1. CONCEPTOS DE SOFTWARE E INGENIERÍA DE SOFTWARE

¿QUÉ ES SOFTWARE?

Por lo general, un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema y sitios web que permitan a los usuarios descargar la información de productos recientes.

Tipos de productos de software:

- Productos genéricos. Son sistemas aislados producidos por una organización de desarrollo y que se venden al mercado abierto a cualquier cliente que le sea posible comprarlos.
- Productos personalizados (o hechos a medida). Son sistemas requeridos por un cliente en particular.

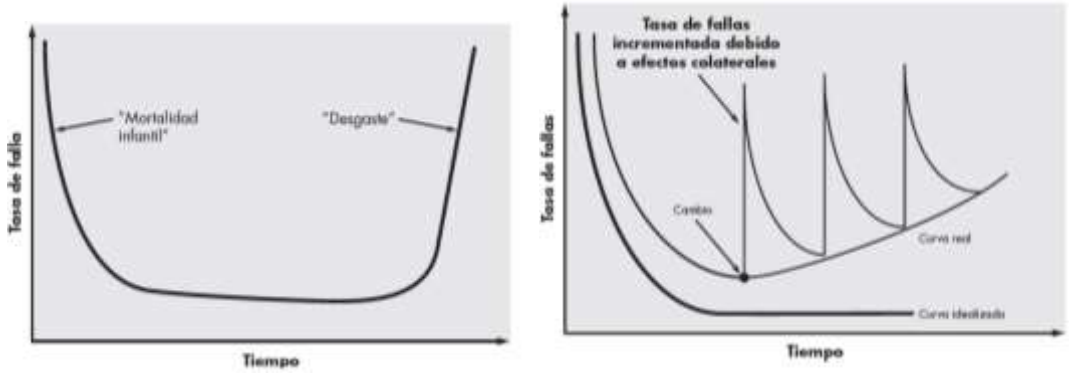
Hay una característica del software que debe atenderse: el hecho que es un sistema. Teniendo presente este aspecto, se puede afirmar que el cambio es posible e inevitable. Pero también se advierte que al percibir al software como sistema, la gestión del cambio presenta retos de mayor complejidad

El software de computadora es un ejemplo de la ley de las consecuencias inesperadas. Hace 50 años, nadie hubiera podido predecir que el software se convertiría en una tecnología indispensable para los negocios, ciencias e ingeniería, ni que permitiría la creación de tecnologías nuevas, ampliación de tecnologías ya existentes y el cambio radical de tecnologías antiguas.

La carga de ejecutar las actividades de "mantenimiento" absorben más personas y recursos que todo el trabajo aplicado a la creación de software nuevo. La comunidad de programadores ha tratado continuamente de desarrollar tecnologías que hagan más fácil, rápida y barata la elaboración de programas de cómputo de alta calidad. Sin embargo, todavía falta por desarrollarse una tecnología de software que haga todo esto, y hay pocas probabilidades de que surja una en el futuro (No silver bullet).

El software es elemento de un sistema lógico y no de uno físico. Por tanto, tiene características que difieren considerablemente de las del hardware:

- El software se desarrolla o modifica con intelecto; no se manufactura en el sentido clásico: tanto en el desarrollo de software y la fabricación de hardware, la alta calidad se logra a través de un buen diseño. Sin embargo, la fase de manufactura del hardware introduce problemas de calidad que no existen (o que se corrigen con facilidad) en el software.
- En la figura 1 se ilustra la tasa de falla del hardware como función del tiempo. La relación, que es frecuente llamar "curva de tina", indica que el hardware presenta una tasa de fallas relativamente elevada en una etapa temprana de su vida (frecuentemente atribuibles a defectos de diseño o manufactura); los defectos se corrigen y la tasa de fallas se abate a un nivel estable durante cierto tiempo. No obstante, conforme pasa el tiempo, la tasa de fallas aumenta de nuevo a medida que los componentes del hardware comienzan a desgastarse (por suciedad, vibración, abuso, temperaturas extremas). El software no es susceptible a los problemas ambientales. Por tanto, en teoría, la curva de la tasa de fallas adopta la forma de la "curva idealizada" que se aprecia en la figura 2. Los defectos ocultos ocasionarán tasas elevadas de fallas al comienzo de la vida de un programa, pero éstas se corrigen y la curva se aplana. Sin embargo, el software sí se deteriora debido a los cambios. Se considera que la curva real en la figura 2. Cuando un componente del hardware se desgasta es sustituido por una refacción. En cambio, no hay refacciones para el software.



- Aunque la industria se mueve hacia la construcción basada en componentes, la mayor parte del software se construye para un uso individualizado. Los componentes reutilizables han sido creados para que el ingeniero pueda concentrarse en los elementos verdaderamente innovadores de un diseño; es decir, en las partes de éste que representan algo nuevo.

La naturaleza del software: El software tiene un papel dual. Es un producto y al mismo tiempo es el vehículo, por el cual se entrega el producto más importante de nuestro tiempo: información. En su forma de producto, hace entrega de la potencia informática del hardware informático. Como vehículo para hacer entrega del producto, el software actúa como la base de control de la computadora (sistemas operativos), la comunicación de información (redes), y la creación y control de otros programas (herramientas de software y entornos).

Evolución del software: En el último medio siglo, el papel del software de cómputo ha sufrido un cambio significativo. Las notables mejoras en el funcionamiento del hardware, los profundos cambios en las arquitecturas de computadora, el gran incremento en la memoria y capacidad de almacenamiento, y una amplia variedad de opciones de entradas y salidas exóticas han propiciado la existencia de sistemas basados en computadora más sofisticados y complejos.

En la actualidad, la enorme industria del software se ha convertido en un factor dominante en las economías del mundo industrializado. Equipos de especialistas de software han reemplazado al programador solitario de los primeros tiempos.

Características:

- Maleabilidad: todo el mundo exige que se realicen cambios sobre el software como respuesta a pequeños cambios en el entorno. Es la capacidad de modificación indefinida, se asocia con la aparente facilidad para la adaptación al cambio.
- El software es único.
- Es intangible
- Alto contenido intelectual
- Su proceso de desarrollo es mano de obra intensivo, basado en equipos y por proyectos
- No hay separación entre R&D (investigación y desarrollo) y producción: mientras vamos investigando, vamos desarrollando.

Características conceptuales

- Su producción es humano-intensiva.
- Tradicionalmente en la ingeniería el ingeniero dispone de herramientas para describir el producto que son distintas del producto, no es así en la ingeniería de software.
- Las cualidades del producto de software están a menudo entremezcladas en especificaciones con las cualidades del diseño.

Mitos

Los mitos del software son creencias erróneas sobre éste y sobre el proceso que se utiliza para obtenerlo.

Mitos de la administración: Los gerentes que tienen responsabilidades en el software con frecuencia se hallan bajo presión para cumplir el presupuesto, mantener la programación de actividades sin desvíos y mejorar la calidad. Ejemplo de estos son los 3 primeros mitos.

Mitos del cliente: El cliente que requiere software de computadora sostiene mitos sobre el software porque los gerentes y profesionales de éste hacen poco para corregir la mala información. Los mitos generan falsas expectativas (por parte del cliente) y, en última instancia, la insatisfacción con el desarrollador. Ejemplo de estos es el 4 y 5 mito.

Mitos del profesional: Los mitos que aún sostienen los trabajadores del software han sido alimentados por más de 50 años de cultura de programación. Durante los primeros días, la programación se veía como una forma del arte. Es difícil que mueran los hábitos y actitudes arraigados. Ejemplo de estos son los últimos 4.

Mitos	**Realidad**
Tenemos un libro lleno de estándares y procedimientos para elaborar software. ¿No le dará a mi personal todo lo que necesita saber?	Tal vez exista el libro de estándares, pero ¿se utiliza? ¿Saben de su existencia los trabajadores del software? ¿Refleja la práctica moderna de la ingeniería de software? ¿Es completo? ¿Es adaptable? ¿Está dirigido a mejorar la entrega a tiempo y también se centra en la calidad?
Si nos atrasamos, podemos agregar más programadores y ponernos al corriente	A medida que se agregan personas, las que ya se encontraban trabajando deben dedicar tiempo para enseñar a los recién llegados, lo que disminuye la cantidad de tiempo dedicada al esfuerzo de desarrollo productivo. Pueden agregarse individuos, pero sólo en forma planeada y bien coordinada.
Si decido subcontratar el proyecto de software a un tercero, puedo descansar y dejar que esa compañía lo elabore.	Si una organización no comprende cómo administrar y controlar proyectos de software internamente, de manera invariable tendrá dificultades cuando subcontrate proyectos de software.
Para comenzar a escribir programas, es suficiente el enunciado general de los objetivos —podremos entrar en detalles más adelante.	Aunque no siempre es posible tener el enunciado exhaustivo y estable de los requerimientos, un "planteamiento de objetivos" ambiguo es una receta para el desastre.
Los requerimientos del software cambian continuamente, pero el cambio se asimila con facilidad debido a que el software es flexible.	Cuando se solicitan al principio cambios en los requerimientos (antes de que haya comenzado el diseño o la elaboración de código), el efecto sobre el costo es relativamente pequeño. Sin embargo, conforme pasa el tiempo, el costo aumenta con rapidez: los recursos ya se han comprometido, se ha establecido la estructura del diseño y el cambio ocasiona perturbaciones que exigen recursos adicionales y modificaciones importantes del diseño.
Una vez que escribimos el programa y hacemos que funcione, nuestro trabajo ha terminado.	Los datos de la industria indican que entre 60 y 80% de todo el esfuerzo dedicado al software ocurrirá después de entregarlo al cliente por primera vez.
Hasta que no se haga "correr" el programa, no hay manera de evaluar su calidad.	Uno de los mecanismos más eficaces de asegurar la calidad del software puede aplicarse desde la concepción del proyecto: *la revisión técnica*.
El único producto del trabajo que se entrega en un proyecto exitoso es el programa que funciona.	Son varios los productos terminados (modelos, documentos, planes) que proporcionan la base de la ingeniería exitosa y, lo más importante, que guían el apoyo para el software.
La ingeniería de software hará que generemos documentación voluminosa e innecesaria, e invariablemente nos retrasará.	La ingeniería de software no consiste en producir documentos. Se trata de crear un producto de calidad.

Crisis**:** La crisis del software se fundamentó en el tiempo de creación de software, ya que en la creación del mismo no se obtenían los resultados deseados, además de un gran costo y poca flexibilidad.

Básicamente, la crisis del software se refiere a la dificultad en escribir programas libres de defectos, fácilmente comprensibles, y que sean verificables. Las causas son, entre otras, la complejidad que supone la tarea de programar, y los cambios a los que se tiene que ver sometido un programa para ser continuamente adaptado a las necesidades de los usuarios.

Montenegro, Micaela Soledad

Además, no existen todavía herramientas que permitan estimar de una manera exacta, antes de comenzar el proyecto, cuál es el esfuerzo que se necesitará para desarrollar un programa.

Esto englobó a una serie de sucesos que se venían observando en los proyectos de desarrollo de software:

- Los proyectos no terminaban en plazo.
- Los proyectos no se ajustaban al presupuesto inicial.
- Baja calidad del software generado.
- Software que no cumplía las especificaciones.
- Código inmantenible que dificultaba la gestión y evolución del proyecto.

Aunque se han propuesto diversas metodologías para intentar subsanar los problemas mencionados, lo cierto es que todavía hoy no existe ningún método que haya permitido estimar de manera fiable el coste y duración de un proyecto antes de su comienzo.

INGENIERÍA DEL SOFTWARE

Es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después de que se utiliza, para obtener software económico que sea confiable y funcione de manera eficiente.

Realidades:

- Cuando ha de construirse una aplicación nueva o sistema incrustado, deben escucharse muchas opiniones. Y en ocasiones parece que cada una de ellas tiene una idea un poco distinta de cuáles características y funciones debiera tener el software. *Se concluye que debe hacerse un esfuerzo concertado para entender el problema antes de desarrollar una aplicación de software.*
- La *complejidad* de estos nuevos sistemas y productos demanda atención cuidadosa a las interacciones de todos los elementos del sistema. *Se concluye que el diseño se ha vuelto una actividad crucial.*
- Los individuos, negocios y gobiernos dependen cada vez más del software para tomar decisiones estratégicas y tácticas, así como para sus operaciones y control cotidianos. Si el software falla, las personas y empresas grandes pueden experimentar desde un inconveniente menor hasta fallas catastróficas. Se concluye que *el software debe tener alta calidad.*
- Conforme se extienda su base de usuarios y el tiempo de uso, las demandas para adaptarla y mejorarla también crecerán. *Se concluye que el software debe tener facilidad para recibir mantenimiento.*

La ingeniería de software es una tecnología con varias capas, y como cualquier enfoque de ingeniería debe basarse en un compromiso organizacional con la calidad:

El fundamento para la ingeniería de software es la capa **proceso**, que define una estructura que debe establecerse para la obtención eficaz de tecnología de ingeniería de software. Es un enfoque adaptable que permite que el equipo de software busque y elija el conjunto apropiado de acciones y tareas para el trabajo.

Los **métodos** de la ingeniería de software incluyen un conjunto amplio de tareas, como comunicación, análisis de los requerimientos, modelación del diseño, construcción del programa, pruebas y apoyo. Se basan en un conjunto de principios fundamentales que gobiernan cada área de la tecnología e incluyen actividades de modelación y otras técnicas descriptivas.

Las **herramientas** de la ingeniería de software proporcionan un apoyo automatizado o semiautomatizado para el proceso y los métodos.

Principios generales (sobre estos escalonamos sobre las técnicas y herramientas)

- Rigor y formalidad: Sólo una aproximación rigurosa puede producir productos más confiables, controlar sus costos e incrementar su confiabilidad.
- Separación de intereses. Se puede hacer según varios criterios:
    - Tiempo (ciclo de vida del software)
    - Cualidades
    - Vistas (flujo de datos, de control, estático, dinámico, etc.)
    - Partes (estructura, diseño)
- Modularidad. No sólo aplica a los aspectos estructurales, sino a todo el proceso de desarrollo.
- Abstracción. Los modelos que construimos para entender los fenómenos son abstracciones de la realidad.
- Anticipación del cambio
- Generalidad. Focalizar su atención en el descubrimiento de un problema más general que puede estar oculto detrás del problema en cuestión
- Incrementalidad. Este principio puede aplicarse al identificar tempranamente subconjuntos útiles de una aplicación para así obtener rápido feedback. Cada incremento debe tener nuevas tareas.

¿Cómo obtener hoy un producto de calidad?

- No existe la bala de plata
- No sólo que funcione, sino que le guste al cliente y que le sirva.
- Responsabilidad profesional
- Prácticas consensuadas. Estás salen de asociaciones profesionales, centros especializados, profesionales prestigiosos, trabajos especializados, etc. Algunas "best practices" son:
    - Análisis de factibilidad
    - Análisis de riesgo
    - Planificación/seguimiento
    - Control de configuraciones
    - Automatizar – uso de herramientas
    - Análisis de requerimientos
    - Diseño
    - Inspecciones y revisiones
    - testing

¿Qué características deben poseer los procesos de desarrollo de software?

- Flexibles
- Tener aceptación
- Deben atacar los problemas esenciales del software

¿Qué sucede cuando no se entiende que es desarrollar software?

- Se crean mitos
- Se cree en "balas de plata"
- No hay gerenciamiento

Una cultura incluye un conjunto de **valores**, **objetivos y principios** compartidos que guían el **comportamiento**, las **actividades** y **prioridades** de un grupo de personas trabajando hacia un objetivo común.

Cualidades externas e internas del software

Ambas son de interés para los desarrolladores, mientras que los usuarios sólo son conscientes de las primeras.

Correctitud: Un software es funcionalmente correcto si se comporta de acuerdo a la especificación de las funciones (especificación de requerimientos funcionales) que debería proveer. No toma en consideración el que la especificación en sí misma puede ser incorrecta.

Confiabilidad: Informalmente el software es confiable si el usuario puede depender de él. Formalmente la confiabilidad se define como: la probabilidad de que el software opere como es esperado en un intervalo de tiempo especificado.

Reusabilidad: Puede integrarse como componente dentro de otras aplicaciones.

Eficiencia o Performance: Un sistema de software es eficiente si utiliza los recursos computacionales en forma económica. La eficiencia afecta la usabilidad del software, por ejemplo, si es muy lento reduce la productividad de los usuarios, si usa demasiado espacio de disco puede ser muy caro de ejecutar, etc.

Robustez: Un programa es robusto si se comporta en forma razonable aún en circunstancias que no fueron anticipadas en la especificación de requerimientos; por ejemplo cuando encuentra datos de entrada incorrectos o algún malfuncionamiento del hardware como rotura de disco.

Amigabilidad o usabilidad: Un sistema de software es amigable si un usuario humano lo encuentra fácil de utilizar. Esta definición refleja la naturaleza subjetiva de la amigabilidad.

Verificabilidad: Un sistema de software es verificable si sus propiedades pueden ser verificadas fácilmente. El diseño modular, prácticas de codificación disciplinadas, y la utilización de lenguajes de programación adecuados contribuyen a la verificabilidad de un sistema. El tester puede ver fácilmente si un requerimiento se cumple o no.

Mantenibilidad: Se puede ver como dos cualidades separadas:

- Reparabilidad: Un sistema de software es reparable si permite la corrección de sus defectos con una carga limitada de trabajo. Un producto de software consistente en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico.
- Evolucionabilidad: Un sistema es evolucionable si acepta cambios que le permitan satisfacer nuevos requerimientos. La documentación del software debe encontrarse actualizada, haciendo que los cambios futuros sean más fáciles de aplicar. También, anticiparse al cambio favorece esta cualidad.

Comprensibilidad: Algunos sistemas de software son más fáciles de comprender que otros, algunas tareas son inherentemente más complejas que otras. Dadas dos tareas con dificultad similar, se pueden seguir ciertas guías para producir diseños y escribir programas más comprensibles. Ayuda a lograr muchas de las otras cualidades como evolucionabilidad y verificabilidad.

Portabilidad: Puede ser usado, sin necesidad de modificarlo, en diferentes plataformas.

Interoperabilidad: Se refiere a la habilidad de un sistema de coexistir y cooperar con otros sistemas.

Productividad: La productividad es una cualidad del proceso de producción de software, mide la eficiencia del proceso y, es la cualidad de performance aplicada al proceso. Un proceso eficiente resulta en una entrega más rápida del producto.

Oportunidad: La oportunidad es una cualidad del proceso que se refiere a la habilidad de entregar un producto a tiempo.

Visibilidad: Un proceso de desarrollo de software es visible si todos sus pasos y su estado actual son claramente documentados. Otros términos utilizados son transparencia y apertura. La idea es que los pasos y el estado del proyecto están disponibles y fácilmente accesibles para ser examinados externamente.

Corrección funcional: Un software es funcionalmente correcto si se comporta de acuerdo a las especificaciones de las funciones que debe provee. Se asume que existe una especificación y busca una equivalencia entre software y su especificación.

Montenegro, Micaela Soledad

- En general:
    - Las especificaciones de sistemas medianos-grandes no son completas
    - No se logra verificar todo
    - ¿Cómo garantizamos la corrección de la verificación?
- Y nunca garantiza que se implementen los requerimientos deseados.

2. PROCESO DE DESARROLLO DE SOFTWARE

DEFINICIÓN DE PROCESO Y PROCESO SOFTWARE

Definición de Proceso

- "Un proceso define quién está haciendo qué, cuándo y cómo alcanzar un determinado objetivo" Jacobson. Booch. Rumbaugh
- "Los procesos son unas series de pasos que involucran actividades, restricciones y recursos que producen una determinada salida esperada" Pfleeger
- "Un proceso es una serie de eventos o fases que tienen lugar con el tiempo y tienen un propósito o resultado identificado" Olson

El proceso por dentro puede estar formado de subprocesos, y prescribe todas las actividades principales. Dichas actividades son organizadas en secuencia, y cada una de ella tiene un criterio de entrada y de salida.

El proceso usa recursos y produce productos intermedios y finales, y tiene un conjunto de principios guías. Además existen restricciones o controles que pueden aplicarse a una actividad, recurso, o producto.

La importancia de la implementación de un proceso es que permite imponer consistencia, capturar experiencias y examinar, comprender, controlar y mejorar las actividades.

Definición de Proceso de Software

Es una serie actividades relacionadas que conducen a la creación de un producto de software. Estas actividades pueden consistir en el desarrollo de software desde cero, o en el desarrollo de nuevo software para la ampliación o modificación de software existente.

Los procesos de software son complejos, y como todo proceso intelectual y creativo, dependen de las personas que toman decisiones y juicios.

El proceso de desarrollo de software

Un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar y mantener software y los productos asociados.

El proceso de software debe incluir cuatro actividades que son fundamentales para la Ingeniería de Software:

- Especificación de Software: Definir tanto la funcionalidad del software como las restricciones de su operación y el desarrollo del sistema. Está conformado por cuatro actividades: Estudio de factibilidad, Obtención y análisis de requerimientos, Especificación de requerimientos y validación de requerimientos.
- Diseño e implementación del software: Desarrollar el software para cumplir con las especificaciones. Esta actividad toma como entradas la plataforma de información (entorno donde se ejecutará el sistema), la especificación de requerimientos y la descripción de datos. Entre las actividades de diseño encontramos el diseño arquitectónico, el diseño de interfaz, el diseño de componentes y el diseño de base de datos, y todas estas actividades conducen a un conjunto de salidas: arquitectura del sistema, especificación de interfaz, especificación de base de datos y especificación de componentes.
- Validación del software: Validar el software para asegurarse que cumple con lo que el cliente quiere. Muestra que un sistema cumple tanto con sus especificaciones como con las expectativas del cliente. Aquí se realizan una serie de pruebas: pruebas de desarrollo, pruebas de sistema y pruebas de aceptación.

- Evaluación del software: El software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

Estas actividades son complejas en sí mismas e incluyen subactividades como por ej. Diseño arquitectónico. Además también existen otras actividades que son de soporte como por ej. La documentación y la configuración del software.

Las descripciones de los procesos deben incluir:

- Productos: resultados de una actividad del proceso.
- Roles: responsabilidades de la gente interviniente en el proceso.
- Precondiciones y Postcondiciones: declaraciones válidas antes y después de que se realice una actividad del proceso o se cree un producto.

Características principales: Visibilidad, Predecible, Aceptación, Fiabilidad, Robustez, Mantenibilidad y Rapidez.

Los procesos de software pueden mejorarse con la estandarización de los procesos donde se reduce la diversidad en los procesos de software de una organización, lo que conduce a mejorar la comunicación, a reducir el tiempo de capacitación, y a que el soporte de procesos automatizados sea más económico.

Que se espera del proceso de desarrollo de software

- Project Manager: Desarrollar un conjunto consistente y gerenciable de documentos del sistema
- Usuario: Disponer de un sistema que satisfaga sus necesidades con respecto a calidad, funcionalidad, costos, etc.
- Ingeniero software: Contar con un esquema en el que ejecutar sus actividades técnicas
- Empresa: Mejorar la habilidad organizacional para desarrollar sistemas de calidad productivamente y rentablemente

Ciclo de vida del software

Los procesos del ciclo de vida del software según la ISO/IEC 12207 son:

- **Procesos del ciclo de vida primario:** Conducen las principales funciones.
  - Proceso de adquisición
    - Tareas del que contractualmente adquiere un producto de software o un servicio
    - Actividades: Definición de la necesidad, pedido de propuesta (RFP), selección de un proveedor y gerenciamiento del proceso hasta la aceptación del sistema
  - Proceso de provisión
    - El servicio puede ser el desarrollo de un producto de software, un sistema conteniendo software, la operación o mantenimiento de un sistema de software
    - Actividades: preparar una propuesta a un comprador, acordar un contrato, identificar los procedimientos y recursos necesarios, desarrollar, ofrecer el servicio
  - Proceso de desarrollo

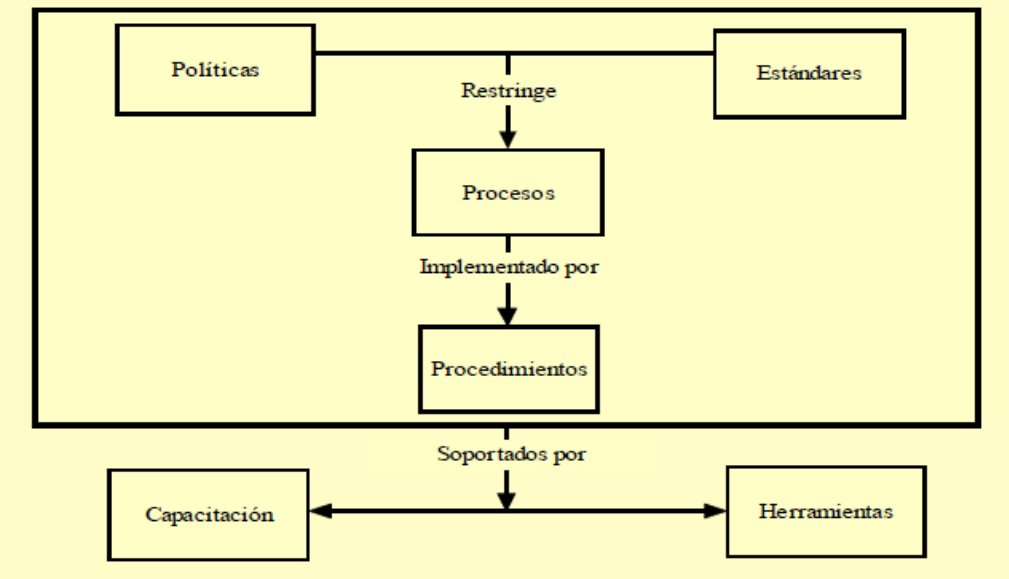
        - Actividades y tareas del desarrollador del sistema y el software
        - Desarrollar software: software nuevo o modificar el existente
        - Actividades: Tareas específicas del desarrollo de software
    - Proceso de operación: Abarca la operación del software y el soporte operacional a los usuarios
        - Actividades: implementación, testeo operativo, operación del sistema y soporte al usuario
    - Proceso de mantenimiento
        - Causa: error, necesidad de mejora, adaptación.
        - Alcance: modificar el código y la documentación asociada, finaliza cuando el sistema se retira del uso
        - Actividades: análisis del problema y modificación, aceptación, migración, retiro del software
- **Procesos del ciclo de vida de soporte:** Dan soporte a otros procesos para realizar una función especial
    - Proceso de documentación: Objetivo: registrar la información producida por los procesos del ciclo de vida.
    - Proceso de gestión de configuración: Identifica y define el baseline de los ítems de software en el sistema, para controlar modificaciones y versiones de los ítems.
    - Proceso de aseguramiento de la calidad: Provee un framework para asegurar (el adquirente o el cliente) el cumplimiento de los productos o servicios con sus requerimientos contractuales y el acatamiento de los planes establecidos.
    - Proceso de verificación: Determina si los requerimientos para un sistema son completos y correctos y que el output de una actividad satisface los requerimientos y condiciones impuestos en actividades previas.
    - Proceso de validación: Determina si el sistema final cumple con el objetivo inicial. No reemplaza otras evaluaciones, solo las complementa.
    - Proceso de revisión conjunta: Provee un framework para la interacción entre el "revisado" el revisor.
    - Proceso de auditoría: El auditor evalúa los productos y actividades del auditado con énfasis en el cumplimiento de los requerimientos y planes.
    - Proceso de resolución de problemas: Se resuelven los problemas tomando acciones correctivas en la medida que se detectan.
- **Procesos del ciclo de vida organizacional:** Establecen, controlan y mejoran los procesos del ciclo de vida
    - Procesos gerencial: Define las actividades y tareas del manager
    - Proceso de infraestructura: La infraestructura puede incluir hardware, software, standards, herramientas, técnicas y medios
        - Objetivo: definir las actividades para establecer y mantener la infraestructura necesaria
    - Proceso de mejora
        - Objetivos: mejorar el proceso organizacional
        - Actividades: evaluar, medir, controlar y mejorar el proceso de ciclo de vida
    - Proceso de entrenamiento: Actividades: identificar y hacer un plan para incorporar o desarrollar recursos de personal.

