 | 1,05 | 2,5 | 0,38 |
| **Semi-acoplado** | 3,0 | 1,12 | 2,5 | 0,35 |
| **Empotrado** | 2,8 | 1,20 | 2,5 | 0,32 |

* Cálculo de la variable FAE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CONDUCTORES DE COSTE** | **VALORACIÓN** | | | | | |
| ***Muy bajo*** | ***Bajo*** | ***Nominal*** | ***Alto*** | ***Muy alto*** | ***Extr. alto*** |
| Fiabilidad requerida del software | 0,75 | 0,88 | 1.00 | **1,15** | 1,40 | - |
| Tamaño de la base de datos | - | 0,94 | **1.00** | 1,08 | 1,16 | - |
| Complejidad del producto | 0,70 | **0,85** | 1.00 | 1,15 | 1,30 | 1,65 |
| Restricciones del tiempo de ejecución | - | - | 1.00 | **1,11** | 1,30 | 1,66 |
| Restricciones del almacenamiento principal | - | - | **1.00** | 1,06 | 1,21 | 1,56 |
| Volatilidad de la máquina virtual | - | 0,87 | **1.00** | 1,15 | 1,30 | - |
| Tiempo de respuesta del ordenador | - | 0,87 | 1.00 | **1,07** | 1,15 | - |
| Capacidad del analista | 1,46 | 1,19 | 1.00 | **0,86** | 0,71 | - |
| Experiencia en la aplicación | 1,29 | 1,13 | 1.00 | 0,91 | **0,82** | - |
| Capacidad de los programadores | 1,42 | 1,17 | 1.00 | 0,86 | **0,70** | - |
| Experiencia en S.O. utilizado | 1,21 | 1,10 | **1.00** | 0,90 | - | - |
| Experiencia en el lenguaje de programación | 1,14 | 1,07 | 1.00 | **0,95** | - | - |
| Prácticas de programación modernas | 1,24 | 1,10 | **1.00** | 0,91 | 0,82 | - |
| Utilización de herramientas software | 1,24 | 1,10 | 1.00 | **0,91** | 0,83 | - |
| Limitaciones de planificación del proyecto | 1,23 | **1,08** | 1.00 | 1,04 | 1,10 | - |

* Cálculo del esfuerzo del desarrollo:
* Cálculo tiempo de desarrollo:
* Productividad:
* Personal promedio:

Según los resultados necesitaremos un equipo de 3 personas trabajando alrededor de 7 meses, pero como una restricción era 3 meses incrementamos a 6 el número de personas. 1 Jefe de proyecto, 2 Analistas, 2 programadores y 1 Responsable de calidad.

Conclusiones

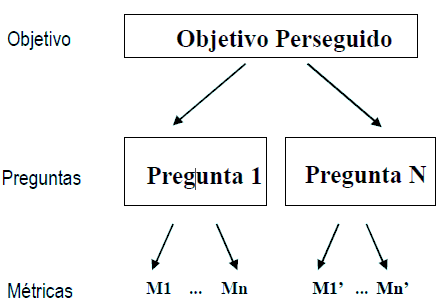
* Medición Esencial si se desea conseguir calidad.
* Métricas de producto: proporcionan la visión interna necesaria para crear modelos efectivos de análisis y diseño, código sólido y pruebas minuciosas.
* Medidas absolutas: no existen en software se obtienen medidas indirectas que nos dan indicaciones de las representaciones del software.

**Programa de Métricas de Software dirigido por Metas**

* Identificar **objetivos** de la empresa.
* Identificar lo que se quiere conocer o aprender.
* Identificar los **subobjetivos**.
* Identificar **entidades** y **atributos** relacionados con los objetivos secundarios.
* Formalizar los objetivos de la medición.
* Identificar **preguntas cuantificables** y los **indicadores** relacionados que se emplearán como apoyo para lograr los objetivos.
* Identificar elementos de datos que se recopilarán para construir los indicadores para responder las preguntas.
* Definir las medidas e identificar acciones para implementarlas medidas.
* Preparar un plan para implementar las medidas.

**Paradigma GQM (Goal – Question – Metric) [Basili Weiss, 1984]**

Fundamento:

* “La medición debe basarse en modelos y objetivos”.