Proceso de desarrollo orientado a proyectos pequeños: Los factores que inciden en la categorización son el tamaño de la organización, la complejidad del proyecto y el n° de interacciones. Su importancia está dada por la atención en la actividad favorita, atrapados en la rutina y no hay pautas claras.

Patrones del proceso

Un framework de proceso es un mecanismo para definir un proceso de desarrollo. Es un repositorio de fragmento de método (tareas, roles, artefactos y guías) y elementos del proceso que pueden adaptarse a necesidades individuales para definir un proceso de desarrollo. Un repositorio con información de buenas prácticas, libros, publicaciones, estandarizaciones, etc.

El Software Process Framework es una herramienta de definición de procesos:



Su template consiste de: Rol, Capacitación, Actividades, Criterio de entrada, Inputs, Criterio de salida, Outputs, Revisiones y auditoria, Administración y control de los productos de trabajo, Mediciones, Procedimientos documentados y Herramientas.

Código:	Nombre:	

Objetivos:

Justificación:

Actividades:			
Paso	Nombre Actividad	Objetivo	Rol

Recursos:	
Humanos:	Tecnológicos:

Técnicas:

Input:			
Nombre	Origen	Criterio de Entrada: Estado o Condición	Tipo

Outout			
Nombre	Destino	Criterio de Salida: Estado o Condición	Tipo

Definiciones:

Restricciones:

Variables a medir:	
Nombre de la variable	

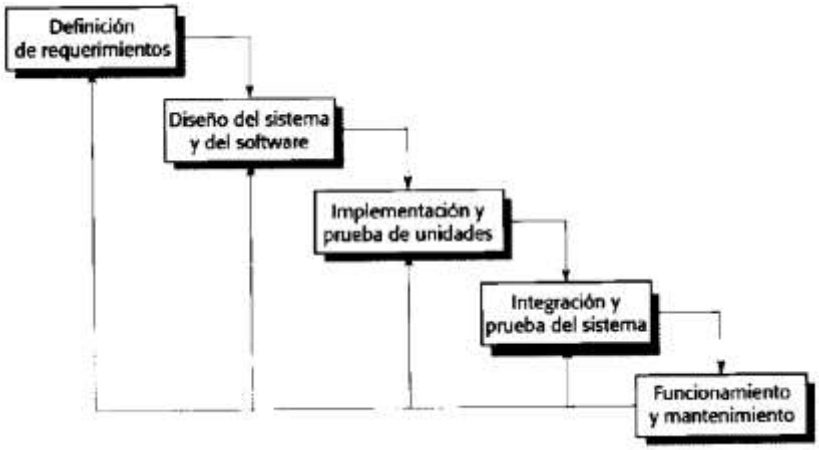
Observaciones:

MODELOS DE PROCESO SOFTWARE

Un modelo de proceso de software es una representación simplificada de este proceso. Tales modelos no son descripciones definitivas de los procesos de software sino más bien, son abstracciones del proceso que se utilizan para explicar los diferentes enfoques del desarrollo de software.

Estos modelos ayudan a una comprensión común, a encontrar inconsistencias, redundancias y omisiones. El modelo debería reflejar los objetivos de desarrollo permitiendo evaluar actividades candidatas para atacar estos objetivos. Los modelos permiten la adecuación a cada situación especial y facilitan las mediciones.

Modelo de Cascada



Éste modelo toma las actividades fundamentales del proceso (especificación, desarrollo, validación y evolución) y luego los representa como fases separadas del proceso.

Montenegro, Micaela Soledad

Las principales etapas del modelo en cascada son:

- Análisis y definición de requerimientos: Los servicios, restricciones y las metas del sistema se establecen mediante consulta a los usuarios del mismo. Luego se definen con detalle y sirven como una especificación del sistema.
- Diseño del sistema y del software: en el diseño de sistemas se asignan los requerimientos, al establecer una arquitectura de sistema global y en el diseño de software se identifican y describen las abstracciones fundamentales del sistema de software y sus relaciones.
- Integración y prueba de unidad: El diseño de software se realiza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.
- Integración y Prueba de Sistema: Las unidades de programas o los programas se integran y prueban como un sistema completo para asegurarse que se cumplan con los requerimientos de software.
- Operación y Mantenimiento: El sistema se instala y se pone en práctica. El mantenimiento incluye corregir errores, mejorar la implementación, incrementar los servicios conforme se descubren nuevos requerimientos.

El resultado de cada fase consiste en uno o más documentos que se autorizaron. La siguiente fase no debe comenzar sino hasta que termine la fase previa.

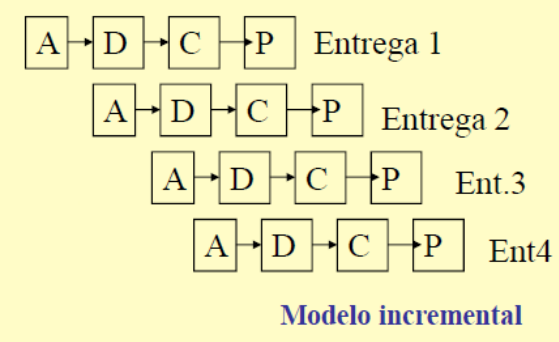
El proceso de software no es un simple modelo lineal, sino que implica retroalimentación de una fase a otra. Entonces es posible que los documentos generados deban modificarse para reflejar los cambios que se realizan. Además el sistema debe evolucionar para mantenerse útil, hacer tales cambios puede implicar la repetición de etapas anteriores del proceso.

En cada fase se produce documentación. Esto hace que el proceso sea visible, de modo que los administradores monitoricen el progreso contra el plan de desarrollo.

Desventajas:

- Partición inflexible del proyecto en distintas etapas: tienen que establecerse compromisos en una etapa temprana del proceso, lo que dificulta responder a los requerimientos cambiantes del cliente.
- Debe usarse cuando los requerimientos se entiendan bien y sea improbable el cambio radical durante el desarrollo del sistema.

<u>Desarrollo Incremental o Mejora Iterativa</u>



Se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado. Las actividades de especificación, desarrollo y validación están entrelazadas, con rápida retroalimentación a través de las actividades.

Se avanza con una serie de pasos hacia una solución y se retrocede cuando se detecta que se cometieron errores. Al desarrollar el software de manera incremental, resulta más barato y fácil realizar cambios en el software conforme éste se diseña.

Cada incremento o versión del sistema incorpora algunas de las funciones que necesita el cliente. El cliente puede evaluar el desarrollo del sistema en una etapa relativamente temprana, para constatar si se entrega lo que se requiere.

Beneficios:

- Se reduce el costo de adaptar los requerimientos cambiantes del cliente.
- Es más sencillo obtener retroalimentación del cliente sobre el trabajo de desarrollo que se realizó.
- Es más rápida la entrega e implementación de software útil al cliente, aun si no se ha incluido toda la funcionalidad.

Problemas:

- El proceso no es visible. Si los sistemas se desarrollan rápidamente resulta poco efectivo en término de costos producir documentos que reflejen cada versión del sistema.
- La estructura del sistema tiende a degradarse conforme se implementan nuevos cambios, por lo que la incorporación de cambios de vuelve más difícil y costosa.

Construcción de Prototipos

Un prototipo es una versión inicial de un sistema de software que se usa para demostrará conceptos, tratar opciones de diseño y encontrar más acerca de un problema y sus posibles soluciones.

Un prototipo de software se usa en un proceso de desarrollo de software para contribuir a anticipar los cambios que se requieran:

- En el proceso de ingeniería de requerimientos, un prototipo ayuda con la selección y validación de los requerimientos del sistema.
- En el proceso de diseño de sistemas, un prototipo sirve para buscar soluciones específicas del software y apoyar el diseño de interfaces de usuario.

Los prototipos del sistema permiten a los usuarios a ver qué tan bien el sistema apoya su trabajo. Pueden obtener nuevas ideas para los requerimientos y descubrir áreas de fortalezas y debilidades en el software. Permite revelar errores y omisiones en los requerimientos propuestos.

Los objetivos de la creación de prototipos deben ser más explícitos desde el inicio del proceso.

La siguiente etapa del proceso consiste en decidir qué poner y qué dejar afuera del sistema de prototipo.

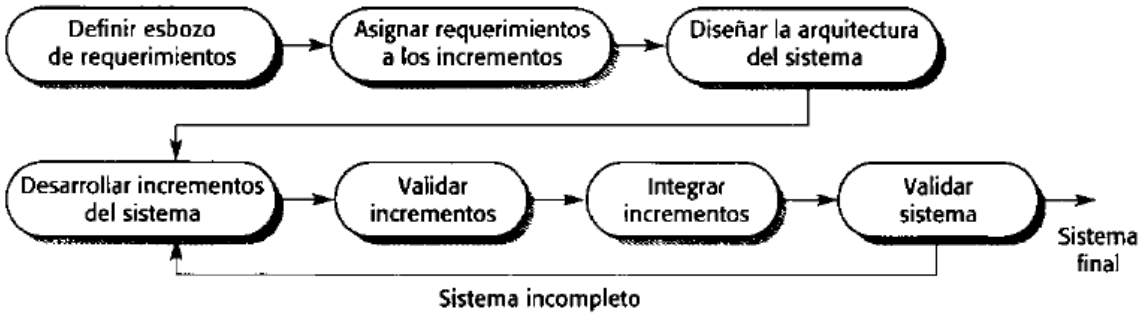
La etapa final del proceso es la evaluación del prototipo. Hay que tomar provisiones durante esta etapa para la capacitación del usuario y usar los objetivos del prototipo para derivar un plan de evaluación.

En ocasiones los desarrolladores están presionados para entregar prototipos desechables, pero esto no es aconsejable debido a:

- Puede ser imposible corregir el prototipo para cubrir requerimientos no funcionales.

- El cambio rápido durante el desarrollo significa que el producto no está documentado, lo que no es bueno para el mantenimiento a largo plazo.
- Los cambios realizados durante el desarrollo de prototipos degrada la estructura del sistema, lo que hará que sea difícil y costoso de mantener.
- Durante el desarrollo del prototipo se hacen más flexibles los estándares de calidad de la organización.

Entrega en Etapas o Entrega Incremental



Es un enfoque al desarrollo de software donde algunos de los incrementos diseñados se entregan al cliente y se implementa para usarse en un entorno operacional. En un proceso de entrega incremental, los clientes identifican los servicios que proporciona el sistema, identifican cuáles son más importantes y cuáles son menos significativos. Entonces se define un número de incrementos de entrega y cada incremento proporciona un subconjunto de la funcionalidad del sistema. Una vez completado y entregado el incremento, los clientes lo ponen en servicio.

A medida que se completan nuevos requerimientos, se integran con los incrementos existentes, de modo que con cada incremento entregado mejore la funcionalidad del sistema.

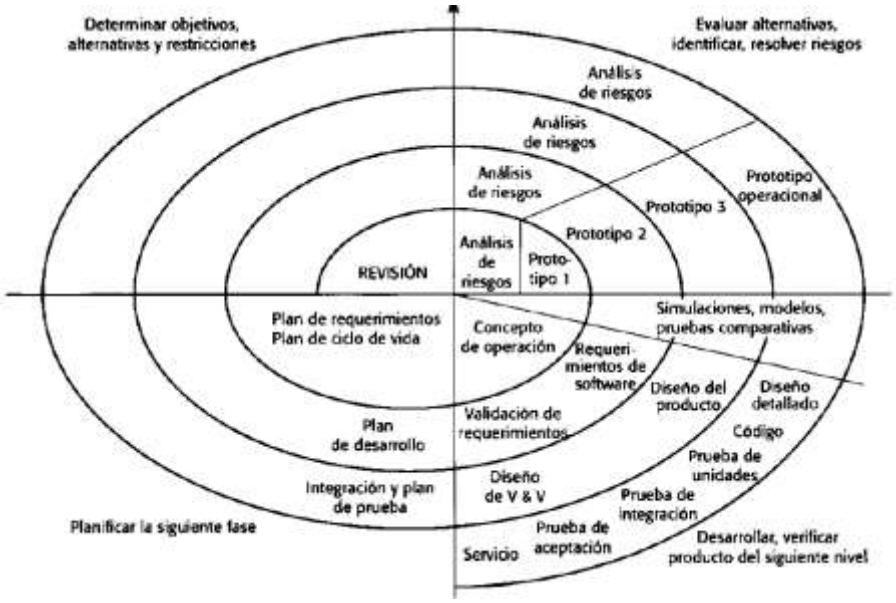
Ventajas:

- Los clientes pueden usar los primeros incrementos como prototipos y adquirir experiencia que informe sobre sus requerimientos
- Los clientes deben esperar hasta la entrega completa del sistema antes de ganar el valor del mismo, sin embargo ya pueden comenzar a utilizarlo en el primer incremento.
- Relativamente sencillo incorporar cambios al sistema
- Los servicios de mayor prioridad reciben pruebas mayores

Problemas:

- Dado que los requerimientos no están definidos en detalle, resulta difícil identificar recursos comunes que necesiten todos los incrementos
- El desarrollo iterativo resulta complicado cuando se diseña un sistema de reemplazo, ya que los clientes requieren toda la funcionalidad del sistema antiguo
- En un enfoque incremental no hay especificación completa del sistema, sino hasta que se define el sistema final. Esto requiere de una nueva forma de contrato que los grandes clientes encontrarían difícil de adoptar

Modelo de Espiral



El proceso se representa como un espiral, donde cada ciclo del espiral representa una fase del proceso de software. El modelo en espiral combina el evitar el cambio con la tolerancia al cambio. Lo anterior supone que los cambios son resultado de riesgos del proyecto e incluye actividades de gestión de riesgos explícitas para reducirlos.

Cada ciclo en la espiral se divide en cuatro sectores:

- Establecimiento de objetivos: se definen objetivos específicos para dicha fase del proyecto. Se identifican restricciones y se traza un plan de gestión detallado. Se identifican los riesgos del proyecto.
- Valoración y reducción del riesgo: en cada uno de los riesgos identificado, se realiza un análisis minucioso. Se dan acciones para reducir el riesgo.
- Desarrollo y validación: Se elige un modelo de desarrollo para el sistema.
- Planeación: El proyecto se revisa y se toma la decisión si hay que continuar con otro ciclo del espiral, si es sí, se trazan los planes para la siguiente fase del proyecto.

La principal característica de este modelo es su reconocimiento explícito de riesgos.

<u>Modelo Basado en Reutilización</u>

Se apoya en una gran base de componentes de software reutilizable y en la integración de marcos para la composición de dichos componentes.

La etapa inicial de especificación de requerimientos, y la etapa final de validación se comparan con otros procesos de software, mientras que las demás etapas intermedias son diferentes:

- Análisis de componentes: Dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación que es parte de la funcionalidad requerida.
- Modificación de requerimientos: Se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles.

- Diseño de sistema con reutilización: Se diseña el marco conceptual del sistema o se reutiliza un marco conceptual existente.
- Desarrollo e integración: Se diseña el software que no puede procurarse de manera externa y se integran los componentes y los sistemas COTS para crear el nuevo sistema.

Existen tres tipos de componentes de software que pueden usarse en un proceso orientado a la reutilización:

- Servicios web que se desarrollan para atender servicios estándares
- Colecciones de objetos que se desarrollan como un paquete para su integración con un marco de componentes
- Sistemas de software independientes que se configuran para usar en un entorno en particular

Ventajas: Reduce la cantidad de software a desarrollar y por ende disminuye costos y riesgos y conduce a entregas más rápidas del software.

Problemas: Es inevitable los compromisos con requerimientos y esto conduce hacia un sistema que no cubra necesidades reales de usuarios, además se pierde algo de control sobre la evolución del sistema.

DESARROLLO ÁGIL

Los métodos ágiles se apoyan en el enfoque incremental para la especificación, el desarrollo y la entrega del software. Tiene la intención de entregar con prontitud el software operativo al cliente, quienes entonces propondrán nuevos requerimientos y variados para incluirlos en posteriores iteraciones del sistema.

Clásico	Ágil
Requisitos detallados	Visión general del producto
Planificación estricta	Adaptación a la situación
Requerimientos no cambiantes	Evolución constante
Seguimiento y Control	Autogestión
División y especialización	Equipo multi- disciplinar

Las Metodologías Ágiles valoran:

- Al individuo y las interacciones en el equipo de desarrollo más que a las actividades y las herramientas
- La colaboración con el cliente más que la negociación de un contrato
- Responder a los cambios más que seguir estrictamente una planificación

Metodologías Ágiles

- Basadas en heurísticas provenientes de prácticas de producción de código especialmente preparados para cambios durante el proyecto
- Impuestas internamente (por el equipo)
- No existe contrato tradicional o al menos es bastante flexible
- El cliente interactúa con el equipo de desarrollo
- Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio
- Pocos artefactos
- Pocos roles

Metodologías tradicionales

- Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
- Cierta resistencia a los cambios
- Impuestas externamente
- Existe un contrato prefijado
- El cliente es parte del equipo de desarrollo mediante reuniones
- Grupos grandes y posiblemente distribuidos
- Más artefactos
- Más roles

Los métodos ágiles comparten una serie de principios:

1. La mayor prioridad es satisfacer al cliente con entregas continuas y tempranas de software valioso.
2. Los cambios en los requerimientos son bienvenidos, aun en etapas avanzadas del desarrollo.
3. Entregas de software funcionando, frecuentemente, desde un par de semanas a un par de meses, con preferencia a escalas de tiempo menores.
4. La gente de negocio y los desarrolladores trabajan juntos diariamente a lo largo del proyecto.
5. La construcción de proyectos con individuos motivados. Hay que ofrecerles el ambiente y soporte que ellos necesitan y confiar en que harán el trabajo.
6. El método más eficiente y efectivo de comunicación de información a y con el equipo de desarrollo es la conversación cara a cara.
7. El software funcionando es la principal medida de progreso.
8. Los procesos ágiles promueven desarrollo sostenible.
9. Sponsors, desarrolladores y usuarios deberían poder mantener pasos constantes, indefinidamente.
10. La atención continua a la excelencia tecnológica y al buen diseño mejoran la agilidad.
11. La simplicidad, el arte de maximizar la cantidad de trabajo no hecho, es esencial.
12. Las mejores arquitecturas, requerimientos y diseños emergen de equipos que se organizan ellos mismos.
13. En intervalos regulares, el equipo refleja cómo puede volverse más eficiente, entonces modifican y ajustan su comportamiento de manera acorde a esto.

Los métodos ágiles son un subconjunto de los métodos iterativos.

El modelado ágil no es un proceso completo o un método ágil, es un conjunto de principios y prácticas para modelado y análisis de requerimientos, que complementa a la mayoría de los métodos de desarrollo iterativos e incrementales.

Problemas: necesita del involucramiento del cliente, los miembros del equipo deben contar con la personalidad adecuada para la participación intensa, priorizar los cambios es difícil, mantener la simplicidad requiere trabajo adicional, difícil de elaborar contratos para este tipo de desarrollo, dificultad de mantener el involucramiento del cliente luego de entregar el software y de mantener la continuidad del equipo de desarrollo una vez entregado el sistema.

Algunos métodos ágiles: SCRUM, XP, Métodos Cystal, Desarrollo de Software Adaptativos (ASD), Modelo Dinámico de Entrega de Soluciones (DSDM) y Desarrollo Conducido por Características (FDD).

Scrum

Es una metodología humana para el desarrollo de software donde su enfoque está en la administración iterativa del desarrollo.

Características de los campos scrum:

- Incertidumbre consustancial
- Equipos auto-organizados
- Fases de desarrollo solapadas
- Control sutil: El término control sutil se refiere a la creación de un ecosistema que potencia y desarrolla el autocontrol entre iguales, como consecuencia de la responsabilidad y del gusto por el trabajo realizado. Algunas acciones para generar este ecosistema son: Selección de las personas adecuadas para el proyecto, Análisis de los cambios en la dinámica del grupo para incorporar o retirar a miembros si resulta necesario, Creación de un espacio de trabajo abierto, Animar a los ingenieros a mezclarse con el mundo real de las necesidades de los clientes, Sistemas de evaluación y reconocimiento en el rendimiento del equipo, Tolerancia y previsión con los errores; considerando que son un medio de aprendizaje y que el miedo al error merma la creatividad y la espontaneidad
- Difusión y transferencia del conocimiento

Scrum es un método adaptativo de gestión de proyectos que se basa en los principios ágiles:

- Colaboración estrecha con el cliente.
- Predisposición y respuesta al cambio
- Prefiere el conocimiento tácito de las personas al explícito de los procesos
- Desarrollo incremental con entregas funcionales frecuentes
- Comunicación verbal directa entre los implicados en el proyecto
- Motivación y responsabilidad de los equipos por la auto-gestión, auto-organización y compromiso.
- Simplicidad. Supresión de artefactos innecesarios en la gestión del proyecto.

Existen tres fases con Scrum:

- Planeación del bosquejo: donde se establecen los objetivos generales del proyecto y el diseño de la arquitectura de software.
- A esto le sigue una serie de ciclos sprint, donde cada ciclo desarrolla un incremento del sistema.
- Finalmente, la fase de cierre del proyecto que concluye el mismo, completa la documentación requerida, y valora las lecciones aprendidas en el proyecto.

La característica innovadora de Scrum es su fase central, los ciclos sprint. Un sprint de Scrum es una unidad de planeación en la que se valora el trabajo que se va a realizar, se seleccionan las particularidades por desarrollar y se implementa el software. Al final del sprint la funcionalidad completa se entrega a los participantes. Las características clave de este proceso son:

- Los sprint tienen longitud fija (de 2 a 4 semanas)
- El punto de partida para la planeación es la cartera de producto. Durante la fase de valoración del sprint esto se revisa y se asignan prioridades y riesgos. El cliente interviene en este proceso y al comienzo de cada sprint puede introducir nuevos requerimientos o tareas.
- La fase de selección selecciona las característica y la funcionalidad a desarrollar durante el sprint.
- El equipo se organiza para desarrollar el software. Se realizan reuniones diarias breves y enfocadas con todos los miembros los cuales comparten información, describen sus avances desde la última reunión, los problemas que surgieron y los planes del día siguiente. Durante esta etapa todas las comunicaciones se canalizas a través de un Maestro de Scrum, cuyo papel es proteger al equipo de desarrollo de distracciones externas, y es quien ordena las reuniones diarias, rastrea el atraso del trabajo a realizar, registra las decisiones, mide el progreso del atraso y se comunica con los clientes y administradores fuera del equipo.
- Al final del sprint el trabajo hecho se revisa y se presenta a los participantes. Luego comienza el siguiente ciclo de sprint.

Ventajas:

- El producto se desglosa en un conjunto de piezas manejables y comprensibles
- Los requerimientos inestables no retrasan el progreso
- Los clientes observan la entrega a tiempo de los incrementos y obtiene retroalimentación sobre cómo funciona el producto
- Todo el equipo tiene conocimiento de todo, y se mejora la comunicación entre el mismo
- Se establece confianza entre clientes y desarrolladores, a l vez que se crea una cultura positiva donde todos esperan el triunfo del proyecto.

Valores del Scrum: Integridad, transparencia, compromiso y responsabilidad total.

Equipo: Se debe promover al mantenimiento de equipos durables, trabajo en parejas, auto-organizado, recompensa para equipo, estrecha comunicación, co-localizado, cross-funcional y jefe facilitador.

¿Qué es un Self-organizing Team?

Equipo Auto-organizado

Los miembros del equipo:

- Acuerdan el reparto del trabajo, en conjunto y/o cada miembro se auto-asigna trabajo en la medida que se quede disponible. **Proactividad**.
- Acuerdan cómo realizar el trabajo. Toman las decisiones técnicas necesarias.
- Comparten un rol genérico, p.e. "Desarrollador". No existe el rol Jefe, o si existe es más bien un "facilitador", el cual no interviene en la asignación de trabajo ni en cómo debe hacerse el trabajo.

Roles del Scrum: Scrum diferencia claramente entre estos dos grupos para garantizar que quienes tienen la responsabilidad tienen también la autoridad necesaria para poder lograr el éxito, y que quienes no tienen la responsabilidad no producen interferencias innecesarias:

- COMPROMETIDOS EN EL PROYECTO: Dueño del producto, Equipo, Scrum Manager
- IMPLICADOS EN EL PROYECTO: Marketing, Comercial, etc.

Los roles en Scrum son:

- Producto Owner - Propietario del producto: Representa a todos los interesados en el producto final. Sus áreas de responsabilidad son: Financiación del proyecto, Definir la funcionalidad del producto - Lista inicial de requerimientos (Product Backlog), Decidir las fechas de liberación y el contenido, Responsable del ROI, Prioriza las funcionalidades de acuerdo a su valor de mercado, Ajusta las prioridades y funcionalidades después de cada iteración y Acepta o rechaza los resultados del trabajo.
- Equipo. Responsable de transformar la pila del sprint (Sprint Backlog) en un incremento de la funcionalidad del software: Normalmente entre 5-10 personas, Multidisciplinario, Programadores, testers, diseñadores gráficos, Los miembros deberían ser full-time, Algunas excepciones comunes: DBAs, Administradores, Los equipos se auto gestionan y organizan y Los miembros cambian de equipo al finalizar los sprints.
- Scrum Manager Responsable del proceso Scrum: Garantía de cumplimiento de roles y responsabilidad, Formación y entrenamiento del proceso, Responsable del cumplimiento de las prácticas y valores de Scrum, Se encarga que el equipo funcione completamente y sea, Facilita cooperación entre todos los roles, Remueve impedimentos, Sirve de interface desde y hacia el equipo y Protege al equipo de interferencias externas.

Sprint: Periodo fijo de tiempo durante el cual el equipo desarrolla un incremento de funcionalidad. Constituye el núcleo de Scrum, que divide de esta forma el desarrollo de un proyecto en un conjunto de pequeñas "carreras".

No se aceptan cambios a los requerimientos acordados, duración máxima: 30 días y el producto es diseñado, codificado y testeado durante el Sprint.

Sólo es posible cambiar el curso de un sprint, abortándolo, y sólo lo puede hacer el Scrum Master si decide que no es viable por alguna de las razones siguientes: La tecnología acordada no funciona, Las circunstancias del negocio han cambiado o El equipo ha tenido interferencias.

Componentes:

- Pila de Producto: Listado con los requerimientos del sistema. Características: Prioridad, Nunca llega a ser una lista completa y definitiva, El empleado para planificar el proyecto es sólo una estimación inicial de requerimientos, Es un documento dinámico que incorpora constantemente las necesidades del sistema, Se mantiene durante todo el ciclo de vida, Es responsabilidad del dueño del producto. Partes: Contenido, Priorización y Disponibilidad.
- Pila de sprint: Trabajo o tareas determinadas por el equipo para realizar en un sprint y lograr al final del mismo un incremento de la funcionalidad. Se recomienda que las tareas reflejadas tengan una duración comprendida entre las 4 y las 16 horas de trabajo. Las de mayor duración deben intentar descomponerse en sub-tareas de ese rango de tiempo.
- Incremento

Formas de estimar: Expertos, Analogía, Dividir, Planning Poker (consiste en Cartas con 1, 2, 3, 5, 8, 20, 40, 100, Aprendizaje y Timeboxed)

User Story: Unidad más pequeña de incremento del sistema. Unidad de estimación y control, Incluye los objetivos y motivaciones del usuario.

Sala de desarrollo:

Reuniones:

- Daily Scrum: Reunión del equipo con duración máxima de 15 minutos.
  - Todos los días en el mismo sitio y a la misma hora.
  - Se recomienda que sea la primera actividad del día.
  - Deben acudir todos los miembros del equipo.
  - Moderada por el Scrum Manager,
  - Sólo habla la persona que informa de su trabajo, el resto escucha y no hay lugar para otras conversaciones.
  - Las gallinas no pueden intervenir ni distraer, y el Scrum Master puede limitar el número de gallinas asistentes si lo considera oportuno.
  - El Scrum Master está para facilitar y promover estas prácticas
  - Las 3 preguntas (qué hice desde el último scrum, qué voy a hacer, que bloqueos tengo) no son para dar el estado al ScrumMaster, son compromisos frente a los pares.

Sprint Review Meeting (Qué construímos)

- **Objetivo:** Presentar al Product Owner y demás involucrados del proyecto el trabajo realizado (incremento del producto) durante el Sprint
- **Participan:** Equipo, Scrum Master, Product Owner, todas las personas involucradas en el proyecto
- **Reglas a seguir:**
  - El Team no invierte más de una hora para preparar el Sprint review
  - Las funcionalidades no finalizadas completamente no se presentan
  - Los miembros del equipo presentan las funcionalidades
  - Las demostraciones se realizan en las workstations de los miembros del equipo
  - Al finalizar la reunión se pide opiniones a los participantes, los cuales pueden sugerir cambios y mejoras

- **Al finalizar:**
  - Se actualiza y vuelve a priorizar el Product Backlog
  - El Scrum Master anuncia el lugar y la fecha de la próxima revisión de Sprint

El Product Owner decide si la funcionalidad presentada cumple con los objetivos del Sprint

Sprint Retrospective (Cómo contruímos)

- **Objetivo**: identificar que cosas se pueden cambiar para hacer el trabajo más agradable y productivo en las próximas iteraciones.
- Se realiza al finalizar el Sprint
- **Participantes**: Team, Scrum Master, Product Owner (opcional).
- **Dos preguntas (todos responden):**
  - Qué cosas hicimos bien?
  - Qué cosas podemos mejorar?
- Todo aquello que afecte como el equipo construye software se debe debatir
- Permite al equipo evolucionar continuamente mejorando durante el proyecto.

Organizaciones que han utilizado Scrum: IBM, Yahoo, Google, Electronic Arts, Lockheed Martin, Philips, Siemens, Nokia, Microsoft, BBC, John Deere, Time Warner.

3. INTRODUCCIÓN A LA GESTIÓN DE PROYECTOS

CONCEPTOS:

Atributo crítico: es una calidad o recurso que puede causar que el sistema colapse si supera determinados límites (peor caso aceptable).

Esfuerzo: tiempo necesario para realizar una actividad medida en horas (días, meses,...) por persona. Tiempo que tardaría una sola persona en realizar esa actividad.

Duración: intervalo de tiempo necesario para realizar una actividad, desde que empieza hasta que termina.

Año-hombre: es el trabajo que puede realizar una persona en un año. Es una unidad de medida para el esfuerzo.

PROYECTO

Es una secuencia única, compleja y conectada, de actividades que tienen un objetivo y propósito y que deben ser completadas en un tiempo específico, en presupuesto, y de acuerdo a una especificación.

	Tipo de trabajo ejecutado por las organizaciones	
	Operaciones	**Proyectos**
ASPECTOS COMUNES	Ejecutados por gente	
	Restringidos por recursos limitados	
	Planeados, ejecutados y controlados	
ASPECTOS DIFERENCIADORES	Continuas	Temporarios
	Repetitivas	Única vez

Un proyecto es una actividad temporaria destinada a crear un producto o servicio único

- **Temporaria:** cada proyecto tiene un comienzo y fin definidos. Por lo tanto, los proyectos tiene una fecha de terminación (deadline), que puede ser auto-impuesta por el líder de proyecto o especificada externamente por un cliente o un organismo externo, estando fuera del control de cualquier integrante del proyecto.
- **Único:** el producto o servicio difiere en algo de todos los productos o servicios similares. Un proyecto nunca sucedió antes, no sucederá a futuro bajo las mismas circunstancias. Generalmente, las variaciones serán de naturaleza aleatoria– existen eventos aleatorios que pueden ocurrir, que nunca podemos estar seguros de cuando, como y que impacto tendrán en la planificación. Estos eventos aleatorios constituyen el desafío de los líderes de proyecto (LP).

Los proyectos tienen r**ecursos limitados**, tales como limitada cantidad de personal, dinero, o maquinas dedicadas al proyecto. Si bien los recursos pueden ser ajustados (aumentados o disminuidos) por la alta gerencia, se deben considerar fijos.
Los proyectos deben tener **un sólo objetivo**. Por eso, proyectos muy grandes y complejos pueden ser divididos en sub-proyectos. Cada uno de ellos, en sí mismo, constituye un proyecto. Esta división se hace para una mejor administración, simplificando la planificación de recursos. Sin embargo, la división en sub-proyectos produce interdependencia entre los mismos, agregando un nivel más de complejidad y comunicación.

El cliente espera un determinado nivel de funcionalidad y de calidad como resultado del proyecto. Estas expectativas pueden ser:

- auto-impuestas por el equipo de desarrollo - calidad en la documentación de programas fuentes.
- impuestas por el usuario - el listado de vencimientos semanal.

Causas frecuentes de fracaso en los proyectos informáticos	
Falta de compromiso de la dirección	Falta de comunicación en el equipo
Los usuarios no se involucran	Malas relaciones con otras partes interesadas en el proyecto
Falta de conocimiento técnico por parte del equipo	Mala definición de autoridad y roles dentro del equipo de proyecto
Plazo de ejecución no realistas	Documentación insuficiente de progreso y seguimiento
Mal ambiente de trabajo	Falta de comunicación en el equipo
Falta de estabilidad de la tecnología	Asignación inadecuada de personal en cantidad o en los perfiles

Ciclo de vida de un proyecto

Definición:

- Son todas las acciones para definir que se hará
- No se incluyen acciones propias para realización del proyecto
- Incluyen el establecimiento de los criterios de éxito
- Algunas tareas típicas:
  1. Reclutar el LP (Líder del proyecto)
  2. Obtener las reales necesidades del cliente
  3. Documentar estas necesidades
  4. Negociar con el cliente como estas necesidades serán cubiertas
  5. Escribir una descripción del proyecto (POS)
  6. Ganar la aprobación de los gerentes para planificar el proyecto

Distribución de los recursos

Fase 1	*Aprobación*	Conceptualización Análisis de viabilidad Selección y aprobación del proyecto	10%
Fase 2	*Definición*	Definición de requerimientos Análisis de riesgos Propuesta del proyecto	20%
Fase 3	*Planificación*	Especificaciones del proyecto Calendario de hitos Distribución de recursos	20%
Fase 4	*Ejecución*	Seguimiento y replanificación Gestión de cambios Gestión de incidencias	40%
Fase 5	*Cierre*	Entrega Evaluación Plan de seguimiento	10%

Planificación:

- Incluye todas las acciones para definir como se hará
- Algunas tareas típicas son:
  1. Definir el trabajo del proyecto
  2. Estimar cuanto tiempo llevara completarlo
  3. Estimar los recursos necesarios
  4. Estimar el costo total
  5. Secuenciar el trabajo
  6. Desarrollar el cronograma inicial
  7. Escribir un plan de contención de riesgos
  8. Documentar el plan del proyecto
  9. Ganar la aprobación de los gerentes para lanzar el proyecto

Lanzamiento

- Son acciones preparatorias para la ejecución del proyecto
- Algunas tareas típicas son:
  1. Reclutar el equipo del proyecto
  2. Emitir el documento de descripción del proyecto
  3. Establecer las reglas del equipo
  4. Establecer el proceso de cambio de alcance
  5. Administrar las comunicaciones del equipo

Montenegro, Micaela Soledad

6. Terminar el cronograma del proyecto
7. Describir los paquetes de trabajo

Monitoreo y control

- Incluye las acciones relacionadas con la ejecución del proyecto
- Algunas tareas típicas son:

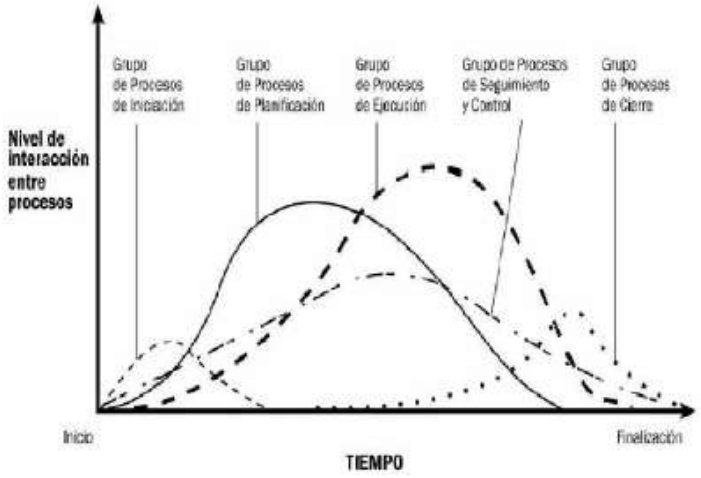
1. Establecer la performance y redactar los reportes
2. Monitorear riesgos
3. Reportar el estado del proyecto
4. Procesar requerimientos de cambio de alcance
5. Descubrir y resolver problemas de ejecución

Finalización o Cierre

- Incluye las acciones relacionadas con la evaluación del proyecto
- Algunas tareas típicas son:

1. Ganar la aprobación del cliente de que se han cumplido las metas
2. Instalar los entregables
3. Escribir el reporte final
4. Realizar la auditoría post-implementación

Superposición de los procesos a lo largo del proyecto



<u>Componentes de un proyecto.</u>

- Tarea. Pieza atómica del planeamiento y seguimiento dentro del proyecto.
- Actividad. Agrupa tareas relacionadas; termina con el cumplimiento de un hito.
- Función. Actividad que se prolonga durante todo el proyecto (como el control de calidad).

<u>Triángulo del alcance</u>

Los proyectos son sistemas dinámicos que deben ser mantenidos en equilibrio.

- Tiempo: es la ventana de tiempo en la cual el proyecto debe terminarse.
- Costo: es el presupuesto disponible para completar el proyecto.
- Recursos: es cualquier consumible usado en el proyecto. Ej: personas, equipos, tinta, papel. Son controlados por el líder del proyecto y necesitan ser identificados de manera independiente.

Cualquier imprevisto hace desbalancear el proyecto, por ejemplo: un recurso menos, un nuevo requerimiento.

¿Quién controla qué?

Líder de Proyecto	• Utilización de recursos • Cronograma de trabajo
Gerencia	• Niveles de recursos • Costos
Cliente	• Alcance • Niveles de calidad • Fechas de entrega

Corrimiento de Alcance: cualquier cambio en el proyecto que no estaba en el plan original. Es responsabilidad del LP acomodar el proyecto para tener en cuenta los cambios.

Corrimiento de Esperanza: informar que no hay demoras, para no dar una mala noticia, pensando que para el próximo informe de tareas se puede recuperar. Es responsabilidad del líder de proyecto controlar si los informes de avances son veraces. Se pueden realizar controles aleatorios.

Corrimiento de Esfuerzo: es el resultado del trabajo de miembros del equipo, que no representan progresos proporcionales al trabajo realizado. Ej: cada semana el informe de avance muestra progresos pero nunca se termina.

Corrimiento de Rasgos: sucede cuando miembros del equipo arbitrariamente agregan rasgos (características) que piensan que el usuario desearía tener en su sistema, pero que no especifico.

STAKEHOLDERS

El término se utiliza para referirse a cualquier persona o grupo que se verá afectado por el sistema, directa o indirectamente. Entre los stakeholders se encuentran los usuarios finales que interactúan con el sistema y todos aquellos en la organización que se pueden ver afectados por su instalación. Otros stakeholders del sistema pueden ser los ingenieros que desarrollan o dan mantenimiento a otros sistemas relacionados, los gerentes del negocio, los expertos en el dominio del sistema y los representantes de los trabajadores. Otros son:

- Gerente de proyecto. El individuo responsable de la gestión del proyecto.
- Cliente. Individuo u organización que usara el producto del proyecto. En general hay varios niveles.
- Organización ejecutora. La empresa cuyos empleados están más directamente involucrados en hacer el trabajo del proyecto
- Patrocinador. Individuo o grupo de la organización ejecutora que provee los recursos financieros (fondos o especies).

DERECHOS DEL CLIENTE.

- Establecer los objetivos del proyecto y que sean seguidos.
- Saber cuánto tiempo llevará y costará el proyecto de software.
- Decidir qué aspectos están en el software y cuáles no.
- Hacer cambios razonables a los req. durante el curso del proyecto y conocer su costo asociado.
- Conocer clara y seguramente el status del proyecto.
- Disponer regularmente de evaluaciones de riesgo que podrán afectar el costo, cronograma o calidad y disponer de opciones para atacar los problemas potenciales.
- Tener un rápido acceso a los entregables del proyecto.

DERECHOS DEL EQUIPO DE PROYECTO.

- Conocer los objetivos del proyecto y tener claridad de las prioridades.
- Conocer en detalle qué producto se espera construir y clarificar la definición del producto si no está clara.

- Rápido acceso a cualquier persona responsable de tomar decisiones sobre la funcionalidad del software.
- Trabajar cada fase del proyecto en forma técnicamente responsable, especialmente no ser forzado a codificar demasiado temprano en el proyecto.
- Aprobar las estimaciones del esfuerzo y cronogramas de cualquier trabajo que se pida de mí.
- Tener mi status del proyecto informando al cliente y la gerencia superior.
- Trabajar en un ambiente productivo, libre de interrupciones frecuentes, en especial en los momentos críticos del proyecto.

GESTIÓN DE PROYECTO

- Conjunto de métodos, técnicas y herramientas usadas para planificar y administrar un proyecto efectivamente, respecto de su alcance, tiempos, costos y calidad.
- "Es la aplicación de conocimiento, habilidades, herramientas y técnicas a las actividades de proyecto para satisfacer o exceder las necesidades y expectativas de los stakeholders de un proyecto." PMI
- Objetivos
  - Cumplir con los requerimientos del proyecto
  - Administrar adecuadamente los recursos
  - Utilizar experiencias exitosas para mejorar estos procesos (mejora continua)
  - Mayor previsibilidad y credibilidad

La buena gestión no puede garantizar el éxito del proyecto. Sin embargo, la mala gestión usualmente lleva al fracaso del proyecto. El software es entregado tarde, los costes son mayores que los estimados y los requerimientos no se cumplen.

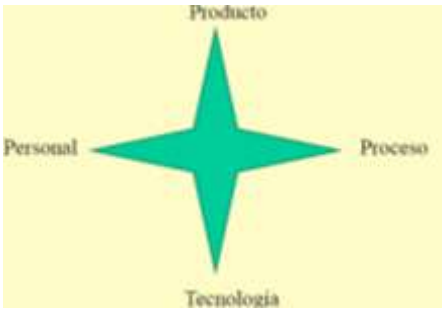
- La administración de proyectos de software es necesaria debido a que la ingeniería de software profesional siempre está sujeta a restricciones organizacionales de tiempo y presupuesto. Los gestores de software, tiene diferencias con otros gestores:
- El producto es intangible: No se puede ver ni tocar. Los gestores de proyectos de software no pueden ver el progreso. Confían en otros para elaborar la documentación necesaria para revisar el progreso.
- No existen procesos del software estándar: los procesos de software varían notablemente de una organización a otra.

Por lo general, los proyectos grandes de software son diferentes de proyectos previos. En consecuencia, los gestores, aun cuando cuenten con una amplia experiencia, ésta no es suficiente para anticipar los problemas. Más aún, los rápidos cambios tecnológicos en las computadoras y las comunicaciones hacen parecer obsoleta la experiencia previa.

Si bien las actividades de gestión dependen de la organización y del producto de software a desarrollar, las más importantes son: planificación, calendarización de proyectos y gestión de riesgos. Otras son la gestión de personal, la estimación de los costes de software y la gestión de la calidad.

Los gestores del proyecto son responsables de informar a los clientes y contratistas sobre el proyecto. Tienen que redactar documentos concisos y coherentes que resuman la información crítica de los informes detallados del proyecto. Además, su trabajo es asegurar que se cumplan las restricciones. Los gestores necesitan información para hacer su trabajo. Como el software es intangible, esta información sólo se puede proveer como documentos que describan el estado del software que se está desarrollando.

Gestión eficaz de un proyecto software se basa en:



Personal: se necesita personal preparado y motivado. Es el factor más valorado por los gestores de proyectos, ya que en última instancia los proyectos los desarrollan personas.

Equipo de desarrollo: Existen muchas razones por las que los proyectos de software tienen problemas. La escala de muchos esfuerzos de desarrollo es grande, lo que conduce a complejidad, confusión y dificultades significativas en la coordinación de los miembros del equipo. La incertidumbre es común, lo que da como resultado un torrente continuo de cambios que detienen al equipo de proyecto. La interoperabilidad se ha convertido en una característica clave de muchos sistemas. El software nuevo debe comunicarse con el software existente y ajustarse a las restricciones predefinidas impuestas por el sistema o por el producto. Para lidiar con esto, se deben establecer mecanismos para la comunicación formal e informal entre los miembros del equipo y los equipos.

Errores típicos:

- Motivación Débil
- Personal Mediocre
- Empleados problemáticos incontrolados
- Hazañas
- Anadir más gente a un proyecto retrasado
- Oficinas repletas y ruidosas
- Fricciones entre los clientes y los desarrolladores
- Expectativas poco realistas
- Falta de un promotor efectivo del proyecto
- Falta de participación de los implicados
- Política antes que desarrollo

Producto: Un gerente de proyecto de software se enfrenta con un dilema en el comienzo mismo de un proyecto de software. Se requieren estimaciones cuantitativas y un plan organizado, pero no hay información sólida disponible. Se pretende que el problema se resuelva desde el principio mismo del proyecto; cuando menos, debe establecer y acotar el ámbito del producto, el cual se determina por:
Contexto:

- Forma de integrarse el software en el sistema, producto o contexto de negocios mayor.
- Limitaciones resultantes del contexto.

Objetivos de información

- Datos visibles al cliente.
- Datos de entrada requeridos

Funciones y rendimiento

- Funciones realizadas por el software para transformar la información de entrada en salida.
- Características especiales de rendimiento.

Errores típicos:

- Planificación excesivamente optimista
- Gestión de riesgos insuficiente
- Planificación insuficiente
- Abandono de la planificación bajo presión

- Escatimar en las actividades iniciales
- Diseño inadecuado
- Escatimar en el control de la calidad
- Programación a destajo
- Control insuficiente de la directiva

Proceso: Existen fases genéricas de IS se encuentran presentes en todos los modelos de proceso, y otras que dependen del proyecto. El problema es aplicar el modelo de proceso más adecuado para el proyecto. Este modelo lo elige el gestor, en base al cliente y las características del producto.

Errores típicos:
- Exceso de requerimientos
- Cambio de las prestaciones
- Desarrolladores meticulosos
- Desarrollo orientado a la investigación

Tecnología: errores típicos
- Síndrome de la panacea: se cree que la tecnología hace mucho más de lo que realmente hace (bala de plata).
- Sobreestimación de las ventajas del empleo de nuevas herramientas o métodos
- Cambiar de herramientas a mitad del proyecto
- Falta de control automático del código fuente

PLANIFICACIÓN DEL PROYECTO

El gestor del proyecto debe anticiparse a los problemas que puedan surgir, así como preparar soluciones a esos problemas. Un plan, preparado al inicio de un proyecto, debe utilizarse como un conductor para el proyecto. Este plan inicial debe ser el mejor posible de acuerdo con la información disponible. Éste evolucionará conforme el proyecto progrese y la información sea mejor.
La planificación es un proceso iterativo que solamente se completa cuando el proyecto mismo se termina. Conforme la información se hace disponible, el plan debe revisarse regularmente.
El proceso de planificación se inicia con una valoración de las restricciones que afectan al proyecto (fecha de entrega requerida, personal disponible, presupuesto global, etcétera). Ésta se lleva a cabo en conjunto con una estimación de los parámetros del proyecto, como su estructura, tamaño y distribución de funciones. Entonces se definen los hitos de progreso y productos a entregar. En ese momento, el proceso entra en un ciclo. Se prepara un calendario para el proyecto y las actividades definidas en el calendario se inician o se continúan. Después de algún tiempo, se revisa el proyecto y se señalan las discrepancias. Debido a que las estimaciones iniciales de los parámetros del proyecto son aproximaciones, el plan siempre deberá actualizarse.
Si el proyecto se retrasa, tienen que renegociar con el cliente las restricciones del mismo y las entregas. Si esta renegociación no tiene éxito y no se puede cumplir el calendario, se debe llevar a cabo una revisión técnica. El objetivo de esta revisión es encontrar un enfoque alternativo que se ajuste a las restricciones del proyecto y cumpla con las metas del calendario.
Por supuesto, los gestores de proyectos inteligentes no suponen que todo irá bien. Las suposiciones iniciales y el calendario deben ser más bien pesimistas que optimistas.

Beneficios de la planificación
- Mejora la comunicación
- Minimiza el retrabajo
- Mejora la predictibilidad del cronograma
- Permite incluir tempranamente los controles de calidad sobre los productos
- Facilita la visualización del proyecto

Efectos cuando falta la planificación
- Estimaciones pobres = mayores costos
- Desvíos no controlados

- Recursos no planificados: dificultad de asignación
- Compromisos incumplidos
- Falta de retroalimentación entre proyectos
- Información sobre el estado generada a destiempo
- Problemas identificados a destiempo
- Riesgos no identificados y no informados

El plan de proyecto

Fija los recursos disponibles, divide el trabajo y crea un calendario de trabajo. Los detalles de este plan varían dependiendo del tipo de proyecto y de la organización. Las secciones más comunes son una introducción, la organización del proyecto (equipo de desarrollo, roles y gente involucrada), análisis de riesgo, requerimientos de recursos de hardware y software, división de trabajo, programa de proyecto, mecanismos de supervisión e informe.
Los tipos de planes de proyecto son:

- Plan de calidad: Describe procedimientos y estándares de calidad.
- Plan de validación: Descubre el enfoque, los recursos y la programación utilizado para validar el sistema.
- Plan de gestión de configuraciones.
- Plan de mantenimiento: Predice los requerimientos del mantenimiento del sistema, los costes y el esfuerzo.
- Plan de desarrollo: describe la metodología a utilizar para el desarrollo del proyecto.
- Plan de desarrollo personal: Describe como se adquirirán y desarrollarán los conocimientos y habilidades del personal.

Hitos y entregas
Cuando se planifica un proyecto, se debe establecer una serie de hitos, que representan el fin de una etapa lógica en el proyecto. Deben ser informes cortos de los logros en una actividad del proyecto.
Como regla general, las entregas son hitos, pero éstos no son necesariamente entregas, por ejemplo, pueden ser resultados internos del proyecto que son utilizados por el gestor del proyecto para verificar el progreso del mismo pero que no se entregan al cliente. Una entrega es el resultado del proyecto que se entrega al cliente.

Calendarización del proyecto
Implica separar todo el trabajo de un proyecto en actividades complementarias y considerar el tiempo requerido para completar dichas actividades, así como también los recursos necesarios en cada tarea. Por lo general, algunas de éstas se llevan a cabo en paralelo. Debemos coordinar estas actividades paralelas y organizar el trabajo para que la mano de obra se utilice de forma óptima. Deben evitarse situaciones en que el proyecto entero se retrase debido a que no se ha terminado una actividad crítica. Al realizar esta actividad, los gestores no deben suponer que cada etapa del proyecto estará libre de problemas.
Los calendarios se deben actualizar continuamente en la medida que se disponga de mejor información acerca del progreso.
Por lo general, el calendario del proyecto se representa como un conjunto de gráficos que muestran la división del trabajo, las dependencias de las actividades y la asignación del personal.

GESTIÓN DE RIESGOS

Una tarea importante del gestor de proyectos es anticipar los riesgos que podrían afectar a la programación del proyecto o a la calidad del software a desarrollar y emprender acciones para evitar esos riesgos, y además definir planes de contingencias por si ellos ocurren. Los resultados de este análisis de riesgos se deben documentar a lo largo del plan del proyecto junto con el análisis de consecuencias cuando el riesgo ocurra. Identificar éstos y crear planes para minimizar sus efectos en el proyecto se llama gestión de riesgos.

Un riesgo es una probabilidad de que una circunstancia adversa ocurra. El riesgo del proyecto es un evento incierto que en caso de que ocurra tendrá efecto negativo o positivo sobre los objetivos del proyecto.

Los riesgos involucran dos características: incertidumbre (el riesgo puede o no ocurrir; es decir, no hay riesgos 100 por ciento probables1) y pérdida (si el riesgo se vuelve una realidad, ocurrirán consecuencias o pérdidas no

deseadas). Cuando se analizan los riesgos es importante cuantificar el nivel de incertidumbre y el grado de pérdidas asociados con cada riesgo.