Debemos entonces establecer objetivos medibles y dirigidos por el modelo apropiado.

GQM:

* Se utiliza a nivel de proceso y proyecto para decidir qué mediciones hacer y cómo utilizarlas.
* Provee un marco de trabajo para interpretar los datos y entender el enfoque sobre los objetivos.
* **Metas** (goals): aquello que la organización o el proyecto intenta alcanzar. Se define un objetivo para una entidad. Por ejemplo:

1. Mejorar la productividad de los programadores,
2. Reducir tiempos de desarrollo,
3. Incrementar la confiabilidad del producto,
4. Etc.

* **Preguntas** (questions): refinamientos de las metas en las que se identifican áreas específicas de incertidumbre. Ejemplos:

1. (productividad) ¿cómo se puede incrementar el número de LDC depuradas?
2. (tipo desarrollo) ¿cómo se puede reducir el tiempo requerido para finalizar la implementación de los requerimientos?
3. (confiabilidad) ¿cómo se pueden llevar a cabo evaluaciones de confiabilidad más efectivas?

* **Métricas**: las mediciones necesarias para ayudar a responder a las preguntas y confirmar si las mejoras del proceso o proyecto cumplieron su objetivo. Ejemplos:

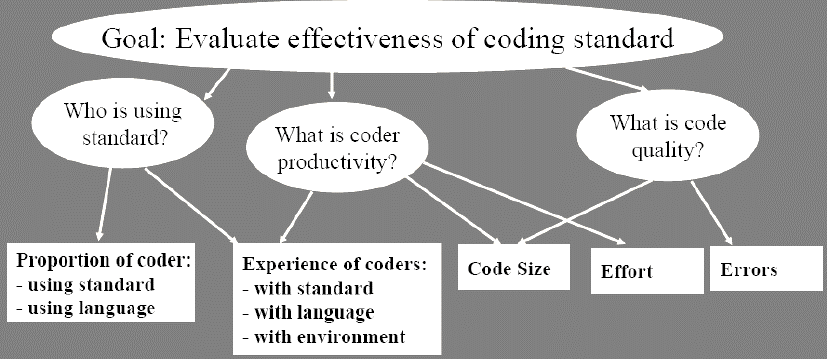
1. (productividad) medir la productividad de los programadores en LDC y nivel de experiencia.
2. (tipo desarrollo) medir número de comunicaciones formales entre cliente y analista para cada cambio de requerimientos.
3. (confiabilidad) medir el número de pruebas requeridas para provocar una caída en el producto.

GQM - Pasos Básicos

1. Desarrollar el conjunto de objetivos corporativos, de división y proyecto.
2. Construir programas para cada objetivo que lo definen en la forma más completa posible.
3. Especificar las métricas requeridas para contestar las preguntas.
4. Construir los procesos de recolección de datos.
5. Recolectar, validar y analizar los datos obtenidos.

Ejemplo 1

*Objetivo:* “efectividad de usar estándares para la codificación.”



Ejemplo 2

*Objetivo*: “Mejorar la Planificación del Tiempo”

* *Pregunta*: ¿Cuál fue la exactitud de la estimación del cronograma del proyecto?
* *Métrica*: Exactitud en Estimación del Cronograma (EEC).

Ejemplo 3

*Objetivo:* “Rediseñar la página web para facilitar su uso”

* *Preguntas:*
  + ¿Qué enlaces se utilizan más?
  + ¿Cuán fácil es acceder a la información?
* *Métricas:*
  + Cantidad de accesos a los enlaces.
  + Cantidad de saltos (clicks) desde la página de inicio.

GQM – Propuesta de Pasos

1. Identificar diversas entidades que necesitan ser evaluadas para lograr la meta planteada.

* productos, recursos, artefactos, actividades
* código fuente, casos de prueba, solicitudes de cambio, tareas de ingeniería de software, planificación, etc.

1. Identificar preguntas asociadas a cada una de las entidades definidas del proyecto relacionadas con el objetivo planteado.
2. Considerando las preguntas definidas:

* Agrupar preguntas relacionadas (agrupando cuestiones relativas a algún área particular, al producto de software, gestión de proyecto, gestión de cambios, interacciones con el cliente).
* Identificar para cada grupo una submeta o subobjetivo relacionado con el original.