Elementos de riesgo – componentes:

- Evento (ej. lluvia -- riesgo inundación)
- Probabilidad: % de ocurrencia
- Impacto: cuanto afecta si ocurre

Los riesgos se pueden caracterizar como:

1. Riesgos del proyecto. Éstos afectan la calendarización o los recursos del proyecto. Un ejemplo podría ser la pérdida de un diseñador experimentado. Hacen que el calendario se deslice y los costos aumenten.
2. Riesgos del producto o técnicos. Éstos afectan a la calidad o al rendimiento del software que se está desarrollando. Un ejemplo podría ser que el rendimiento en un componente que hemos comprado sea menor que el esperado.
3. Riesgos del negocio o empresariales. Estos afectan a la organización que desarrolla o suministra el software. Por ejemplo, que un competidor introduzca un nuevo producto es un riesgo de negocio.

Estos tipos de riesgos no son exclusivos entre sí.

Otra clasificación más general es:

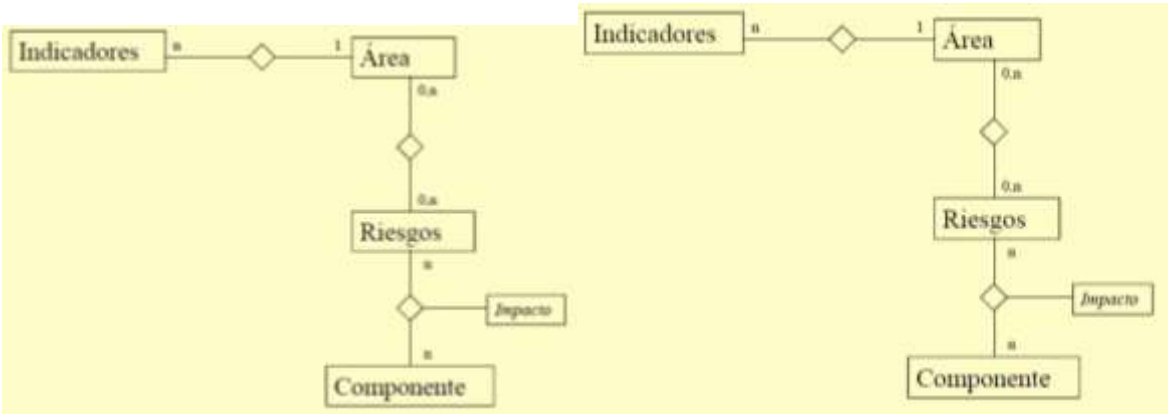
- Riesgos conocidos: se pueden descubrir evaluando el plan, el entorno técnico y comercial. Por ejemplo, fecha de entrega irreal, falta de requisitos documentados.
- Riesgos predecibles: se extrapolan de la experiencia en proyectos anteriores. Por ejemplo, rotación de personal, pobre comunicación con el cliente,
- Riesgos impredecibles: pueden ocurrir pero es difícil identificarlos por adelantado.

<u>Componentes y promotores de riesgo</u>

La fuerza aérea estadounidense publicó un escrito que contiene lineamientos para la identificación y reducción de los riesgos de software. El enfoque de la fuerza aérea requiere que el gerente del proyecto identifique los promotores de riesgo que afectan los componentes de riesgo de software:

- Riesgo de rendimiento: grado de incertidumbre de que el producto satisfará sus requisitos y se ajustará al uso pretendido.
- Riesgo de costo: grado de incertidumbre de que el presupuesto del proyecto se mantendrá.
- Riesgo de apoyo: grado de incertidumbre de que el software resultante será fácil de corregir, adaptar y mejorar.
- Riesgo de calendario: grado de incertidumbre de que el calendario del proyecto se mantendrá y de que el producto se entregará a tiempo.

El impacto de cada promotor de riesgo sobre el componente de riesgo se divide en una de cuatro categorías de impacto: despreciable, marginal, crítico o catastrófico.



Niveles en la gestión de riesgos

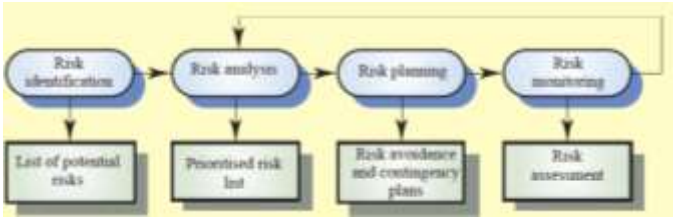
- Control de crisis: Apagar el fuego, controlar los riesgos sólo cuando se han convertido en problemas.
- Arreglar cada error: Detectar y reaccionar rápidamente ante cualquier riesgo, pero sólo después de que se haya producido.
- Mitigación de riesgos: Planificar con antelación el tiempo que necesitaría para cubrir riesgos en el caso de que ocurran, pero no intentar eliminarlos inicialmente.
- Prevención: Crear y llevar a cabo un plan como parte del proyecto para identificar y evitar que se conviertan en problemas.
- Eliminación de las causas principales. Identificar y eliminar los factores que puedan hacer posible la presencia de algún tipo de riesgo

Estrategias frente al riesgo

Las **estrategias reactivas de riesgo** se han llamado irrisoriamente la "escuela de gestión de riesgo de Indiana Jones". En las películas que llevan su nombre, Indiana Jones, cuando enfrenta una dificultad abrumadora, invariablemente dice: "No te preocupes, ¡pensaré en algo!". Al nunca preocuparse por los problemas hasta que suceden, Indy reaccionará en alguna forma heroica. Generalmente, el gerente promedio de proyectos de software no es Indiana Jones y los miembros del equipo del proyecto de software no son sus fieles ayudantes.

Una **estrategia proactiva** comienza mucho antes de iniciar el trabajo técnico. Los riesgos potenciales se identifican, su probabilidad e impacto se valoran y se clasifican por importancia. Luego, el equipo de software establece un plan para gestionar el riesgo.

Proceso de manejo de riesgos



1- Identificación de riesgos

Comprende el descubrimiento de los posibles riesgos del proyecto. Un método para identificar riesgos es crear una lista de comprobación de elementos de riesgo. Una categorización (lista de comprobación) de riesgos para identificarlos es:

- Tamaño del producto: riesgos asociados con el tamaño global del software que se va a construir o a modificar.
- Impacto empresarial: riesgos asociados con restricciones impuestas por la administración o por el mercado.
- Características de los participantes: riesgos asociados con la sofisticación de los participantes y con la habilidad de los desarrolladores para comunicarse con los participantes en forma oportuna.

Montenegro, Micaela Soledad

- Definición del proceso: riesgos asociados con el grado en el que se definió el proceso de software y la manera como se sigue por parte de la organización desarrolladora.
- Entorno de desarrollo: riesgos asociados con la disponibilidad y calidad de las herramientas por usar para construir el producto.
- Tecnología por construir: riesgos asociados con la complejidad del sistema que se va a construir y con lo "novedoso" de la tecnología que se incluye en el sistema.
- Tamaño y experiencia del personal: riesgos asociados con la experiencia técnica y de proyecto global de los ingenieros de software que harán el trabajo.

Existen dos tipos distintos de riesgos en esta categorización: Los riesgos genéricos son una amenaza potencial a todo proyecto de software. Los riesgos específicos del producto pueden identificarse solamente por quienes tienen clara comprensión de la tecnología, el personal y el entorno específico del software que se construye.

2- Análisis de riesgos

También se conoce como estimación de riesgo o proyecto de riesgo. En esta actividad se califica cada riesgo en dos formas: 1) la posibilidad o probabilidad de que el riesgo sea real y 2) las consecuencias de los problemas asociados con el riesgo, en caso de que ocurra. Esto recae en la opinión y experiencia del gestor del proyecto. Generalmente, no se hace una valoración con números precisos sino en intervalos.

El resultado de este proceso de análisis se debe colocar en una tabla, que tiene una lista de los riesgos en la primera columna y en otras columnas contiene: clasificación, probabilidad de ocurrencia, valoración del impacto del riesgo, y el MMMR (plan de mitigación, monitoreo y manejo de riesgo).

Nota: El manejo del riesgo y la planificación de contingencia suponen que los esfuerzos de mitigación fracasaron y que el riesgo se convirtió en realidad. El monitoreo se explica en la siguiente hoja.

La tabla se debe ordenar por probabilidad y prioridad. Es posible definir una línea de corte, que implica que sólo los riesgos que se encuentran por arriba de la línea recibirán mayor atención, de esta manera se desprecian riesgos poco probables y los medianamente probables con poco impacto.

Riesgo	Categoría	Proba-bilidad	Impacto	RMMM
Cliente cambie los requerimientos	Proecto	80 %	Crítico	
Falta de experiencia en herramientas	Entorno Desarrollo	80 %	Marginal	
Rotación demasiado alta	Equipo	60 %	Crítica	
Tamaño estimado demasiado pequeño	Proyecto	40 %	Crítico	
Mas número de usuarios de lo planificado	Proyecto	30 %	Marginal	

La tabla se debe actualizar durante cada iteración del proceso de riesgos, cambiando conforme se disponga mayor información.

3- Planificación de riesgos

Considera cada uno de los riesgos clave que han sido identificados, así como las estrategias para gestionarlos. No existe un proceso que permita establecer los planes de gestión de riesgo, depende del juicio y de la experiencia del gestor del proyecto. Las estrategias se pueden dividir en:

1- Estrategias de prevención o evasión. Siguiendo estas estrategias, la probabilidad de que el riesgo aparezca se reduce.
2- Estrategias de minimización. Siguiendo estas estrategias se reducirá el impacto del riesgo.
3- Planes de contingencia. Seguir estas estrategias es estar preparado para lo peor y tener una estrategia para cada caso.

EL OBJETIVO: la función del gerenciamiento del riesgo es Alejar la incertidumbre del Riesgo y Acercarla a la Oportunidad

Montenegro, Micaela Soledad

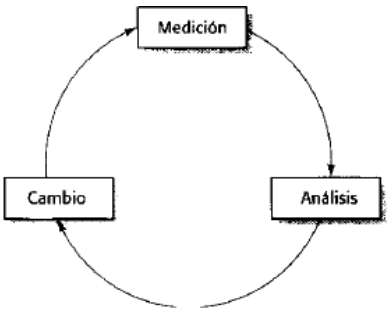
4- Supervisión o monitorización de riesgos

Es una actividad de seguimiento del proyecto con tres objetivos principales:

- Valorar si los riesgos predichos en efecto ocurren,
- Asegurar que los pasos para evitar el riesgo definidos para un riesgo determinado se aplican de manera correcta y
- Recopilar información que pueda usarse para futuros análisis de riesgos.
- Intentar asignar orígenes (cuál riesgo causó cuál problema a lo largo del proyecto).

Implementar el plan de gestión de riesgos

- Ejecutar el plan como parte del plan de proyecto
- Ejecutar las medidas de mitigación en la medida que los riesgos vayan ocurriendo
- Comunicar y documentar las acciones realizadas (learned lessons)
- Evaluar los resultados
  - Estado general del proyecto
  - Re-evaluar los riesgos, probabilidades, impactos y eventos
  - Re-evaluar estrategias de mitigación, éxitos, fracasos, eventos y riesgos re-evaluados
  - Ajustar el plan
  - Comunicar cambios y estado.



4. MEJORA DEL PROCESO SOFTWARE

Frente a la "Crisis del Software", donde los proyectos se caracterizan por:

- Terminar fuera de tiempo,
- con más costos de los programados,
- con menos funcionalidad que la deseada,
- con dificultad para cerrar los proyectos,
- y con dudosa calidad,

Surge la necesidad de la creación de modelos de calidad para empresas de software.

La mejora de procesos significa entender los procesos existentes y cambiarlos para mejorar la calidad del producto y/o reducir los costos y el tiempo de desarrollo.

Los procesos también tienen atributos ej: comprensión (¿hasta que punto se define completamente el proceso y como de fácil es comprender su definición?), aceptación (¿el proceso definido es aceptable y utilizable por los ingenieros responsables de producir software?), robustez (¿puede continuar el procesos a pesar de los probleas inesperados?); no es posible hacer la mejora de procesos para todos sus atributos simultáneamente.

La mejora de procesos es una actividad cíclica y tiene 3 estados principales:

1. Proceso de medición de los atributos del proyecto actual o del producto. El objetivo es mejorar las mediciones de acuerdo con las metas de la organización involucrada.
2. Proceso de análisis, el proceso es valorado y se identifican puntos flacos y cuellos de botella.
3. Introducción los cambios.

La mejora de procesos es una actividad a largo plazo y continúa ya que el entorno del negocio cambia y los procesos evolucionan para tener en cuenta estos cambios.

El término mejoramiento del proceso de software (MPS) implica:

- que los elementos de un proceso de software efectivo pueden definirse en forma efectiva

- que un enfoque organizacional existente sobre el desarrollo del software puede valorarse en contraste con dichos elementos
- que es posible definir una estrategia de mejoramiento significativa.

Modelo

Un modelo es una representación del mundo real, los cuales ayudan a entender y aplicar ideas de desarrollo de procesos. Un modelo se usa:

- Para determinar objetivos y prioridades de mejoramiento
- Para asegurar procesos estables, maduros y capaces
- Como una guía para mejoramiento organizacional
- Como una guía de asesoramiento, para diagnosticar el estado de los esfuerzos de mejoramiento

MODELO DE MADURACIÓN DE CAPACIDADES

Un modelo de madurez se aplica dentro del contexto de un marco conceptual MPS. La intención del modelo de madurez es proporcionar un indicio global de la "madurez del proceso" que muestra una organización de software, es decir, un indicio de la calidad del proceso de software, el grado en el que los profesionales entienden y aplican el proceso, y el estado general de la práctica de ingeniería del software. Esto se logra usando algún tipo de escala ordinal.

Estos modelos proponen un conjunto de prácticas que las organizaciones pueden adoptar para implantar procesos productivos más efectivos. Son llamados modelos de madurez porque proponen adoptar dichas prácticas en forma gradual: primero deben ponerse en práctica áreas de proceso pertenecientes a un nivel determinado, para luego, sobre esta base, introducir las correspondientes al nivel siguiente.

MODELO DE MADUREZ DE CAPACIDADES (CMM)

El Modelo de Madurez de Capacidades o CMM (Capability Maturity Model), es un modelo de evaluación de los procesos de una organización. Fue desarrollado inicialmente para los procesos relativos al desarrollo e implementación de software por la Universidad Carnegie-Mellon para el SEI (Software Engineering Institute).

Al modelo CMM lo siguieron otros modelos por ej: el modelo de madurez de la capacidad de personal (P-CMM-2001).

El proyecto Bootstrap tenía el objetivo de extender adaptar el modelo de madurez del SEI para hacerlo aplicable a un amplio espectro de compañías. Dicho proyecto utiliza los niveles de madurez del SEI pero además incorpora:

1- Guías de calidad para ayudar en la mejora de procesos de las compañías.
2- Una distinción importante entre organización metodología y tecnología.
3- Un modelo de proceso base que podría adoptarse.

El Modelo de Madurez de Capacidades (CMM) establece un conjunto de prácticas o procesos clave agrupados en Áreas Clave de Proceso. Para cada área de proceso define un conjunto de buenas prácticas que habrán de ser:

- Definidas en un procedimiento documentado
- Provistas de los medios y formación necesarios
- Ejecutadas de un modo sistemático, universal y uniforme (institucionalizadas)
- Medidas
- Verificadas

A su vez estas Áreas de Proceso se agrupan en cinco "niveles de madurez", de modo que una organización que tenga institucionalizadas todas las prácticas incluidas en un nivel y sus inferiores, se considera que ha alcanzado ese nivel de madurez.

Los niveles son:

1. Inicial: Pocos procesos definidos. Las organizaciones en este nivel no disponen de un ambiente estable para el desarrollo y mantenimiento de software. Aunque se utilicen técnicas correctas de ingeniería, los esfuerzos se ven minados por falta de planificación. El éxito de los proyectos se basa la mayoría de las veces en el esfuerzo personal, aunque a menudo se producen fracasos y casi siempre retrasos y sobrecostes. El resultado de los proyectos es impredecible.
2. Repetible. En este nivel las organizaciones disponen de unas prácticas institucionalizadas de gestión de proyectos, existen unas métricas básicas y un razonable seguimiento de la calidad. La relación con subcontratistas y clientes está gestionada sistemáticamente.
3. Definido. Además de una buena gestión de proyectos, a este nivel las organizaciones disponen de correctos procedimientos de coordinación entre grupos, formación del personal, técnicas de ingeniería más detalladas y un nivel más avanzado de métricas en los procesos. Se implementan técnicas de revisión por pares (peer reviews).
4. Gestionado. Se caracteriza porque las organizaciones disponen de un conjunto de métricas significativas de calidad y productividad, que se usan de modo sistemático para la toma de decisiones y la gestión de riesgos. El software resultante es de alta calidad.
5. Optimizado. La organización completa está volcada en la mejora continua de los procesos. Se hace uso intensivo de las métricas y se gestiona el proceso de innovación. La organización tiene sistemas de realimentación cuantitativa en su lugar para identificar las debilidades del proceso y fortalecer esos puntos de manera proactiva. Los equipos de proyecto analizan defectos para determinar sus causas; los procesos de software se evalúan y actualizan para evitar que recurran tipos conocidos de defectos.

Con la excepción del primer nivel, cada uno de los restantes Niveles de Madurez está compuesto por un cierto número de Áreas Claves de Proceso. Cada una de éstas identifica un conjunto de actividades y prácticas interrelacionadas, las cuales cuando son realizadas en forma colectiva permiten alcanzar las metas fundamentales del proceso. Las Áreas Claves del proceso pueden clasificarse en 3 tipos de proceso: Gestión, Organizacional e Ingeniería.

Las prácticas que deben ser realizadas por cada Área Clave de Proceso están organizadas en 5 Características Comunes, las cuales constituyen propiedades que indican si la implementación y la institucionalización de un proceso clave es efectivo, repetible y duradero. Estas 5 características son:

- Compromiso de la realización
- La capacidad de realización
- Las actividades realizadas
- Las mediciones y el análisis
- La verificación de la implementación.

Estructura del CMM

Un nivel de madurez es un esfuerzo evolutivo bien definido para alcanzar un proceso de software maduro. Cada nivel de madurez está compuesto por un conjunto de objetivos de proceso que, cuando se satisfacen, establecen un componente importante del proceso de software.

Cada nivel de madurez ha sido descompuesto en partes constituyentes. , on excepción del nivel inicial, se compone de varias Áreas Claves de Proceso (son claves ya que son determinantes para alcanzar el nivel de madurez) .

Las áreas clave de proceso identifican los problemas que se deben tratar para alcanzar un nivel de madurez. Una organización que está en el nivel 3 ha alcanzado todas las áreas clave de proceso de los niveles 2 y 3. Cada área clave de proceso identifica un grupo de actividades relacionadas que, cuando se realizan colectivamente, se consiguen un conjunto de metas consideradas importantes para mejorar la capacidad del proceso.

Las metas de cada área clave de proceso deben resumirse en prácticas clave y pueden usarse para determinar cuándo una organización o proyecto ha implementado eficientemente el área clave de proceso. Las metas significan la visibilidad, los límites y la intención de cada área clave de proceso. Al adaptar las prácticas clave de un área clave de proceso a un proyecto o situación de una organización específica, las metas se pueden usar para determinar si la adaptación es razonable o no. De forma similar, cuando se evalúan alternativas de implementación

de un área clave de proceso, las metas se pueden utilizar para determinar si la alternativa satisface la intención del área clave de proceso.

Áreas claves de proceso por nivel:

- Nivel 2: Las áreas claves son:
  - Administración de requerimientos
  - Planeamiento del proyecto de software
  - Control de proyectos de software
  - Administración de subcontratos
  - Aseguramiento de la calidad del software
  - Administración de configuración de software:
- Nivel 3: Las áreas claves son:
  - Concentración del proceso organizacional
  - Definición de procesos organizacionales
  - Programa de capacitación
  - Administración integral de software
  - Ingeniería de productos de software
  - Coordinación entre grupos
  - Revisiones puntuales
- Nivel 4: Las áreas claves son:
  - Administración cuantitativa de procesos
  - Administración de calidad de software
- Nivel 5 : Las áreas claves son:
  - Prevención de Defecto
  - Administración de Cambio de Tecnología
  - Administración del Cambio del Proceso

Ejemplo de área clave de proceso

Por ejemplo, el nivel de madurez 2, definido, tiene como objetivo establecer la administración y control básico del proyecto.

Para una de sus áreas de procesos, "Administración de Requerimientos" cuyo objetivo es establecer un entendimiento común entre el cliente y los requerimientos del proyecto, se definen como prácticas claves:

- El grupo de ingeniería de software revisa los requerimientos antes de que se incorporen al proyecto de software.
- El grupo de ingeniería de software usa los requerimientos encontrados como base para los planes, productos y actividades. Los requerimientos encontrados:
- Se revisan los cambios en los requerimientos encontrados y se incorporan al proyecto.

MODELO DE MADUREZ DE CAPACIDADES INTEGRADO

El SEI se embarcó en un nuevo programa para desarrollar un modelo de capacidad integrado (CMMI-2001) que integra otros modelos de ingeniería, utiliza las mejores prácticas de Ingeniería de Software (IS), proporciona guías para mejorar los procesos de la organización y la capacidad para gestionar el desarrollo, adquisición, mantenimiento de productos y servicios .

CMMI es un metamodelo de proceso exhaustivo que se impulsa en un conjunto de sistemas y capacidades de ingeniería del software que deben presentarse conforme las organizaciones alcanzan diferentes niveles de capacidad y madurez del proceso.

Tiene dos instancias en etapas que es compatible con el CMM de software y permite un desarrollo del sistema de la organización, la gestión de los procesos a valorar y su asignación a un nivel de Madurez entre 1 y 5, y su versión continua permite una clasificación más detallada y considera 24 áreas de procesos.

CMMI es un modelo que contiene cinco niveles de madurez los cuales describen un camino evolutivo que va desde una organización caótica a una organización con procesos de software maduros y disciplinados. Está basado en conceptos de Calidad Total y de mejoramiento continuo y fue creado por el SEI en el año 2000 basado en modelos prexistentes.

Beneficios

- Reducción de costos por
    - Estimaciones basadas en hechos.
    - Reducción de reprocesos.
    - Acuerdos claros sobre el servicio y la funcionalidad del producto a entregar.
- Aumento en la confiabilidad por
    - Reducción consistente de errores.
    - Cumplimiento consistente de fechas.
- Mayor efectividad por
    - Visibilidad sobre el proceso y sobre el producto.
    - Operar con estándares documentados.
    - Personal entrenado.

Modelo CMMI

Áreas de proceso: El CMMI identifica 24 áreas de procesos que son relevantes para la capacidad y mejora del proceso de software. Estas están organizadas en 4 grupos en el modelo CMMI continuo. Las áreas de proceso se clasifican por Categorias (4: Gestión de proyectos, Soporte, Ingeniería y Gestión de Procesos) en el Modelo CMMI continuo, y por niveles de madurez(5) en el modelo CMMI por etapas.

Ejemplos de Áreas de proceso:

- Planificación del proyecto
- Control y seguimiento del proyecto
- Gestión de riesgos

Otro ejemplo: Para la categoría “Ingeniería” algunas de las áreas de procesos asociadas son:

- Gestión de requerimientos
- Desarrollo de requerimientos
- Verificación
- Validación

Metas: son descripciones abstractas de un estado deseable que debería ser alcanzado para una organización. El CMMI tiene metas específicas asociadas a cada área de proceso y que definen el estado deseable para esta área. También tiene metas genéricas que son asociadas con las buenas prácticas.

Ejemplos de Metas:

Un ejemplo de meta genérica es: “El proceso es institucionalizado como un proceso definido” y ejemplo de metas específicas asociada al área de proceso “Seguimiento y control del proyecto” pueden ser:

- Gestión de acciones correctivas cuando el rendimiento del proyecto o sus resultados se desvían del plan
- El rendimiento actual y el progreso del proyecto es monitorizado comparándolo con el plan de proyecto

Prácticas: son descripción de vías para conseguir una meta. Se pueden asociar hasta 7 prácticas específicas o genéricas con cada meta dentro de cada área de proceso.

Ejemplos de Prácticas:

Asociada a la meta: “Los requerimientos son analizados y validados, y se define la funcionalidad requerida” se pueden definir las siguientes prácticas:

- Análisis de los requerimientos derivados para asegurar que son necesarios y suficientes
- Validar los requerimientos para asegurar que el producto resultante funciona tal y como se pretendía en el entorno del usuario, utilizando múltiples técnicas cuando sea apropiado

Las metas y las prácticas dependen de la madurez de la organización.

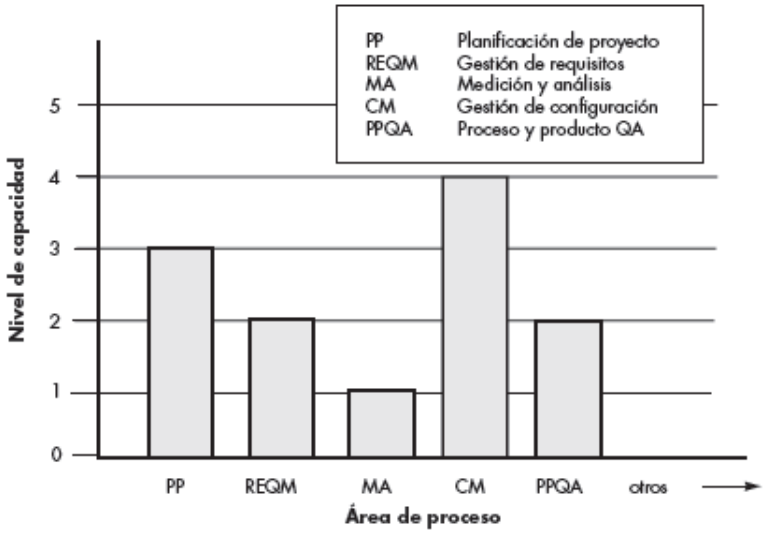
Formas de implementar

CMMI provee dos representaciones:

Continua

- Permite entre otras cosas, seleccionar el orden de implementación que mejor reúna lo objetivos de la organización.
- Los niveles indican el grado de capacidad del proceso
- Los niveles se definen para cada área de proceso.Existen 6 niveles (0-5)

Esta representación Continua favorece la flexibilidad en el orden hacia el cual se dirigen las mejoras. A medida que una organización del SW adquiere madurez en suproceso, esta lo institucionaliza.



Aquí cada área de proceso (por ejemplo, planificación de proyecto o gestión de requisitos) se valora formalmente contra metas y prácticas específicas y se clasifica de acuerdo con los siguientes niveles de capacidad:

- Nivel 0: Incompleto: también llamado No Productivo. Aquí no se satisfacen una o más de las metas específicas asociadas con el área de proceso.
- Nivel 1: Realizado: se satisface las metas asociadas al área de proceso, y para todos los procesos del ámbito del trabajo a realizar fijado y comunicado a los miembros del equipo.
- Nivel 2: Administrado: las metas asociadas al área de proceso son conocidas y tienen lugar políticas organizacionales que definen cuando se debe utilizar cada proceso.
- Nivel 3: Definido: se centra en la estandarización organizacional y el desarrollo de procesos.
- Nivel 4: Administrado cuantitativamente: existe una responsabilidad organizacional de usar métodos estadísticos y otros métodos cuantitativos para controlar los subprocesos.
- Nivel 5: Optimizado: : analizar las tendencias y adaptar los procesos a las necesidades de los cambios del negocio.

En etapas

- Provee una probada secuencia de mejoras, comenzando con prácticas básicas de gestión y progresando por medio de un predefinido y probado camino de sucesivos niveles. Cada uno de estos niveles es el fundamento para el siguiente

- Los niveles indican el grado de madurez organizacional
- Los niveles se basan en agrupamientos de áreas de proceso. Existen 5 niveles (1-5)

La principal diferencia entre el modelo continuo y el en etapas, es que el primero define niveles de capacidad mientras que en el segundo define niveles de madurez. Los niveles de madurez son:

- Nivel 1: Inicial
- Nivel 2: Administrado
- Nivel 3: Definido
- Nivel 4: Administrado cuantitativamente
- Nivel 5: Optimizado

Las principales ventajas de la representacion por etapas son:

- Secuencia contrastada de mejoras, inicia con las prácticas de gestión básica de proyectos.
- Permite comparaciones dentro de la organización y entre diferentes organizaciones.
- Fácil migración desde el modelo SW-CMM al CMMI.
- Permite efectivizar una mejora no sólo de productos sino también de procesos.

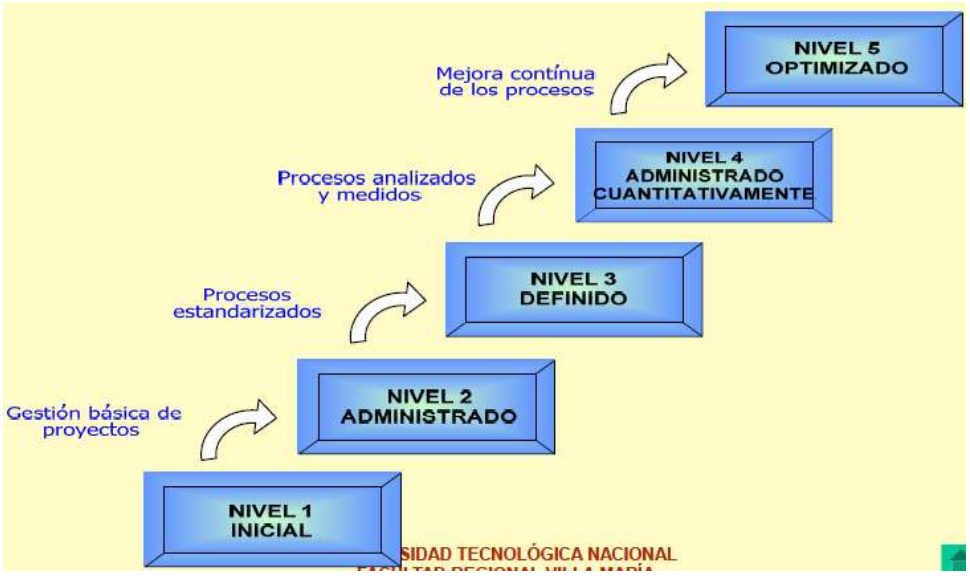
En cuanto a las desventajas podemos mencionar que la adopción de éste modelo puede resultar costosa para organizaciones pequeñas.

Disciplinas del modelo

- Ingenieria de sistemas: desarrollo total de sistemas, software o no software.
- Gestion de proveedores: adquisicion de productos de trabajos externos.
- Desarrollo integrado de prodectos y procesos: acercamiento sistematico, colaboracion entre partes, satisfacer requerimientos de clientes.
- Ingenieria de software: desarrollo de sw, aplicación sistematica, disciplina, cuantitativa, desarrollo, operación, mantenimiento

Niveles de madurez

El modelo describe las metas que se deben alcanzar en cada uno de estos niveles. La mejora de procesos se lleva a cabo implementando prácticas en cada nivel, subiendo desde el nivel inferior hasta el superior del modelo.



1- Inicial: La empresa no dispone de procesos y controles definidos.

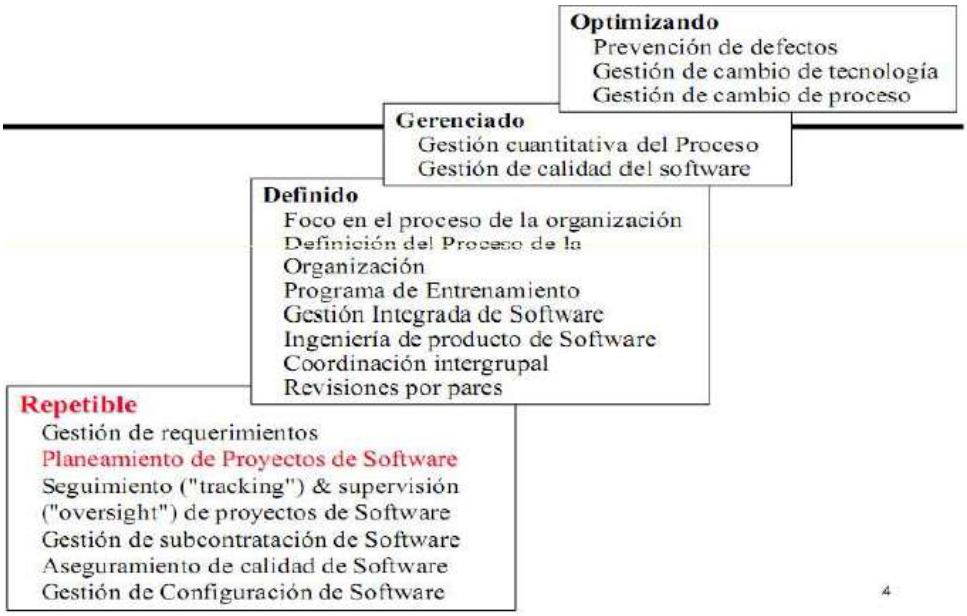
Se trabaja con procedimientos que no están normalizados, es decir, procedimientos tanto del propio desarrollo de software como de su planificación y control, que no están establecidos explícitamente antes de su uso.
La característica de las empresas que se encuentran en este nivel es que no hay un control de la gestión de proyectos software efectivo, porque puede suceder que la empresa disponga de procedimientos y técnicas formales, tanto de gestión como del proceso, y de herramientas, pero no se utilizan de manera estándar en todos los proyectos.

2- Repetible: La empresa tiene métodos estandarizados facilitando procesos repetibles.
Las empresas que se encuentran en este nivel son las que disponen de un control básico de la gestión de proyectos, gestión de calidad y gestión de la configuración.
El problema en este tipo de organización es que introducir cualquier cambio tiene un alto grado de riesgo de fracaso.

3- Definido: La empresa monitoriza y mejora sus procesos.
Las empresas que se encuentran en este nivel se caracterizan por disponer de:
   - Un grupo de proceso, cuyo objetivo es el de mejorar el proceso software.
   - Una metodología de desarrollo software que describa las actividades técnicas y de gestión requeridas para la adecuada ejecución del proceso.

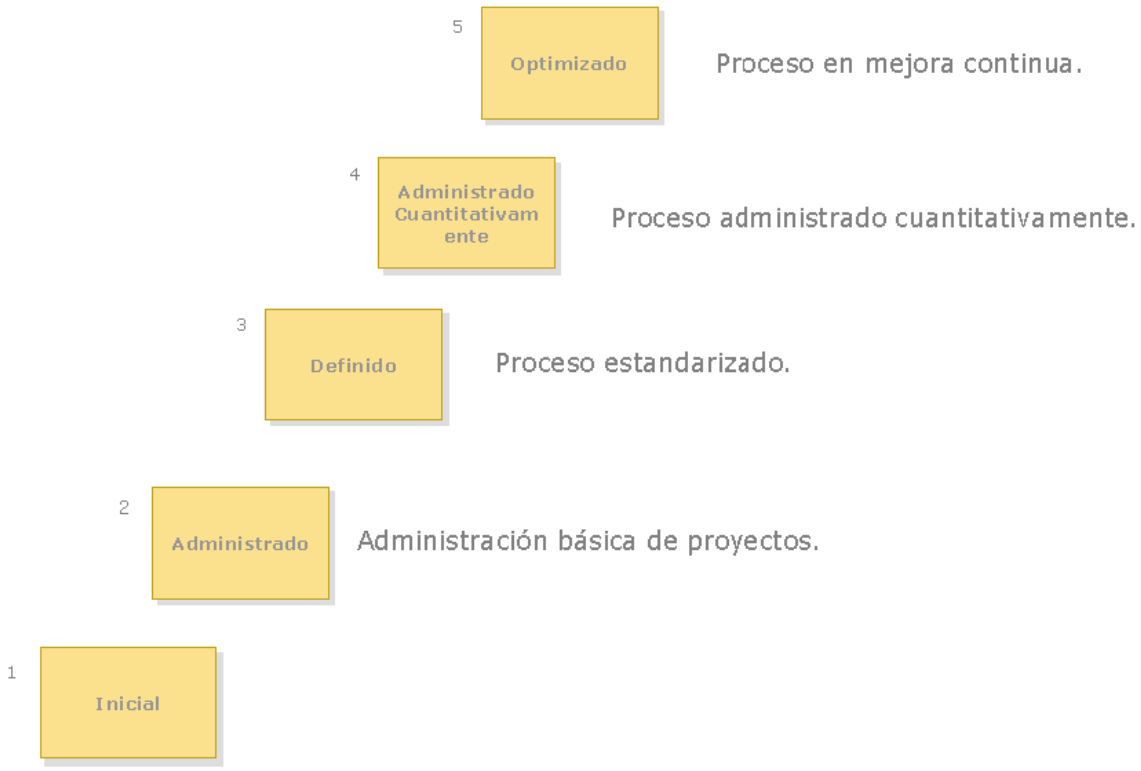
4- Administrado cuantitativamente: La empresa posee controles avanzados, métricas y retroalimentación.
Las empresas que han alcanzado este nivel disponen de un control de los costos y calidad de las principales etapas del proceso. Es prerequisito que exista una metodología de desarrollo software para realizar una medición efectiva.

5- Optimización: La empresa emplea métricas con propósitos de optimización.
En este nivel, las organizaciones se encuentran en un proceso de mejora continua. Se usan todos los procesos y técnicas modernas, lo mismo que la administración cuantitativa. Las organizaciones se enfocan en la mejora a través de técnicas y procesos de prevención de defectos, cambios en tecnología y cambios en procesos.

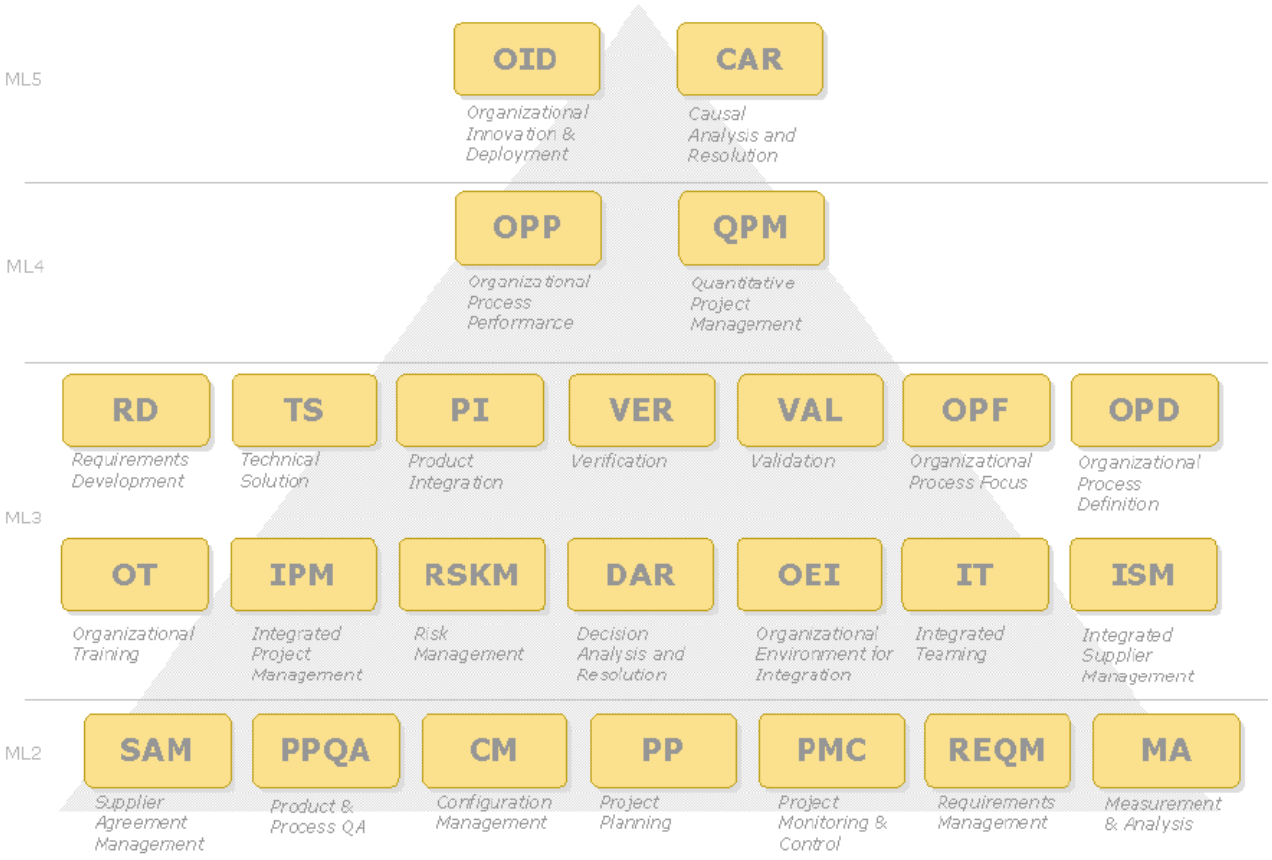


Estructura

CMMI plantea que las organizaciones pueden ubicarse en alguno de cinco posibles niveles de madurez, dependiendo del grado de sofisticación de sus procesos.

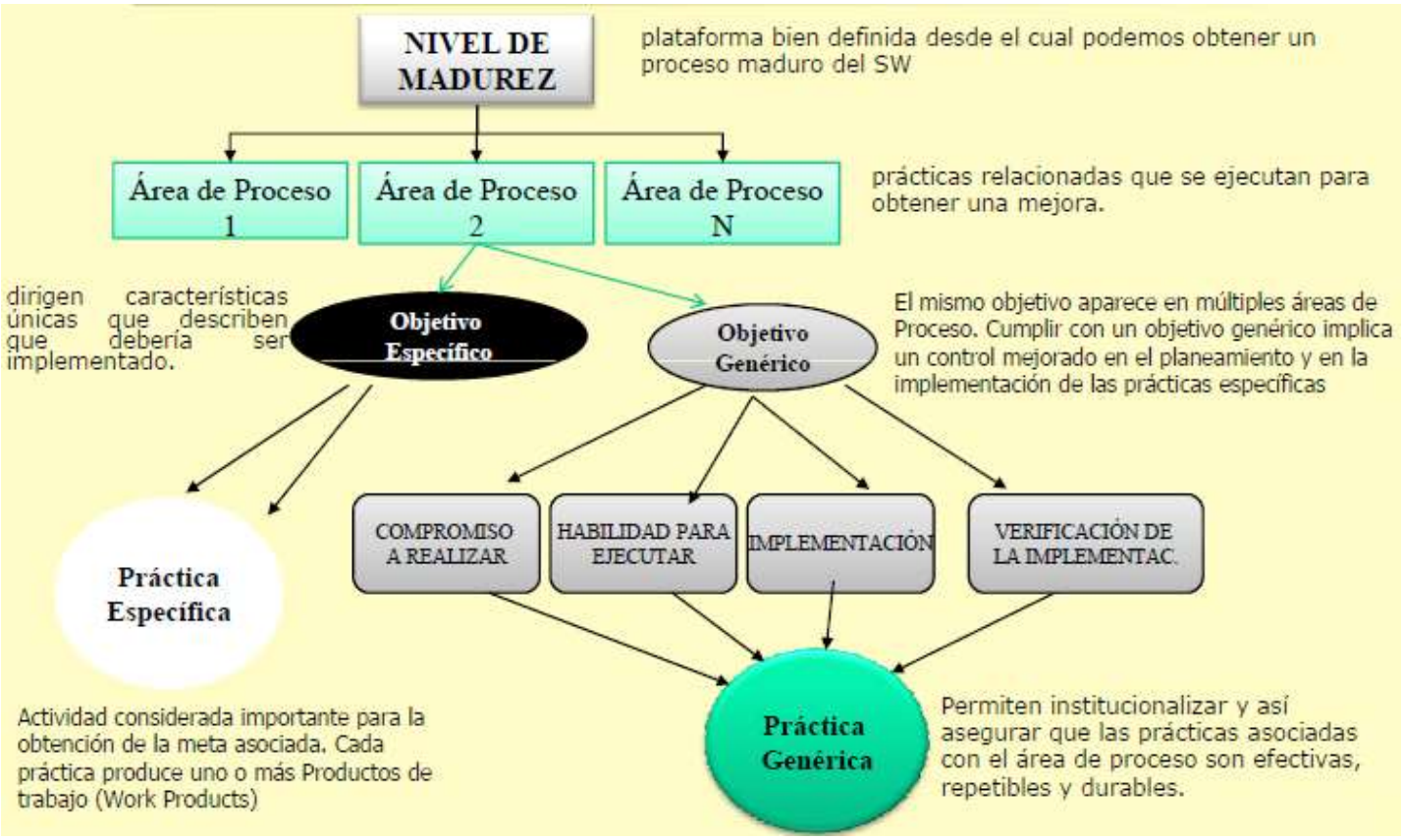


A su vez, cada nivel de madurez -con excepción del inicial- queda caracterizado por un conjunto de áreas de proceso que agrupan prácticas que, al ser ejecutadas colectivamente, permiten cumplir con algún objetivo que es considerado importante para el modelo. Cabe destacar que antes de introducir prácticas de un nivel determinado deben estabilizarse las prácticas del nivel anterior. En la siguiente imagen podemos ver la representación por niveles (áreas de proceso por nivel de madurez):



Además de poder ver las áreas de proceso por nivel de madurez, el modelo propone una vista alternativa (llamada continua) en donde las áreas están agrupadas por categoría, donde las mismas son: Gestión de Proyecto, Soporte, Ingeniería y Gestión de Procesos. En la siguiente imagen podemos ver la representación continua (áreas de proceso por categoría)

En la representación por niveles, cada nivel de madurez contiene varias áreas de proceso, las que a su vez quedan definidas por uno o varios objetivos específicos y un objetivo genérico. Cada uno de ellos tiene vinculado un conjunto de prácticas, llamadas específicas y genéricas respectivamente.



- Objetivo y Prácticas Genéricas
  Los objetivos y prácticas genéricas tienen que ver con el grado de institucionalización de los procesos (compromiso con la ejecución, capacidad para ejecutar, dirección de la ejecución, verificación de la ejecución) Son llamados así porque son los mismos en todas las áreas de proceso (aunque hay aspectos específicos para cada una de ellas).
  Objetivos genéricos:
  - Compromiso para la ejecución: Acciones que deben tomarse para asegurar que el proyecto se establezca y perdure.
  - Capacidad para ejecutar: Precondiciones que deben existir (en el proyecto u organización) para implementar el proceso
  - Actividades ejecutadas: Roles y procedimientos necesarios para implementar un área clave de proceso.
  - Mediciones y análisis: Medir el proceso y analizar las mediciones
  - Verificación de implementación: Pasos para asegurar que las actividades se ejecutan de acuerdo con el proceso que se ha implementado
- Objetivos y Prácticas Específicas
  Los objetivos y prácticas específicas están vinculados a un área de proceso determinada. Son considerados elementos que deben ser satisfechos para implementar exitosamente los procesos relacionados con un área de proceso en particular.

<u>Ejemplo</u>

A modo de ejemplo tomaremos el nivel 2: “Gestionado”. Lo que permite a un proceso ser gestionado es:

- Establecer y mantener una política organizacional de planificación y mejora del proceso de planificación.
- Proveer los recursos adecuados para mejorar el proceso de gestión del proyecto, desarrollando las herramientas y proveyendo los servicios del proceso.
- Seguimiento y control del proyecto y aplicación de las medidas correctivas adecuadas cuando sea necesario.

- Revisión de actividades, estado y resultados del proceso de planificación del proyecto con una gestión a alto nivel y resolución de problemas.

Las areas de proceso definidas para el nivel 2 del modelo, nivel gestionado, son:

- Gestión de requerimientos (REQM): gestiona los requerimientos del proyecto y de sus componentes e identifica inconsistencias.
- Planificación del proyecto (PP): establece y mantiene los planes que definen las actividades del proyecto.
- Seguimiento y control del proyecto (PMC): comprensión del progreso del proyecto y la aplicación de medidas correctivas cuando el plan del proyecto se desvía.
- Acuerdos con los proveedores (SAM): adquisición de productos y servicios de proveedores externos con los cuales existen acuerdos.
- Análisis y mediciones (MA): desarrolla y mantiene una capacidad de medición que se utiliza en el soporte de gestión de la información.
- Garantía de la calidad del proceso y del producto (PPQA): provee personal y gestión con comprensión objetiva del proceso y los producto de trabajo asociados.
- Gestión de configuraciones (CM): establece y mantiene la integridad de los productos de trabajo usando identificación, control y estado de configuraciones.

Para el nivel 2, se define el siguiente objetivo genérico (GG), y las siguientes prácticas genéricas (GP):

- GG 2 Institucionalizar un proceso administrado
  - GP 2.1 Establecer políticas organizacionales
  - GP 2.2 Planificar el proceso
  - GP 2.3 Proveer Recursos
  - GP 2.4 Asignar responsabilidades
  - GP 2.5 Entrenar al personal
  - GP 2.6 Administrar la configuración
  - GP 2.7 Identificar e involucrar a los interesados
  - GP 2.8 Monitorear y controlar los procesos
  - GP 2.9 Evaluar adhesión objetivamente
  - GP 2.10 Revisar el estado con la alta gerencia

Por ejemplo, en el área Administración de Requerimientos (REQM), los objetivos y prácticas específicos son las siguientes:

Objetivo Específico	Prácticas Específicas
SG 1 Administrar Requerimientos *Los requerimientos son administrados, y se identifican las inconsistencias entre los requerimientos y los planes y otros artefactos del proyecto.*	SP 1.1 Comprender el significado de los requerimientos SP 1.2 Obtener compromiso de los participantes/interesados acerca de los requerimientos SP 1.3 Administrar cambios a los requerimientos SP 1.4 Mantener la trazabilidad bidireccional de los requerimientos SP 1.5 Identificar inconsistencias entre los requerimientos y otros productos del proyecto

Desde el punto de vista práctico, el área de proceso se satisface poniendo en marcha algún tipo de mecanismo o sistema que:

- Identifique quienes son las fuentes 'oficiales' de requerimientos;
- Identifique cuáles son los canales válidos para ingresar requerimientos al proyecto;
- Controle los cambios (no cualquiera estaría en condiciones de generar un cambio a un requerimiento);
- Permita determinar si todos los involucrados tienen la misma visión respecto al significado de los requerimientos (por ejemplo, mediante la aprobación de algún tipo de documento formal);

- Mantenga la trazabilidad entre los requerimientos y otros artefactos (incluyendo al producto y sus componentes)

Otro ejemplo: para el área Monitoreo y Control del Proyecto (PMC), los objetivos y prácticas específicos son las siguientes:

Objetivos Específicos	Prácticas Específicas
SG 1 Monitorear el Proyecto *El avance y la performance del proyecto se monitorean respecto a lo establecido en el plan de proyecto.*	SP 1.1 Monitorear los parámetros de planificación del proyecto SP 1.2 Monitorear los compromisos SP 1.3 Monitorear los riesgos del proyecto SP 1.4 Monitorear la administración de datos del proyecto SP 1.5 Monitorear la participación de los interesados SP 1.6 Conducir revisiones de avance SP 1.7 Conducir revisiones de cumplimiento de hitos
SG 2 Gestionar Acciones Correctivas *Cuando los resultados o la performance del proyecto se desvian significativamente del plan se gestionan acciones correctivas.*	SP 2.1 Analizar temas pendientes SP 2.2 Ejecutar acciones correctivas SP 2.3 Administrar acciones correctivas

Las areas de proceso definidas para el nivel 3 del modelo, nivel gestionado, son:

- Desarrollo de Requerimientos (RD)
- Solución Técnica (TS)
- Integración del Producto (PI)
- Verificación (VE)
- Validación (VA)
- Enfoque Organizacional en el Proceso (OPF)
- Definición Organizacional del Proceso (OPD)
- Entrenamiento Organizacional (OT)
- Gestión del Riesgo (RSKM)
- Análisis y Toma de Decisiones (DAR)
- Administración Integrada del Proyecto (IPM)
- Gestión Integrada de Proveedores (SS) (ISM)
- Ambiente Organizacional para la Integración (IPPD) (OEI)
- Equipo Integrado (IPPD) (IT)

Para el nivel 2, se definen los siguientes objetivos genéricos (GG), y las siguientes prácticas genéricas (GP) asociadas:

- GG 2 Institucionalizar un proceso administrado
  - GP 2.1 Establecer políticas organizacionales
  - GP 2.2 Planificar el proceso
  - GP 2.3 Proveer Recursos
  - GP 2.4 Asignar responsabilidades
  - GP 2.5 Entrenar al personal
  - GP 2.6 Administrar la configuración
  - GP 2.7 Identificar e involucrar a los interesados
  - GP 2.8 Monitorear y controlar los procesos
  - GP 2.9 Evaluar adhesión objetivamente

    - GP 2.10 Revisar el estado con la alta gerencia
- GG 3 Institucionalizar un proceso definido
    - GP 3.1 Establecer un proceso definido
    - GP 3.2 Recolectar información para mejoras

Por ejemplo, en el área Desarrollo de Requerimientos (RD), los objetivos y prácticas específicos son las siguientes:

Objetivos Específicos	Prácticas Específicas
SG1 Desarrollar Requerimientos del Cliente Se relevan las necesidades, expectativas, restricciones e interfaces y se traducen en requerimientos del cliente.	SP 1.1 Relevar Necesidades SP 1.2 Desarrollar los Requerimientos del Cliente
SG2 Desarrollar los Requerimientos del Producto Los requerimientos del cliente son refinados y elaborados para obtener los requerimientos del producto y sus componentes.	SP 2.1 Establecer Requerimientos del Producto y sus Componentes SP 2.2 Asignar Requerimientos a las Componentes del Producto SP 2.3 Identificar Requerimientos de Interfaz
SG3 Analizar y Validar Requerimientos Los requerimientos son analizados y validados, y se desarrolla una definición de la funcionalidad requerida.	SP 3.1 Desarrollar Concepto de Operación y Escenarios SP 3.2 Desarrollar una Definición de la Funcionalidad Requerida SP 3.3 Analizar Requerimientos SP 3.4 Analizar Requerimientos para Balancear Necesidades y Restricciones SP 3.5 Validar Requerimientos

Otro ejemplo: para el área Gestión de Riesgos (RSKM) los objetivos y prácticas específicos son las siguientes:

Objetivos Específicos	Prácticas Específicas
SG1 Preparar la Gestión del Riesgo Se establece y mantiene una estrategia para identificar, analizar y mitigar riesgos.	SP 1.1 Determinar Fuentes y Categorías de Riesgo SP 1.2 Definir Parámetros de Riesgo SP 1.3 Establecer una Estrategia para la Gestión del Riesgo
SG2 Identificar y Analizar Riesgos Los riesgos son identificados y analizados para determinar su importancia relativa.	SP 2.1 Identificar Riesgos SP 2.2 Evaluar, Categorizar y Priorizar Riesgos
SG3 Mitigar Riesgos Los riesgos son manejados y mitigados para reducir su impacto negativo en los objetivos.	SP 3.1 Desarrollar Planes de Mitigación de Riesgo SP 3.2 Implementar Planes de Mitigación de Riesgo

5. CALIDAD DEL SOFTWARE

CONCEPTOS

- Propiedad o conjunto de propiedades inherentes a una persona cosa que permiten diferenciarla con respecto a las que restan de su especie, como de mejor o peor calidad.
- Conjunto de características de un proceso o servicio que le confieren su aptitud para satisfacer las necesidades expresadas eimplícitas.
- Conjunto de propiedades y de características de un producto o servicio, que le confieren aptitud para satisfacer las necesidades explícitas o implícitas.