1. Identificar subentidades y atributos necesarios a evaluar para las submetas definidas en el punto anterior.
2. Definir de manera completa métricas e indicadores para los atributos identificados en el punto anterior.

GQM – Ejercicio

1. Entidades:

* Requerimientos de software.
* Manual de Usuario.
* Interfaces.
* Casos de prueba.
* Plazos.
* Contrato.

1. Preguntas para las entidades:

* Requerimientos de software:
  + ¿Se elicitaron completamente? ¿Se cumplieron en su totalidad?
* Documentación:
  + ¿Es completa? ¿Es entendible? ¿Es útil? …
* Interfaces:
  + ¿Son atractivas? ¿Muestran la información deseada? ¿El diseño es intuitivo? …
* Casos de prueba.
  + ¿Son suficientes los casos de prueba? ¿Cuántos defectos encontró el cliente? …
* Plazos.
* Contrato.

1. a) Áreas:

* Relación con el cliente.
* Calidad del producto de software.
* Calidad de la documentación.

b) Submetas relacionadas con la original

* Elicitar y cumplir la totalidad de requerimientos.
* Mejorar la interacción con el cliente.
* Minimizar los defectos encontrados por el cliente.
* Mejorar la calidad de la documentación del producto.

1. Subentidades y atributos:

* **Requerimientos**: Requerimientos encontrados, requerimientos cumplidos, requerimientos no cumplidos, requerimientos modificados durante el proceso de desarrollo, etc.
* **Defectos y Errores**: Defectos encontrados antes de la entrega, errores encontrados por el cliente, etc.
* **Manual de Usuario**: tamaño, legibilidad, consistencia, errores, modificabilidad, etc.

1. Métricas:

* **Requerimientos**:
  + #Reqs Cumplidos / #Reqs Totales
  + #Reqs Validados / #Reqs Totales
* **Defectos y Errores**:
  + #Defectos / (#Defectos + #Errores)
* **Manual de Usuario**:
  + Índice Fog: Medida de la longitud promedio de las palabras y declaraciones en los documentos  a mayor índice de Fog, más difícil será comprender el documento.

¿qué está faltando? INDICADORES

**Métricas Privadas vs Métricas Públicas**

Métricas Públicas: Engloban información originalmente privada para particulares y equipos.

* Ejemplos: esfuerzo total, tiempo, etc.

Algunas métricas son:

* Privadas para el equipo de un proyecto y públicas para la organización.
* Ejemplo: índice de defectos por proyecto y general.

Métricas - Privacidad según niveles

* Propone clasificar las métricas directas o básicas en algún nivel de privacidad:
  + **Individual**: sólo los individuos que recolectan los datos sobre su propio trabajo conocen su información. Por ej.: horas semanales individuales dedicadas a actividades de desarrollo.
  + **Equipo de proyecto**: Los datos son privados para el equipo de proyecto. Por ej.: horas semanales totales dedicadas a actividades de desarrollo.
  + **Organización**: los datos son compartidos a lo largo de la organización. Por ej.: horas semanales promedio de desarrollo a lo largo de todos los proyectos.

ISO / IEC 15939 – 2007

Define un proceso de medición para el desarrollo de software e ingeniería de sistemas.

|  |  |
| --- | --- |
| **ACTIVIDAD** | **TAREAS** |
| Establecer y Mantener el  Compromiso de Medición | Aceptar los requisitos de la medición |
| Asignar recursos |
| Planificar el Proceso de Medición | Obtener las características de la organización |
| Identificar las necesidades de información |
| Seleccionar las medidas |
| Definir los procedimientos de recolección de datos, análisis e informes |
| Definir los criterios de evaluación de los productos de información y el proceso de medición |
| Revisar, aprobar y proporcionar recursos para las tareas de medición |
| Adquirir y utilizar tecnologías de apoyo |
| Realizar el Proceso de Medición | Integrar los procedimientos |
| Recoger los datos |
| Analizar los datos y desarrollar productos de información |
| Comunicar los resultados |
| Evaluar la Medición | Evaluar los productos de información y el proceso de medición |
| Identificar las mejoras potenciales |