La calidad es un concepto complejo y de facetas múltiples que puede describirse desde cinco diferentes puntos de vista:

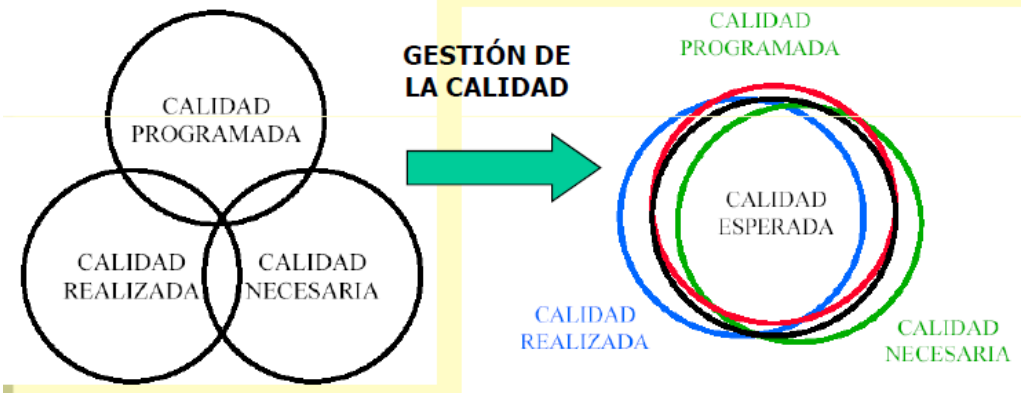
- El trascendental dice que la calidad es algo que se reconoce de inmediato, pero que no es posible definir explícitamente.
- El del usuario concibe la calidad en términos de las metas específicas del usuario final. Si un producto las satisface, tiene calidad.
- El del fabricante la define en términos de las especificaciones originales del producto. Si éste las cumple, tiene calidad.
- El del producto sugiere que la calidad tiene que ver con las características inherentes de un producto.
- El basado en el valor la mide de acuerdo con lo que un cliente está dispuesto a pagar por un producto.

Características

- No es absoluta
- Esta sujeto a restricciones
- Trata de compromisos aceptables
- Los criterios de la calidad no son independientes
- La calidad está en permanente evolución
- Es multidimensional

La calidad puede incluir:

- *Conformidad con las especificaciones*
  - Control del proceso
  - Productos Industriales “idénticos”
  - Conformidad con los requisitos y la confiabilidad en el funcionamiento
  - Cero defectos
  - Adecuación del producto al uso
- *Satisfacción de las expectativas del cliente*
  - El mercado como eje de producto y servicios
  - Dificultad para mediar las expectativas
- *Valor por dinero*
  - Mercado basado en Precio y Calidad
  - Preferencias del consumidor
- *Excelencia*
  - Aplicable a: Productos, Servicios, procesos, Empresas
  - Lo mejor posible



- *Calidad:* "Conjunto de propiedades y características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades explícitas o implícitas"
- *Control de calidad:* "Conjunto de técnicas y actividades de carácter operativo, utilizadas para verificar los requerimientos relativos a la calidad del producto o servicio".
- *Garantía de calidad:* "Conjunto de acciones planificadas y sistemáticas necesarias para proporcionar la confianza adecuada de que un producto o servicio satisfará los requerimientos dados sobre calidad".

Situación Actual

- El producto (software) es algo intangible y no regido por las leyes físicas
- La disciplina, ingeniería del software, es relativamente reciente y muchos de sus conceptos importantes están aún inmaduros
- Carencia de un corpus de conocimiento aceptado mayoritariamente que sirva como fundamentos

En una organización inmadura:

- Procesos software normalmente improvisados. Si se han especificado, no se siguen rigurosamente
- Organización reactiva (resolver crisis inmediatas)
- Planes y presupuestos excedidos sistemáticamente
- Si hay plazos rígidos, se sacrifican funcionalidad y calidad del producto para satisfacer el plan
- No existen bases objetivas para juzgar la calidad del producto
- Cuando los proyectos está fuera de plan, las revisiones o pruebas se recortan o eliminan

CALIDAD DE SOFTWARE

Montenegro, Micaela Soledad

Conjunto de característica de un entidad (producto o servicio) que le confieren su aptitud para satisfacer necesidades expresadas e implícitas.

Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan. Existen tres puntos importantes:

1. Un proceso eficaz de software establece la infraestructura que da apoyo a cualquier esfuerzo de elaboración de un producto de software de alta calidad.

2. Un producto útil entrega contenido, funciones y características que el usuario final desea; sin embargo, de igual importancia es que entrega estos activos en forma confiable y libre de errores. Un producto útil siempre satisface los requerimientos establecidos en forma explícita por los participantes. Además, satisface el conjunto de requerimientos con los que se espera que cuente el software de alta calidad.

3. Al agregar valor para el productor y para el usuario de un producto, el software de alta calidad proporciona beneficios a la organización que lo produce y a la comunidad de usuarios finales.

Principios básicos de la Calidad de Software

- Debe ser construida durante análisis y diseño, no únicamente mediante la realización de revisiones y pruebas
- Solo se alcanza con la contribución de todas las personas involucradas
- Debe ser planificada y gestionada con eficacia
- Dirigir esfuerzos a prevención de defectos
- Reforzar los sistemas de detección y eliminación de defectos durante las primeras fases
- Es un parámetro importante del proyecto al mismo nivel que los plazos de entrega, costos y productividad
- Es esencial la participación de la dirección, que ha de propiciar la calidad

Estándares y modelos de evaluación y mejora de los procesos software

La garantía de calidad es el proceso que define cómo lograr calidad del software y cómo la organización de desarrollo conoce el nivel de calidad requerido en el software. El proceso de QA se ocupa ante todo de definir o seleccionar los estandares que deben ser aplicados al proceso de desarrollo de software o al producto de software.

Podemos definir dos tipos de estandatres como parte del proceso de garantía de calidad:

- Estandares de producto: se aplican sobre el producto de software a desarrollar. Ej. : de documentación, de definición de clases, de codificación.
- Estandares de proceso: definen los procesos que deben seguirse durante el desarrollo de software. Ej. : definición en los procesos de especificación, diseño y validación.

Los estandares son importantes por varias razones:

- Estan basadas en el conocimiento de la mejor o más apropiada práctica de la empresa.
- Proveen un marco de trabajo alrededor del cual se implementa el proceso de garantía de calidad.
- Ayudan a la continuidad cuando una persona continua el traajo que llevaba a cabo otra.

Algunos estándares son:

- ISO 9000 (ISO 9001:2000)
- (SPICE) ISO/IEC 15504
- CMM - CMMI
- Certificación. Organismos

CALIDAD EN EL PRODUCTO

El objetivo no es necesariamente alcanzar una calidad perfecta, sino la necesaria y suficiente para cada contexto de uso a la hora de la entrega y del uso por parte de los usuarios.

Es necesario comprender las necesidades reales de los usuarios con tanto detalle como sea posible.

Calidad del Software como producto:

- ¿Hace lo que el usuario necesita?
- ¿Es lo que el usuario quiere?
- ¿Le soluciona el problema?
- ¿Lo hace como el quiere?
- ¿Se puede construir?
- ¿Es fácil de modificar, de corregir, de extender?
- ¿Se lo puede hacer en tiempo y forma, con costos bajos?
- ¿Me gusta? ¿Le gusta al usuario?

Hay un vínculo entre la calidad del proceso y del producto en producción debido a que el proceso es relativamente fácil de estandarizar y monitorizar. El desarrollo de software es un proceso más creativo que mecánic donde la experiencia y habilidades individuales son importantes. La calidad del producto tambien se ve afectada por factores externos (novedad, presión comercial). En el desarrollo del software la relaion entre calidad del proceso y del producto es muy complejo.

ATRIBUTOS DE CALIDAD

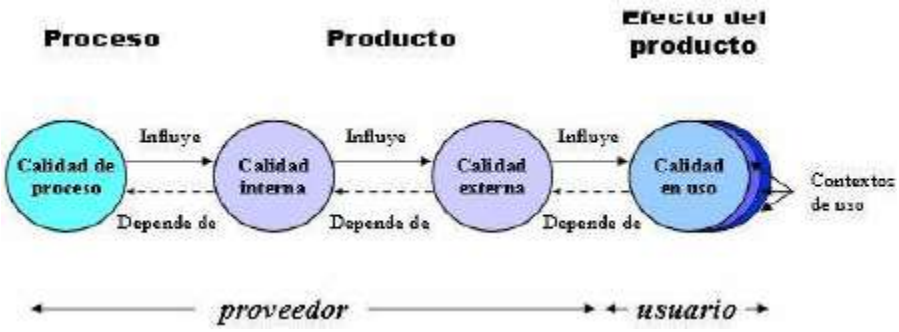
Los atributos de calidad son propiedades o características del sistema, que pueden afectar el grado de satisfacción de los interesados. Es decir, es el grado de concordancia entre las necesidades y el producto final.

Bass et al. (1998) establece una clasificación de los atributos de calidad en dos categorías:

- Observables vía ejecución (Externos): aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. Aquí se encuentra la: Disponibilidad, Confidencialidad, Funcionalidad, Desempeño, Confiabildiad, Seguridad Externa (Safety), Seguridad Interna (Security)
- No observables vía ejecución (Internos): aquellos atributos que se establecen durante el desarrollo del sistema. Aquí se encuentran la: Configurabilidad, Integrabilidad, Integridad, Interoperabilidad, Modificabilidad, Mantenibilidad, Portabilidad, Reusabilidad, Escalabilidad, Capacidad de Prueba (Testability).

Existen diferentes aspectos de la calidad:

- Interna: medible a partir de las características intrínsecas, como el código fuente
- Externa: medible en el comportamiento del producto, como en una prueba
- En uso: durante la utilización efectiva por parte del usuario



- Calidad del producto: Correctitud, usabilidad, mantenibilidad, confiabilidad, rendimiento, disponibilidad, robustez, performance, amigabilidad, reusabilidad, portabilidad etc.
- Calidad del proceso: El proceso debe estar definido, documentado y debe ser practicado y medido

- Criterios de Calidad Es necesario establecer criterios para medir y evaluar la calidad del producto y del proceso.

MODELOS DE CALIDAD

Los modelos de calidad tratan de poner en práctica el concepto de calidad. La calidad del software se puede describir de manera jerárquica

Modelo de McCall

El modelo de McCall fue el primero en ser presentado en 1977, se originó motivado por US Air Force y DoD. Este se focaliza en el producto final, identificando atributos claves desde el punto de vista del usuario. Estos atributos se denominan factores de calidad y son normalmente atributos externos, pero también se incluyen algunos atributos posiblemente internos. Los factores de calidad son demasiados abstractos para ser medidos directamente, por lo que por cada uno de ellos se introduce atributos de bajo nivel denominados criterios de calidad.

Los factores que afectan la calidad del software se centran en tres aspectos importantes del producto de software: sus características operativas, su capacidad de ser modificado y su adaptabilidad a nuevos ambientes, es decir se centran en tres aspectos importantes de un producto software (McCall):

- revisión del producto: habilidad para ser cambiado
- transición del producto: adaptabilidad al nuevo ambiente
- operación del producto: características de operación

- Características operativas
  - Corrección. ¿Hace lo que quiero?
  - Fiabilidad. ¿Lo hace de forma fiable todo el tiempo?
  - Eficiencia. ¿Se ejecutará en mi hardware lo mejor que pueda?
  - Seguridad (Integridad). ¿Es seguro?
  - Facilidad de uso. ¿Está diseñado para ser usado?
- Capacidad de soportar los cambios
  - Facilidad de mantenimiento. ¿Puedo corregirlo?
  - Flexibilidad. ¿Puedo cambiarlo?
  - Facilidad de prueba. ¿Puedo probarlo?
- Adaptabilidad a nuevos entornos

- Portabilidad. ¿Podré usarlo en otra máquina?
- Reusabilidad. ¿Podré reutilizar alguna parte del software?
- Interoperabilidad. ¿Podré hacerlo interactuar con otro sistema?

La revisión del producto incluye los siguientes factores de calidad:

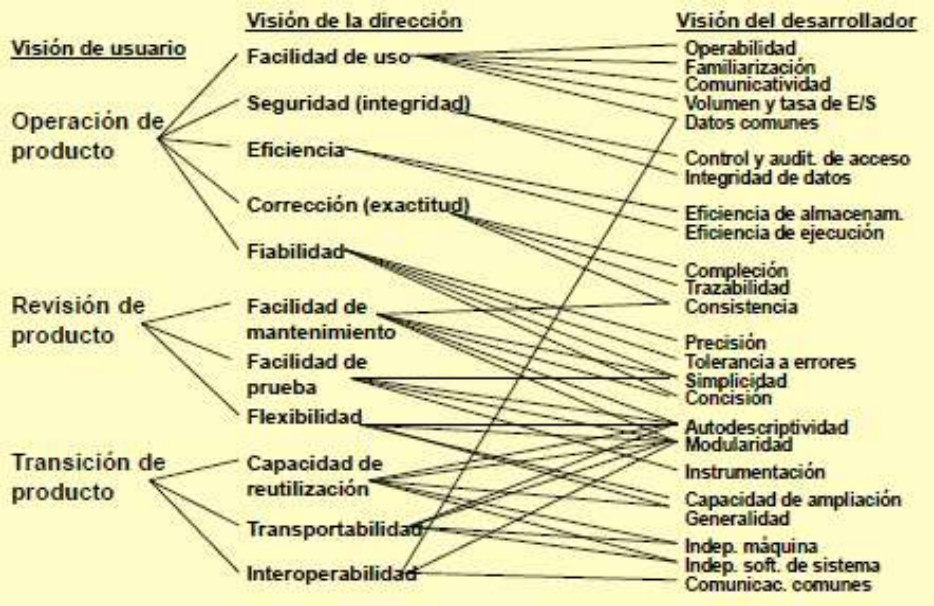
- Mantenibilidad: esfuerzo requerido para localizar y corregir fallas. El factor mantenibilidad incluye los siguientes criterios: consistencia, simplicidad, concisidad, auto-descripción y modularidad.
  A la mantenibilidad se la puede dividir en tres categorías:
  - Correctiva: concerniente a remover pequeñas fallas remanentes después del testeo
  - Adaptativa: concerniente al cambio del producto necesario por el cambio de sus requerimientos
  - Perfectiva: busca solo mejorar los algoritmos usados para hacerlos más eficientes
- Flexibilidad: facilidad de realizar cambios. Esfuerzo necesario para modificar un programa que ya opera.
- Testeabilidad: facilidad para realizar el testing, para asegurarse que el producto no tiene errores y cumple con la especificación. Esfuerzo que se requiere para probar un programa a fin de garantizar que realiza la función que se pretende.

La transición del producto incluye los siguientes factores de calidad:

- Portabilidad: esfuerzo requerido para transferir entre distintos ambientes de operación
- Reusabilidad: facilidad de reusar el software en diferentes contextos
- Interoperabilidad: esfuerzo requerido para acoplar el producto con otros sistemas

La operación del producto incluye los siguientes factores de calidad:

- Correctitud: el grado en el que el producto cumple con su especificación
- Confiabilidad: la habilidad del producto de responder ante situaciones no esperadas. el factor confiabilidad incluye los siguientes criterios: tolerancia a errores, consistencia, simplicidad y exactitud.
  Combina la tolerancia tanto a errores de hardware como de software. Puede ser medido con medidas como:
  - Tiempo medio entre fallas
  - Tiempo medio antes de mantenimiento
  - Tiempo medio antes de recuperación
  - Probabilidad de falla
- Eficiencia: el uso de los recursos tales como tiempo de ejecución y memoria de ejecución
- Integridad: protección del programa y sus datos de accesos no autorizados
- Usabilidad: facilidad de operación del producto por parte de los usuarios. El factor usabilidad incluye los siguientes criterios: operabilidad, entrenamiento, comunicación, volumen de E/S y tasa de E/S
  La usabilidad se puede subdividir en:
  - Ergonomía general: el equipo es adecuado para el uso previsto
  - Ergonomía de software: estilos de diálogos, metáforas, diseño de pantallas, etc



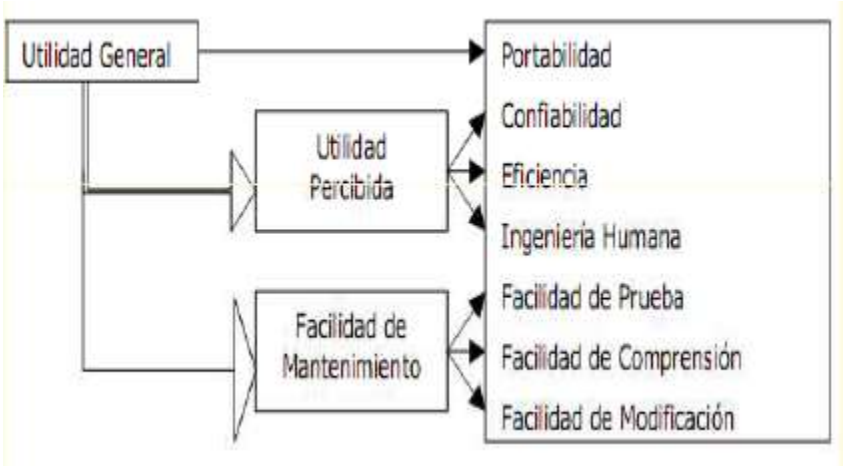
Modelo de Boehm

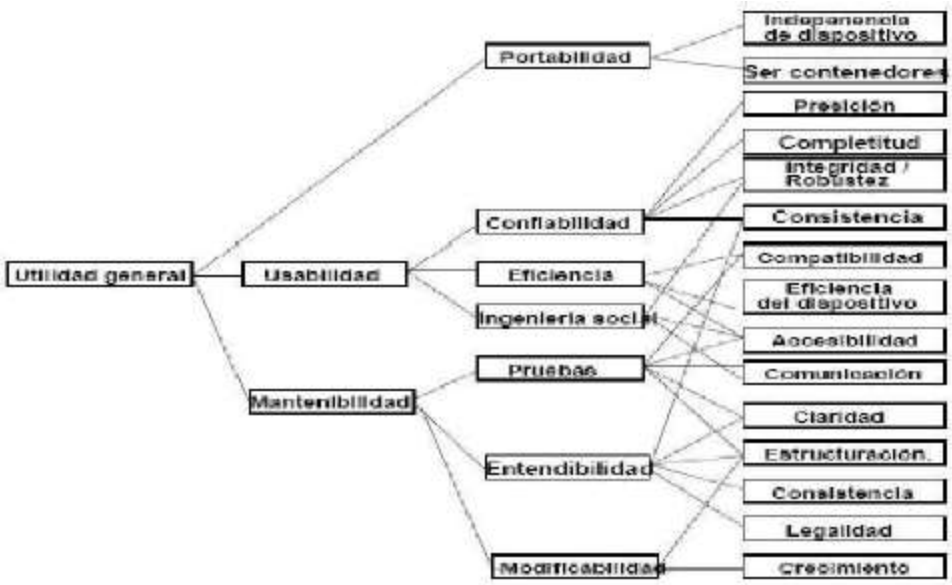
El modelo que presenta Boehm presenta una jerarquía de características donde cada una de ellas contribuye a la calidad global. Se centra en:

- Sus características operativas.
- Su capacidad para soportar los cambios.
- Su adaptabilidad a nuevos entornos.
- La evaluación del desempeño del hardware.

El modelo comienza con la utilidad general del software, afirmando que el software es útil, evitando pérdida de tiempo y dinero. La utilidad puede considerarse en correspondencia a los tipos de usuarios que quedan involucrados.

- El primer tipo de usuario queda satisfecha si el sistema hace lo que el pretende que haga;
- El segundo tipo es aquel que utiliza el sistema luego de una actualización
- El tercero, es el programador que mantiene el sistema.





Características de alto nivel representan requerimientos generales de uso. Pueden ser:

- utilidad per-se cuan (usable, confiable, eficiente) es el producto en sí mismo
- mantenibilidad cuan fácil es modificarlo, entenderlos y retestearlo
- utilidad general si puede seguir usándose si se cambia el ambiente

Características de nivel intermedio representan los factores de calidad:

- portabilidad
- confiabilidad
- eficiencia
- usabilidad
- testeabilidad
- facilidad de entendimiento
- modificabilidad o flexibilidad

Características Primitivas: el nivel más bajo corresponde a características directamente asociadas a una o dos métricas de calidad

- de portabilidad
  - independencia de dispositivos
- de confiabilidad:
  - exactitud
  - completitud
  - consistencia
  - robustez/integridad

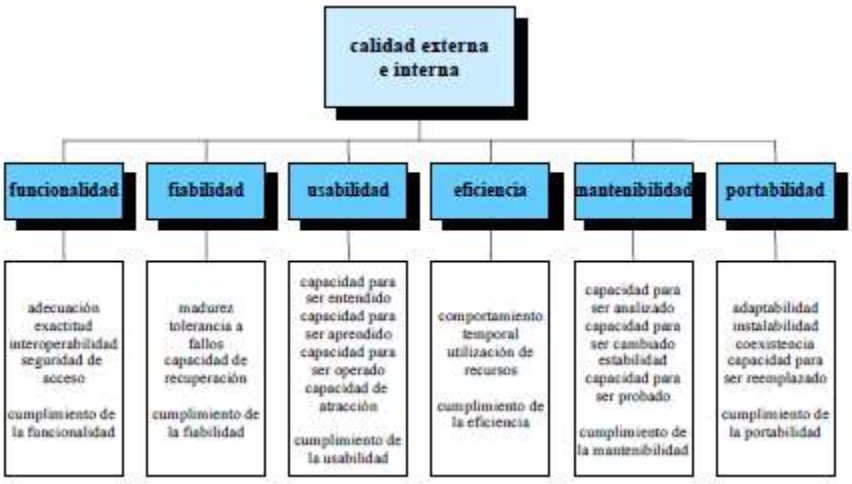
<u>ISO 9126</u>

- Parte 1: Modelo de Calidad
- Parte 2: Métricas Externas
- Parte 3: Métricas Internas
- Parte 4: Métricas de Calidad en Uso

Ejemplos de uso:

- Validar la compleción de una definición de requerimientos
- Identificar requerimientos software

- Identificar objetivos para el diseño software
- Identificar requerimientos para las pruebas del software
- Identificar requerimientos para el aseguramiento de la calidad
- Identificar criterios de aceptación para un producto software terminado



El estándar ISO 9126 se desarrolló con la intención de identificar los atributos clave del software de cómputo. Este sistema identifica seis atributos clave de la calidad:

- Funcionalidad. Grado en el que el software satisface las necesidades planteadas según las establecen los atributos siguientes:
  - Adecuación: Capacidad del producto software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.
  - Exactitud: Capacidad del producto software para proporcionar los resultados o efectos correctos o acordados, con el grado necesario de precisión.
  - Interoperabilidad: Capacidad del producto software para interactuar con uno o más sistemas especificados.
  - Seguridad de acceso: Capacidad del producto software para proteger información y datos de manera que las personas o sistemas no autorizados no puedan leerlos o modificarlos, al tiempo que no se deniega el acceso a las personas o sistemas autorizados
  - Cumplimiento funcional: Capacidad del producto software para adherirse a normas, convenciones o regulaciones en leyes y prescripciones similares relacionadas con funcionalidad.
- Confiabilidad. Cantidad de tiempo que el software se encuentra disponible para su uso, según lo indican los siguientes atributos:
  - Madurez: Capacidad del producto software para evitar fallar como resultado de fallos en el software.
  - Tolerancia a fallos: Capacidad del software para mantener un nivel especificado de prestaciones en caso de fallos software o de infringir sus interfaces especificados.
  - Capacidad de recuperación: Capacidad del producto software para reestablecer un nivel de prestaciones especificado y de recuperar los datos directamente afectados en caso de fallo.
  - Cumplimiento de la fiabilidad: Capacidad del producto software para adherirse a normas, convenciones o regulaciones relacionadas con al fiabilidad.
- Usabilidad. Grado en el que el software es fácil de usar, según lo indican los siguientes subatributos:
  - Capacidad para ser entendido: Capacidad del producto software que permite al usuario entender si el software es adecuado y cómo puede ser usado para unas tareas o condiciones de uso particulares.
  - Capacidad para ser aprendido: Capacidad del producto software que permite al usuario aprender sobre su aplicación.
  - Capacidad para ser operado: Capacidad del producto software que permite al usuario operarlo y controlarlo.

    - Capacidad de atracción: Capacidad del producto software para ser atractivo al usuario.
    - Cumplimiento de la usabilidad: Capacidad del producto software para adherirse a normas, convenciones, guías de estilo o regulaciones relacionadas con la usabilidad.
- Eficiencia. Grado en el que el software emplea óptimamente los recursos del sistema, según lo indican los subatributos siguientes:
    - Comportamiento temporal: Capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas.
    - Utilización de recursos: Capacidad del producto software para usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas.
    - Cumplimiento de la eficiencia: Capacidad del producto software para adherirse a normas o convenciones relacionadas con la eficiencia.
- Facilidad de recibir mantenimiento. Facilidad con la que pueden efectuarse reparaciones al software, según lo indican los atributos que siguen:
    - Capacidad para ser analizado: Es la capacidad del producto software para serle diagnosticadas deficiencias o causas de los fallos en el software, o para identificar las partes que han de ser modificadas.
    - Capacidad para ser cambiado: Capacidad del producto software que permite que una determinada modificación sea implementada.
    - Estabilidad: Capacidad del producto software para evitar efectos inesperados debidos a modificaciones del software.
    - Capacidad para ser probado: Capacidad del producto software que permite que el software modificado sea validado.
    - Cumplimiento de la mantenibilidad: Capacidad del producto software para adherirse a normas o convenciones relacionadas con la mantenibilidad.
- Portabilidad. Facilidad con la que el software puede llevarse de un ambiente a otro según lo indican los siguientes atributos: adaptable, instalable, conformidad y sustituible.

ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE SQA

Es el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto (software) satisfará los requerimientos dados de calidad. El aseguramiento de calidad del software se diseña para cada aplicación antes de comenzar a desarrollarla y no después.

Es una serie de actividades que tienen que ser implementadas a través del proceso de desarrollo de software que incluyen actividades para asegurar alta calidad de producto, pruebas de aseguramiento de calidad y métricas para desarrollar estrategias que mejorarán el proceso de software.

Función

Mejorar la calidad de los procesos de desarrollo y mantenimiento del software, monitoreando, durante el transcurso del ciclo de vida de los diferentes proyectos, el cumplimiento de los estándares y procesos establecidos, antes de su puesta en productivo.

Actividades

- Establecer planes, estándares y procesos que satisfagan las políticas de la organización y se ajusten a las necesidades de cada proyecto en particular.
- Revisar y auditar los productos y actividades desarrolladas para verificar que ellos satisfacen los procesos y estándares definidos.
- Proveer las herramientas necesarias que den soporte al proceso definido, facilitando su ejecución, visualización, administración y seguimiento;
- Proveer al equipo de proyecto y a otros interesados, los resultados sobre las revisiones, auditorías y actividades;
- Escalar problemas no resueltos dentro del equipo de un proyecto hacia un nivel apropiado de administración para su resolución.

También incluye: Métricas de software para el control del proyecto, Verificación y validación del software a lo largo del ciclo de vida lo cual incluye las pruebas y los procesos de revisión e inspección y Gestión de la configuración del software.

Plan de Calidad

Mapa para institucionalizar la garantía de calidad del software. Es una plantilla para definir las actividades de SQA aplicables a cada proyecto de software. El plan incluye:

- Sección Gestión: Tareas y actividades de SQA dentro del proceso de software y los roles y responsabilidades relativas a la calidad del producto.
- Sección Documentación: Detalle de los productos de trabajo del proceso de software que podrán ser revisados.
- Sección Estándares, Prácticas y Convenciones: Detalle de lo que está acordado y establecido para el proceso y los productos a obtener.
- Sección Revisiones y Auditorias: Revisiones que se llevarán a cabo durante el proceso y los responsables de cada una de ellas. (Ejemplos: Revisiones de documentación, revisiones técnico formales (RTF's), etc.)
- Sección de Pruebas: Plan y procedimiento de Pruebas del Software y de gestionar los defectos detectados.
- Sección Métodos y Herramientas que soportan las actividades de SQA

DEFECTO

En el contexto del proceso del software, los términos defecto y falla son sinónimos. Los dos implican un problema de calidad descubierto después de haberse liberado el software a los usuarios finales (o a otra actividad estructural del proceso del software).

- Error: Problema de calidad que se detecta antes de que el software se entregue a los usuarios finales.
- Defecto: problema de calidad que se encuentra después de haber entregado el software a los usuarios finales.
- Falla: Es la discrepancia visible que se produce al ejecutar un programa con un defecto, el cual es incapaz de funcionar correctamente.

Inspeciones de Software

Proceso de Verificación y Validación estático (es diferente a las pruebas) en el que el sistema de software se revisa para encontrar errores, omisiones y anomalias. Generalmente, se centran en el código fuente, pero puede inspecionarse tambióen modelos de diseños, o especificaciones de requerimientos.

Las inspeciones de progrmas son revisiones cuyo es la detección de defectos en el programa.

Métricas para la Inspección:

- Esfuerzo de preparación, Ep: esfuerzo (en horas-hombre) requerido para revisar un producto del trabajo antes de la reunión de revisión real.
- Esfuerzo de evaluación, Ea: esfuerzo requerido (en horas-hombre) que se dedica a la revisión real.
- Esfuerzo de la repetición, Er: esfuerzo (en horas-hombre) que se dedica a la corrección de los errores descubiertos durante la revisión.
- Tamaño del producto del trabajo, TPT: medición del tamaño del producto del trabajo que se ha revisado (por ejemplo, número de modelos UML o número de páginas de documento o de líneas de código).
- Errores menores detectados, Err menores: número de errores detectados que pueden clasificarse como menores.
- Errores mayores detectados, Err mayores: número de errores encontrados que pueden clasificarse como mayores.

El esfuerzo total de inspección y el número total de errores descubiertos se definen como sigue:

E revisión = Ep + Ea + Er

E rr total = Err menores + Err mayores

La densidad del error representa los errores encontrados por unidad de producto del trabajo revisada:

Densidad del error = E rr total / TPT

MEDICIONES DE ATRIBUTOS DE CALIDAD

Es imposible medir los atributos de calidad de software directamente. Loas atributos de calidad como la mantenibilidad, usabilidad son atributos externos que nos dicen cómo ven el software los desarrolladores y los usuarios. Sin embargo, existe una relación entre los atributos internos y externos del software.

Por ejemplo:

- Mantenibilidad está relacionado con el número de parámetros del procedimiento, la complejidad ciclomática, el tamaño del programa en líneas de código y la extensión del manual de usuario.
- Usabilidad está relacionado con número de mensajes de erro y la extensión del manual de usuario.

Las métricas de producto se refierean a las características del software. Éstas estan relacionadas con diversos atributos de calidad. Las métricas dinámicas (recogidas de mediciones hechas en el programa) ayudan a valorar la eficiencia y fiabilidad de un programa. Ejemplo por medio de medio el tiempo de ejecución de una función, el número y tipo de caidas del sistema. Las métricas estáticas (recogidas de mediciones hechas en las representaciones del sistema como diseño, el programa o la documentación) ayudan a valorar la complejidad, comprensión y mantenibilidad de un sistema de software. Ejemplo de estas métricas: Longitud de código, complejidad ciclomática.

6. INGENIERÍA DE REQUERIMIENTOS

REQUERIMIENTOS

Los requerimientos para un sistema son descripciones de lo que el sistema debe hacer: el servicio que ofrece y las restricciones de su operación. Tales requerimientos reflejan las necesidades de los clientes por un sistema que atienda ciertos propósitos. Al proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se le llama Ingeniería de Requerimientos.

Un requerimiento de software puede ser definido como:

- Una capacidad del software necesaria por el usuario para resolver un problema o alcanzar un objetivo.
- Una capacidad del software que debe ser reunida o poseída por un sistema o componente del sistema para satisfacer un contrato, especificación, estándar, u otra documentación formal.

Impacto de los errores en la etapa de requerimientos

- El software resultante puede no satisfacer a los usuarios
- Las interpretaciones múltiples de los requerimientos pueden causar desacuerdos entre clientes y desarrolladores
- Es imposible que a través del testeo el software satisfaga sus requerimientos
- Puede gastarse tiempo y dinero construyendo el sistema erróneo

En conclusión el problema está en entender el sistema.

Rol de los requerimientos

- Acuerdo desarrolladores-clientes-usuarios finales

    - Aspecto contractual
    - Multipartes (comunicación, conflicto y cambio de visiones)
- Base para el diseño del software
    - Minimizar defectos tanto como sea posible
    - Técnicamente factible
- Soporte para verificación y validación
- Soporte a la evolución del sistema
    - Evolución del sistema (cambio de sistema viejo a sistema nuevo y cambio de requerimientos)

Caracterización

El equipo de desarrollo puede organizar a los requerimientos agrupándolos según las características que posean. Como resultado de esta clasificación, la gestión de los mismos se torna más manejable. De las diferentes definiciones y clasificaciones referidas a los requerimientos se observa que difieren principalmente en el grado de abstracción de la declaración de los mismos, como así también desde el punto de vista de los stakeholders que los definen.

Las características más importantes de los requerimientos son:

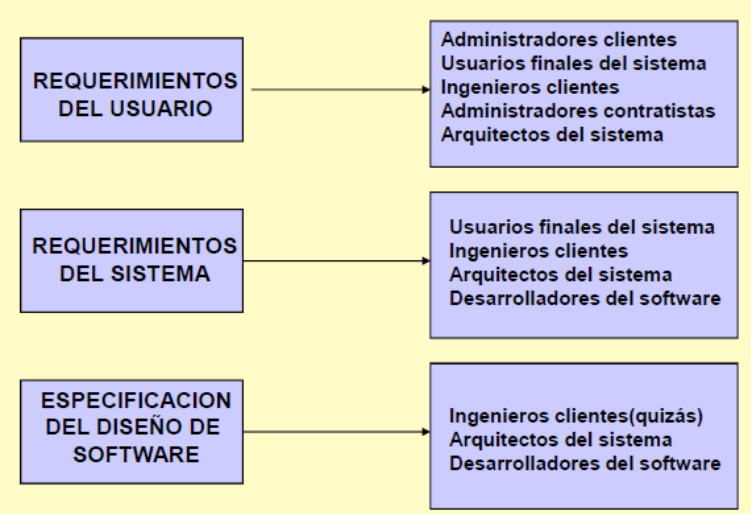
- Evolución: Si los requerimientos se expresan para solucionar algún problema del mundo real, que está en continuo devenir, tales hechos repercutirán en la volatilidad de los mismos
- Situados: los requerimientos se encuentran situados, en tanto que dependen de los detalles de una situación particular o contexto del cual emergen
- Negociables: los requerimientos son negociables en tanto que resultan de las discusiones entre las partes interesadas

Tipos de requerimientos

Existen diferentes niveles de descripción:

- Requerimientos del Usuario: se ocupan para asignar los requerimientos de más alto nivel. Son enunciados en lenguaje natural junto con diagramas, acerca de qué servicios esperan los usuarios del sistema, y las restricciones con las cuales éste debe operar.
- Requerimientos del Sistema: se usa para designar descripción más detallada de lo que el sistema debe hacer, de sus funciones, servicios y restricciones operacionales del sistema de software. El documento de requerimientos del sistema, también llamado Especificación Funcional tiene que definir con exactitud lo que se implementará, y puede formar parte del contrato entre cliente y desarrolladores.
- Requerimientos del Dominio:
    - Se derivan del dominio del sistema más que de las necesidades específicas de los usuarios.
    - Si estos requerimientos no se satisfacen, es imposible hacer que el sistema trabaje de forma satisfactoria.

Estos diferentes niveles de requerimientos son útiles debido a que informan sobre el sistema a distintos tipos de lector. Los lectores de los diferentes tipos de especificaciones son los que se muestran en la imagen:



Los requerimientos del sistema de software se pueden clasificar en:

- Requerimientos funcionales: son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas. No tiene en cuenta cuestiones de implementación.
  La especificación de los requerimientos funcionales de un sistema debe ser completa (deben definirse todos los servicios requeridos por los usuarios) y consistente (los requerimientos tienen que evitar definiciones contradictorias).
- Requerimientos no funcionales: Son limitaciones sobre servicios o funciones que ofrece el sistema. Describen aspectos del sistema visibles por el usuario que no se relacionan en forma directa con el comportamiento funcional del sistema. Los requerimientos no funcionales a menudo son más significativos que los funcionales individuales, y deben tratar de escribirse cuantitativamente para que puedan ser puestos objetivamente a prueba.
  Los requerimientos no funcionales se pueden clasificar en:
  - Requerimientos de producto: Especifican o restringen el comportamiento del software.
  - Requerimientos de negocio: son derivados de políticas y procedimientos en la organización del cliente y del desarrollador.
  - Requerimientos externos: son derivados de factores externos al sistema y su proceso de desarrollo.
- Requerimientos de implementación: son necesidades del cliente que restringen la implementación (por ejemplo, lenguaje de programación, plataforma hardware)

INGENIERÍA DE REQUERIMIENTOS

Proceso de descubrir el propósito por medio de identificar a los stakeholders, sus necesidades y documentarlas en forma tal que se puedan disponer en el análisis, comunicación y las sucesivas implementaciones. Es decir es el proceso de establecer los servicios que el sistema debe proveer y las restricciones bajo las cuales deben operar.

El proceso de RE es sistemático, iterativo y cooperativo, e involucra además actividades de análisis, documentación y chequeo.

Características

- Representación, aspecto social y aspecto cognitivo
- De una formulación informal a una especificación formal
- Proceso no determinístico y no lineal
- Elicitar, especificar y validar requerimientos, no son actividades predominantemente técnicas.
- Típica actividad de resolución de problemas
- Sistemático, iterativo y cooperativo

Producto y Procesos de la Ingeniería de Requerimientos

Elicitar
- Identificar fuentes de conocimiento
- Adquirir conocimiento
- Decidir sobre la relevancia del conocimiento
- Comprender el significación del conocimiento y su impacto

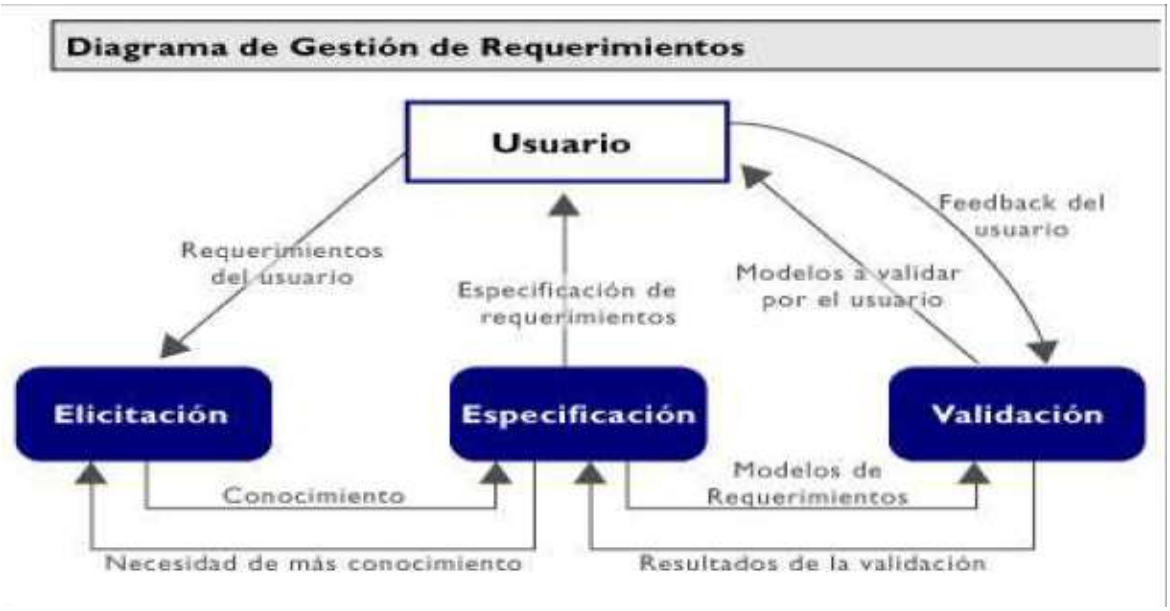
Especificación
- Análisis y asimilación del conocimiento de los requerimientos
- Síntesis y organización del conocimiento en un modelo de requerimientos coherente y lógico

Validación
- Certifica la consistencia del modelo con las intenciones de los clientes y usuarios
- Ayuda a hacer el artefacto correcto

Los procesos de Ingeniería de requerimientos incluyen cuatro actividades de alto nivel. Éstas se enfocan en valorar si el sistema es útil para la empresa (Estudio de factibilidad), descubrir requerimientos (adquisición y análisis), convertir dichos requerimientos en alguna forma estándar (especificación) y comprobar que los requerimientos definan realmente el sistema que quiere el cliente (validación).

Las actividades están organizadas como un proceso iterativo alrededor del espiral, y la salida es un documento de requerimientos del sistema.

Los procesos de la Ingeniería de Requerimientos son:



- **<u>Adquisición y Análisis (Elicitación) de Requerimientos:</u>** Después de un estudio de factibilidad inicial, los ingenieros de software trabajan con clientes y usuarios finales del sistema para descubrir el dominio de la aplicación, qué servicios debe proporcionar el sistema, el desempeño requerido de éste, las restricciones de hardware, etc. Las actividades del proceso son:
  - <u>Descubrimiento de requerimientos</u>: Se descubren los requerimientos de usuario, de dominio y la documentación. Aquí se recopila información sobre el sistema requerido y los sistemas existentes de distintas fuentes (documentación, participantes del sistema, especificaciones de sistemas similares)
  - <u>Clasificación y organización de requerimientos:</u> Agrupa los requerimientos relacionados y los organiza en grupos coherentes.
  - <u>Priorización y negociación de requerimientos</u>: prioriza los requerimientos, y encuentra y resuelve conflictos de requerimientos mediante la negociación.

    - Especificación de requerimientos: los requerimientos se documentan e ingresan a la siguiente ronda de la espiral. Pueden producirse documentos formales o informales.
- **Especificación de Requerimientos:** Proceso de escribir, en un documento de requerimientos, los requerimientos de usuario y del sistema. De manera ideal, los requerimientos del usuario y del sistema deben ser claros, sin ambigüedades, fáciles de entender, completos y consistentes.
- **Validación de requerimientos:** Proceso de verificar que los requerimientos definan realmente el sistema que en verdad quiere el cliente, esto es importante porque los errores en un documento de requerimientos pueden conducir a grandes costos por tener que rehacer, cuando dichos problemas se descubren durante el desarrollo del sistema o después de que éste se halla puesto en servicio.
  Durante el proceso de validación se realizan diferentes tipos de comprobaciones sobre los requerimientos contenidos en el documento de requerimientos: comprobaciones de validez (permite identificar funciones adicionales que se requieran), de consistencia (no se deben contradecir), de totalidad (que contenga toda las funciones y restricciones pretendidas por el cliente), de realismo (realmente deben poder implementarse), verificabilidad (deben poder verificarse).
  Algunas de las técnicas de validación son:
    - Revisiones de requerimientos: éstos se analizan sistemáticamente usando un equipo de revisores que verifican errores e inconsistencias.
    - Creación de Prototipos: se muestra un modelo ejecutable del sistema en cuestión a los usuarios y clientes, para que estos experimenten con él y podes así saber si cubre o no sus necesidades.
    - Generación de casos de prueba: Los requerimientos deben ser comprobables. Si las pruebas de requerimientos de realizan como parte del proceso de validación esto revela problemas en los requerimientos.

GESTIÓN DE REQUERIMIENTOS

Los requerimientos para los grandes sistemas de software siempre cambian. Existen muchas razones por las que es inevitable el cambio:

- El ambiente empresarial y técnico del sistema siempre cambian después de la instalación.
- Los individuos que pagan por un sistema y los usuarios de dichos sistema, por lo general no son lo mismo.
- Los sistemas grandes tienen regularmente una comunidad de usuarios diversa, en la cual muchos individuos tienen diferentes requerimientos y prioridades que quizás estén en conflictos o sean contradictorios.

El proceso de comprender y controlar los cambios en los requerimientos del sistema se denomina gestión de requerimiento.

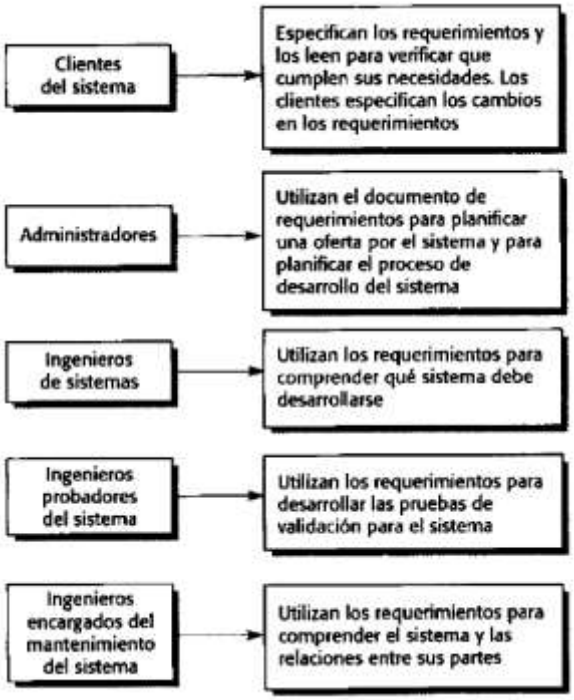
- **Planeación de la administración de requerimientos:** Es la primera etapa en el proceso de gestión de requerimiento. Esta etapa establece el nivel de detalle que se requiere en la administración de requerimientos. Decide sobre:
    - Identificación de requerimientos.
    - Proceso de administración del cambio: conjunto de actividades que valoran el efecto y el costo de los cambios.
    - Políticas de seguimiento: definen las relaciones entre cada requerimiento así como entre los requerimientos y el diseño de sistema que debe registrarse.
    - Herramientas de apoyo.

  Se necesitan herramientas de apoyo para el almacenamiento de requerimiento, la administración del cambio y la administración del seguimiento.
- **Administración del cambio en los requerimientos:** La administración del cambio en los requerimientos deben aplicarse a todos los cambios propuestos a los requerimientos de un sistema, después de aprobarse el documento de requerimientos. La administración del cambio es esencial porque es necesario determinar si los beneficios de implementar nuevos requerimientos están justificados por los costos de la implementación. Etapas:

- Análisis del problema y especificación del cambio: el problema o la propuesta de cambio se analizan para comprobar que es válida. Esto retroalimenta al solicitante del cambio, quien responderá con una propuesta de cambio de requerimiento más específica o decidirá retirar la petición.
- Análisis del cambio y estimación del costo: el efecto del cambio propuesto se valora usando información de seguimiento y conocimiento general del requerimiento del sistema. El costo por realizar el cambio se estima en términos de modificaciones al documento de requerimientos y al diseño y la implementación del sistema. Luego se toma la decisión acerca de si se procede o no con el cambio del requerimiento.
- Implementación del cambio: se modifica el documento de requerimientos y, donde sea necesario, y el diseño y la implementación del sistema.

DOCUMENTOS DE ESPECIFICACIÓN DE REQUERIMIENTOS. STD 830

Comunicado oficial de lo que deben implementar los desarrolladores del sistema. Incluye tanto los requerimientos del usuario para un sistema, como una especificación detallada de los requerimientos del sistema. El documento de requerimientos tiene un conjunto variado de usuarios, desde el administrador que paga por el sistema hasta los ingenieros responsables de su desarrollo.



La diversidad de posibles usuarios significa que el documento de requerimientos debe ser un compromiso entre la comunicación de los requerimientos a los clientes, la definición de los requerimientos con detalle preciso para los desarrolladores y examinadores, y la inclusión de información sobre la posible evolución del sistema. El nivel de detalle que se incluya en un documento de requerimientos depende del tipo de sistema a diseñar y el proceso de desarrollo utilizado. La información que se incluye en el documento de requerimiento depende del tipo de software que se vaya a desarrollar y del enfoque para el desarrollo que se use.

7. MÉTRICAS DE PROCESO Y PROYECTO

La medición permite ganar comprensión acerca del proceso y del proyecto, al proporcionar un mecanismo de evaluación.

La medición puede aplicarse al proceso de software con la intención de mejorarlo de manera continua. Puede usarse a través de un proyecto de software para auxiliar en estimación, control de calidad, valoración de productividad y control de proyecto. Finalmente, la medición pueden usarla los ingenieros del software para

ayudar en la valoración de la calidad de los productos de trabajo y auxiliar en la toma de decisiones tácticas conforme avanza un proyecto.

Métrica: Las métricas de proceso y proyecto de software son medidas cuantitativas que permiten obtener comprensión acerca de la eficacia del proceso del software y de los proyectos que se realizan, usando el proceso como marco conceptual.

Las razones por las que se mide son:

- Para caracterizar un esfuerzo y obtener comprensión de los procesos, productos, recursos y entornos, y establecer líneas de referencia para comparar con valoraciones futuras.
- Para evaluar y "determinar el estado de avance con respecto a los planes.
- Para predecir al obtener comprensión de las relaciones entre procesos y productos, y construir modelos de dichas relaciones
- Para mejorar al identificar barricadas, causas raíz, ineficiencias y otras oportunidades para mejorar la calidad del producto y el desempeño del proceso

MÉTRICAS EN DOMINICOS DEL PROCESO Y PROYECTO

Las métricas de proceso se recopilan a través de todos los proyectos y durante largos espacios de tiempo. Su intención es proporcionar un conjunto de indicadores de proceso que conduzca a mejorar el proceso de software a largo plazo. Las métricas de proyecto permiten al gerente de un proyecto de software:

- Valorar el estado de un proyecto en marcha
- Rastrear riesgos potenciales
- Descubrir áreas problema antes de que se vuelvan "críticas
- Ajustar el flujo de trabajo o las tareas
- Evaluar la habilidad del equipo del proyecto para controlar la calidad de los productos operativos del software.

Las métricas del proceso y la mejora del proceso de software

La única forma racional para mejorar cualquier proceso es medir atributos específicos del mismo, desarrollar un conjunto de métricas significativas con base en dichos atributos y luego usarlas para proporcionar indicadores que conducirán a una estrategia para mejorar.

Es importante observar que el proceso sólo es uno de varios "factores controlables en el mejoramiento de la calidad del software y del desempeño organizativo"

El proceso se asienta en el centro de un triángulo que conecta tres factores que tienen profunda influencia sobre la calidad del software y en el desempeño de la organización. La habilidad y motivación del personal ha demostrado ser el factor individual más influyente en la calidad y el desempeño. La complejidad del producto puede tener un impacto sustancial sobre la calidad y el desempeño del equipo. La tecnología que puebla el proceso también tiene un impacto.

Además, existe el triángulo de proceso dentro de un círculo de condiciones ambientales que incluyen entorno de desarrollo, condiciones empresariales (fechas límite, reglas empresariales) y características del cliente (facilidad de comunicación y colaboración).

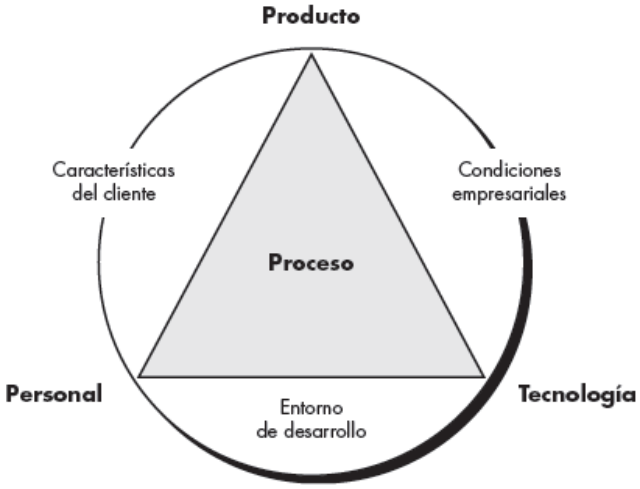
Las métricas pueden ser:

- Privadas. Incluyen índices de defecto por individuo, índice de defecto por componente de software y errores encontrados durante el desarrollo. Los datos de proceso privados pueden funcionar como un importante promotor para que el trabajo individual del ingeniero mejore.

- Intermedias. Algunas métricas de proceso son privadas para el equipo de proyecto, pero públicas para todos los miembros del equipo. Por ejemplo los defectos que reportan las grandes funciones de software, errores detectados

durante las revisiones técnicas formales y líneas de código o puntos de función por módulo o por función. Dichos datos los revisa el equipo para descubrir indicadores que mejoren su desempeño.

- Públicas. Por lo general asimilan información que originalmente era privada para los individuos y equipos. Los índices de defecto al nivel de proyecto, esfuerzo, planificación y datos relacionados se recopilan y evalúan con la finalidad de descubrir indicadores que pueden mejorar el desempeño organizacional.



La eficacia de un proceso de software sólo puede medirse de manera indirecta. Esto significa que es posible derivar un conjunto de métricas con base en los resultados que pueden derivarse del proceso. Los resultados incluyen medidas de los errores descubiertos antes de liberar el software, defectos entregados a y reportados por usuarios finales, productos operativos entregados (productividad), esfuerzo humano empleado, tiempo calendario consumido, conformidad con la agenda y otras medidas. También pueden derivarse métricas de proceso al medir las características de tareas de ingeniería de software específicas.

Las métricas de proceso de software pueden proporcionar beneficios significativos conforme una organización trabaja para mejorar su nivel global de madurez de proceso.

Conforme una organización se siente más cómoda con la recolección y uso de métricas de proceso, la derivación de los indicadores simples da lugar a un enfoque más riguroso llamado mejora estadística de proceso de software (MEPS). En esencia, MEPS usa análisis de falla del software para recopilar información acerca de todos los errores y defectos que se encuentren conforme se desarrolle y use una aplicación, sistema o producto.

Métricas de proyecto

Las medidas de proyecto de software son tácticas. Es decir, el gerente de proyecto y un equipo de software usan las métricas de proyecto y los indicadores derivados de ellas para adaptar el flujo de trabajo del proyecto y las actividades técnicas.

La primera aplicación de las métricas de proyecto sobre la mayoría de los proyectos de software ocurre durante la estimación. Las métricas recopiladas de proyectos anteriores se usan como la base desde la cual se hacen estimaciones de esfuerzo y tiempo para el trabajo de software nuevo. Conforme avanza un proyecto, las medidas de esfuerzo y tiempo calendario utilizadas se comparan con las estimaciones originales. El gerente del proyecto usa dichos datos para monitorear y controlar el progreso.

Mientras comienza el trabajo técnico, otras métricas del proyecto empiezan a tener significado. Se miden las tasas de producción representadas en términos de modelos creados, horas de revisión, puntos de función y líneas de fuente entregadas. Además, se rastrean los errores descubiertos durante cada tarea de ingeniería del software. Conforme el software evoluciona desde los requerimientos hasta el diseño, se recopilan métricas técnicas a fin de valorar la calidad del diseño y proporcionar indicios que influirán en el enfoque tomado para generación y prueba de código.

Montenegro, Micaela Soledad

La intención de las métricas de proyecto es doble. Primero, se usan para minimizar el calendario de desarrollo al hacer los ajustes necesarios para evitar demoras y mitigar potenciales problemas y riesgos. Segundo, se usan para valorar la calidad del producto sobre una base en marcha y, cuando es necesario, modificar el enfoque técnico para mejorar la calidad.

Conforme la calidad mejora, los defectos se minimizan, y conforme el conteo de defectos baja, la cantidad de reelaboración requerida durante el proyecto también se reduce. Esto conduce a una reducción en el costo global del proyecto.

MÉTRICAS DEL SOFTWARE

Las métricas de software pueden clasificarse en:

- Medidas directas del proceso de software incluyen costo y esfuerzo aplicado. Las medidas directas del producto incluyen líneas de código (LOC) producidas, rapidez de ejecución, tamaño de memoria y defectos reportados sobre cierto espacio de tiempo.
- Medidas indirectas del producto incluyen funcionalidad, calidad, complejidad, eficiencia, confiabilidad, capacidad de mantenimiento entre otras.

El dominio de la métrica del software se dividió en métricas de proceso, proyecto y producto, y se dijo que las métricas de producto que son privadas para un individuo con frecuencia se combinan para desarrollar métricas de proyecto que son públicas para un equipo de software. Luego las métricas de proyecto se consolidan para crear métricas de proceso que son públicas para la organización del software como un todo.

Métricas orientadas a tamaño

Las métricas de software orientadas a tamaño se derivan al normalizar las medidas de calidad y/o productividad para considerar el tamaño del software que se produjo.

Con la finalidad de desarrollar métricas que puedan asimilarse con métricas similares de otros proyectos, pueden elegirse líneas de códigos como un valor de normalización. A partir de los rudimentarios datos contenidos en la tabla, pueden desarrollarse métricas simples orientadas a tamaño para cada proyecto:

- Errores por KLOC (miles de líneas de código).
- Defectos por KLOC.
- $ por KLOC.
- Páginas de documentación por KLOC.

Además, es posible calcular otras métricas interesantes:

- Errores por persona-mes.
- KLOC por persona-mes.
- $ por página de documentación.

Las métricas orientadas a tamaño no se aceptan universalmente como la mejor forma de medir el proceso de software. La mayor parte de la controversia gira en torno del uso de líneas de código como medida clave. Quienes proponen la medida LOC afirman que las LOC son un “artefacto” de todos los proyectos de desarrollo de software y que pueden contarse fácilmente; que muchos modelos existentes de estimación de software usan LOC o KLOC como entrada clave y que ya existe un gran cuerpo de literatura y predicado de datos acerca de LOC. Por otra parte, los opositores argumentan que las medidas LOC dependen del lenguaje de programación; que cuando se considera la productividad, castigan a los programas bien diseñados pero cortos; que no pueden acomodarse con facilidad en lenguajes no procedurales y que su uso en la estimación requiere un nivel de detalle que puede ser difícil de lograr.

Métricas orientadas a función

Las métricas de software orientadas a función usan una medida de la funcionalidad entregada por la aplicación como un valor de normalización. La métrica orientada a función de mayor uso es el punto de función (PF). El

cálculo del punto de función se basa en características del dominio y de la complejidad de información del software. El punto de función, es controvertido. Quienes lo proponen afirman que el PF es independiente del lenguaje de programación, y que se basa en datos que es más probable que se conozcan tempranamente en la evolución de un proyecto. Sus opositores afirman que el método requiere cierta "maña", pues dicho cálculo se basa en datos subjetivos más que objetivos, que el conteo del dominio de información puede ser difícil de recopilar después del hecho y que el PF no tiene significado físico directo: es sólo un número.

Métricas orientadas a objeto

Un conjunto de métricas para proyectos OO puede consistir en:

- Número de guiones de escenario
- Número de clases clave que son los "componentes enormemente independientes" centrales en el dominio del problema
- Número de clases de apoyo
- Número promedio de clases de apoyo por clase clave
- Número de subsistemas

Métricas orientadas a caso de uso

Los casos de uso se utilizan ampliamente como un método para describir los requerimientos en el dominio en el nivel del cliente o empresarial, que implican características y funciones del software. El caso de uso se define al principio del proceso del software, lo que permite emplearlo para estimación antes de iniciar actividades significativas de modelado y construcción. Los casos de uso describen las funciones y características visibles para el usuario que son requisitos básicos para un sistema. El caso de uso es independiente del lenguaje de programación. Además, el número de casos de uso es directamente proporcional al tamaño de la aplicación en LOC y al número de casos de prueba que tendrán que designarse para ejercitar por completo la aplicación.

Puesto que los casos de uso pueden crearse en niveles de abstracción enormemente diferentes, no hay "tamaño" estándar para un caso de uso.

Los investigadores sugieren puntos de caso de uso (PCU) como un mecanismo para estimar el esfuerzo del proyecto y otras características. Los PCU son una función del número de actores y transacciones implicados por los modelos de caso de uso y su análogo al PF en algunas formas.

Métricas de proyectos webapp

El objetivo de todos los proyectos webapp es entregar al usuario final una combinación de contenido y funcionalidad. Entre las medidas que pueden recopilarse están:

- Número de páginas web estáticas
- Número de páginas web dinámicas
- Número de vínculos de página internos que son punteros que proporcionan un hipervínculo hacia alguna otra página web dentro de la webapp.
- Número de objetos de datos persistentes
- Número de objetos de contenido estáticos.
- Número de objetos de contenido dinámicos.
- Número de funciones ejecutables que proporcionan cierto servicio computacional al usuario final

MÉTRICAS PARA CALIDAD DEL SOFTWARE

La calidad de un sistema, aplicación o producto sólo es tan buena como los requerimientos que describen el problema, el diseño que modela la solución, el código que conduce a un programa ejecutable y las pruebas que ejercitan el software para descubrir errores. Conforme el software se somete a ingeniería, pueden usarse mediciones para valorar la calidad de los modelos de requerimientos y de diseño, el código fuente y los casos de prueba que se crearon. Para lograr esta valoración en tiempo real, las métricas de producto se aplican a fin de evaluar la calidad de los productos operativos de la ingeniería del software en forma objetiva, en lugar de subjetiva.

Un gerente de proyecto también debe evaluar la calidad conforme avanza el proyecto. Aunque muchas medidas de calidad pueden recopilarse, el empuje primario en el nivel del proyecto es medir los errores y defectos. Las métricas derivadas de estas medidas proporcionan un indicio de la efectividad de las actividades, individuales y grupales, de aseguramiento y control de la calidad del software.

Métricas como errores de producto operativo por punto de función, errores descubiertos porhora de revisión y errores descubiertos por prueba por hora proporcionan comprensión de la eficacia de cada una de las actividades implicadas por la métrica. Los datos de error también pueden usarse para calcular la eficiencia de remoción de defecto (ERD) para cada actividad de marco conceptual del proceso.

Medición de la calidad

Aunque existen muchas medidas de calidad del software, la exactitud, capacidad de mantenimiento, integridad y usabilidad proporcionan útiles indicadores para el equipo del proyecto.

- Exactitud: Grado en el cual el software realiza la función requerida. La medida más común de la exactitud son los defectos por KLOC, donde un defecto se define como una falta verificada de acuerdo con los requerimientos.
- Capacidad de mantenimiento: Es la facilidad con la que un programa puede corregirse si se encuentra un error, la facilidad con que se adapta si su entorno cambia o de mejorar si el cliente quiere un cambio en requerimientos. Aquí deben usarse medidas indirectas: tiempo medio al cambio (TMC), el tiempo que tarda en analizarse la petición de cambio, diseñar una modificación adecuada, implementar el cambio, probarlo y distribuirlo a todos los usuarios.
- Integridad: Mide la habilidad de un sistema para resistir ataques a su seguridad. Los ataques pueden hacerse en los tres componentes de software: programas, datos y documentación. Para medir la integridad, deben definirse dos atributos adicionales: amenaza y seguridad. Amenaza es la probabilidad de que un ataque de un tipo específico ocurrirá dentro de un tiempo dado. Seguridad es la probabilidad de que el ataque de un tipo específico se repelerá.
- Usabilidad: Es un intento por cuantificar la facilidad de uso.

Eficiencia en la remoción del defecto

Una métrica de calidad que proporciona beneficio tanto en el nivel del proyecto como en el del proceso es la eficiencia de remoción del defecto (ERD). En esencia, la ERD es una medida de la habilidad de filtrado de las acciones de aseguramiento y control de la calidad según se aplican a lo largo de todas las actividades del marco conceptual del proceso.

Cuando se considera para un proyecto como un todo, la ERD se define en la forma siguiente: ERD = E/ (E+D) donde E es el número de errores que se encontraron antes de entregar el software al usuario final y D es el número de defectos que se encontraron después de la entrega.

El valor ideal para la ERD es 1. Es decir, no se encuentran defectos en el software. Si se usa como una métrica que proporciona un indicio de la capacidad de filtrado de las actividades de control y aseguramiento de la calidad, la ERD alienta a un equipo de software a instituir técnicas para encontrar tantos errores como sea posible antes de entregar.

La ERD también puede usarse dentro del proyecto a fin de valorar la habilidad de un equipo para encontrar errores antes de que pasen a la siguiente actividad de marco conceptual o acción de ingeniería del software.

INTEGRACIÓN DE LAS MÉTRICAS DENTRO DEL PROCESO DE SOFTWARE

Argumentos para métricas de software

- Si no se mide el proceso de ingeniería del software y del producto, no hay forma real de determinar si se está mejorando. Y si no se mejora, se está perdido. Al solicitar y evaluar las medidas de productividad y calidad, un equipo de software puede establecer metas significativas para mejorar el proceso de software. Si puede mejorarse el proceso a través del cual se desarrolla, puede tenerse como resultado un impacto

directo sobre la línea de referencia. Por tanto, la medición se utiliza para establecer una línea de referencia del proceso desde el cual puedan valorarse las mejoras.

- Los gerentes de proyecto de software se preocupan por conflictos más mundanos: desarrollar estimaciones de proyecto significativas, producir sistemas de alta calidad, sacar el producto a tiempo. Al usar la medición para establecer una línea de referencia del proyecto, cada uno de estos conflictos se vuelve más manejable.

Establecimiento de una línea de referencia

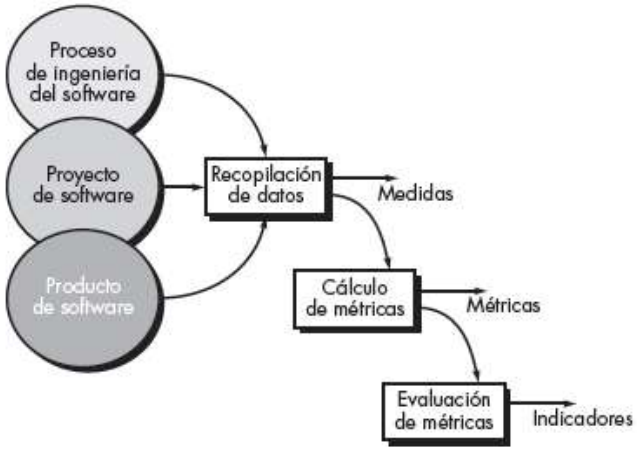
Al establecer una línea de referencia para métricas, pueden obtenerse beneficios en el proceso, el proyecto y el producto. Las mismas métricas pueden servir a muchos dominios. La línea de referencia de métricas consiste en los datos recopilados a partir de los proyectos de desarrollo de software.

Para ser un auxiliar efectivo en el mejoramiento del proceso y/o estimación de costo y esfuerzo, los datos de la línea de referencia deben tener los siguientes atributos:

- Deben ser razonablemente precisos y deben evitarse "suposiciones" acerca de los proyectos anteriores
- Deben recopilarse para tantos proyectos como sea posible
- Deben ser consistentes
- Las aplicaciones deben ser similares al trabajo que debe estimarse

Recolección, cálculo y evaluación de métricas

La recopilación de datos para establecer una línea de referencia requiere una investigación histórica de los proyectos anteriores a fin de reconstruir los datos requeridos. Una vez recopiladas las medidas, es posible el cálculo de métricas. Dependiendo de la envergadura de las medidas recopiladas, las métricas pueden abarcar un amplio rango de métricas orientadas a aplicaciones, así como otras orientadas a calidad y proyecto. Finalmente, las métricas deben evaluarse y aplicarse durante la estimación, el trabajo técnico, el control del proyecto y la mejora del proceso. La evaluación de métricas se enfoca en las razones subyacentes para los resultados obtenidos y produce un conjunto de indicadores que guían al proyecto o al proceso.



ESTABLECIMIENTO DE UN PROGRAMA DE MÉTRICAS DEL SOFTWARE

El manual desarrollado por el Software Engineering Intitute (SEI) para establecer un programa de métrica de software orientado a metas sugiere los siguientes pasos:

1. Identificar las metas empresariales.
2. Identificar lo que se quiere conocer o aprender.
3. Identificar las submetas.
4. Identificar las entidades y atributos relacionados con las submetas.
5. Formalizar las metas de medición.

6. Identificar preguntas cuantificables y los indicadores relacionados que se usarán para ayudar a lograr las metas de medición.
7. Identificar los elementos de datos que se recopilarán para construir los indicadores que ayuden a responder las preguntas.
8. Definir las medidas que se van a usar y hacer operativas estas definiciones.
9. Identificar las acciones que se tomarán para implementar las medidas.
10. Preparar un plan para la implantación de las medidas.

Puesto que el software apoya las funciones empresariales, diferencia los sistemas o productos basados en computadora y actúa como un producto en sí mismo, las metas definidas por la empresa casi siempre pueden rastrearse en las metas específicas de ingeniería del software.

La organización de software examina cada meta empresarial y pregunta: ¿qué actividades manejamos, ejecutamos o apoyamos y qué hacemos para mejorar dichas actividades? Para responder estas preguntas, el SEI recomienda la creación de una "lista de entidad-pregunta" en la que se anotan todas las cosas (entidades: recursos de desarrollo, codigo fuente, casos de prueba) existentes dentro del proceso de software que se gestionan o que influyen en la organización de software. Para cada entidad mencionada, el personal de software desarrolla un conjunto de preguntas que valoran las características cuantitativas de la entidad (por ejemplo, tamaño, costo, tiempo para desarrollar). Las preguntas derivadas como consecuencia de la creación de una lista de entidad-pregunta conducen a la derivación de un conjunto de submetas que se relacionan directamente con las entidades creadas y con las actividades realizadas como parte del proceso de software.

A partir de estas preguntas, la organización de software puede derivar la siguiente submeta: mejorar el rendimiento del proceso de gestión del cambio. Entonces, se identifican las entidades y atributos del proceso de software que son relevantes para la submeta y se delinean las metas de medición asociadas con ellas.

8. ESTIMACIÓN PARA PROYECTO DE SOFTWARE

ESTIMACIÓN

Estimación: Apreciar, poner precio, evaluar algo.

- Estimar es echar un vistazo al futuro con algún grado de incertidumbre.
- La estimación, es mas un arte que una Ciencia.
- Es una actividad importante que no debe llevarse a cabo de forma descuidada.
- Una estimación es una predicción basada en un modelo probabilístico, no un modelo determinístico; es decir, la cantidad que se está estimando puede tomar no solamente un valor sino distintos valores.

Estimación de proyectos de software: Actividad de la planificación del proyecto de software que intenta determinar cuánto dinero, esfuerzo, recursos y tiempo tomará construir un sistema o producto software.

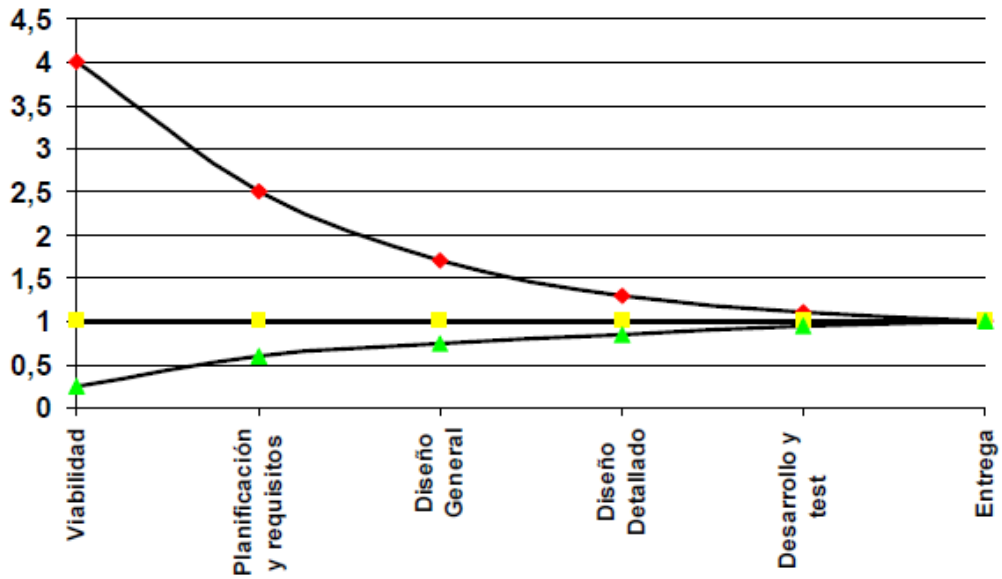
Estimacion para proyectos de software es la aplicación continua de técnicas basadas en las medidas de los procesos de desarrollo del software y sus productos, para producir una información de gestión significativa y a tiempo. Esta información se utilizará para mejorar esos procesos los productos que se obtienen de ellos.

Existen muchos problemas relacionados a la estimación, entre ellos: Averiguar lo que costara desarrollar una aplicación (mesespersona, $,…), Momento en que se desea conocer el costo, Siempre se quiere muy pronto.

Factores que afectan al riesgo de estimación

- La complejidad del proyecto tiene un fuerte efecto sobre la incertidumbre inherente a la planificación. Sin embargo, la complejidad es una medida relativa que es afectada por la familiaridad con el esfuerzo pasado.
- El tamaño del proyecto es otro factor importante que puede afectar la precisión y la eficacia de las estimaciones. Conforme aumenta el tamaño, la interdependencia entre varios elementos del software crece rápidamente.
- El grado de incertidumbre estructural también tiene un efecto sobre el riesgo de estimación. En este contexto, estructura se refiere al grado en el cual se solidificaron los requisitos, la facilidad con la que se dividieron las funciones y la naturaleza jerárquica de la información que debe procesarse.
- Disponibilidad de información histórica: métricas sobre proyectos pasados.

Presición de las estimaciones en función de la fase del proyecto



Mientas más conozca, mejor estimará. En consecuencia, actualice sus estimaciones conforme avance el proyecto.

La precisión de las estimaciones producidas depende de la información del sistema que este disponible. Conforme avance el proceso de software, más información estará disponible y las estimaciones serán más precisas cada vez. Si la estimación inicial del esfuerzo requerido es de x meses de esfuerzo, el rango puede estar comprendido entre 0,25x y 4x en la primera propuesta. Éste se afinará durante el proceso de desarrollo. Por supuesto, justo antes que se entregue el sistema, se puede realizar una estimación muy precisa.

Objetivos de la Estimación

- Predecir las variables involucradas en el proyecto con cierto grado de certeza.
- Trata de aportar una predicción de algún indicador importante para la gestión de proyectos de software tiempo, esfuerzo, cantidad de defectos esperados entre otros.
- Es razonable conocer, antes de comenzar a desarrollar el SW, cuánto se va a invertir, qué tareas se deben realizar y cuánto tiempo se necesitará.

Estimador

Se requiere de un estimador temprano, independiente del lenguaje y fácil de calcular. El estimador debe ser un profesional que no tenga ningún interés, directo o indirecto, en los resultados del proceso de estimación y que este únicamente guiado por su profesionalismo.

El principal objetivo del estimador es obtener estimaciones de calidad, las cuales no tienen siempre por qué coincidir con las expectativas de la empresa en términos de costo y tiempo.

Requisitos que debe cumplir un buen estimador:

- Formación y experiencia profesional adecuada.
- Una posición en la organización que le permita adoptar un juicio independiente.
- Debe basarse en un método que pueda ser explicado, cuestionado, discutido y auditado.
- Debe poder describir su experiencia en cada estimación.
- Debe documentar su estimación, incluyendo los resultados obtenidos y cualquier información necesaria para hacer el proceso de estimación repetible y verificable.

Cuándo se debe llevar a cabo la estimación

La estimación es un proceso continuo. A medida que el proyecto avanza, más se conoce de él, y por lo tanto más parámetros están disponibles para introducir en un modelo de estimación.

La estimación continua nos permite el uso de un único modelo coherente que pueda capturar y utilizar la información sobre el proyecto a medida que éste se conozca.

El proceso de estimación comienza usando unas pocas variables claves para proveer las «macrocaracterísticas» de un proyecto, y evoluciona incorporando información de más bajo nivel para producir las «micro- características» del proyecto.

Problemas de estimación

- Problemas Políticos: cuando las estimaciones se convierten en objetivos, cuando se ajusta el precio por conveniencia.
- Problemas Técnicos: No existen datos históricos para estimar.

Planificación de proyectos

El objetivo de la planificación del proyecto de software es proporcionar un marco conceptual que permita al gerente hacer estimaciones razonables de recursos, costo y calendario. Además, las estimaciones deben intentar definir los escenarios de mejor caso y peor caso, de modo que los resultados del proyecto puedan acotarse. Aunque hay un grado inherente de incertidumbre, el equipo de software se embarca en un plan que se haya establecido como consecuencia de dichas tareas. Por tanto, el plan debe adaptarse y actualizarse conforme avanza el proyecto.

Montenegro, Micaela Soledad

CONJUNTO DE TAREAS

Conjunto de tareas para planificación de proyectos

1. Establecer ámbito del proyecto.
2. Determinar la factibilidad.
3. Analizar los riesgos (capítulo 28).
4. Definir recursos requeridos.
   a) Determinar recursos humanos requeridos.
   b) Definir recursos de software reutilizables.
   c) Identificar recursos ambientales.
5. Estimar costo y esfuerzo.
   a) Descomponer el problema.
   b) Desarrollar dos o más estimaciones usando tamaño, puntos de función, tareas de proceso o casos de uso.
   c) Reconciliar las estimaciones.
6. Desarrollar un calendario del proyecto (capítulo 27).
   a) Establecer un conjunto de tareas significativas.
   b) Definir una red de tareas.
   c) Usar herramientas de calendarización para desarrollar un cronograma.
   d) Definir mecanismos de seguimiento de calendario.

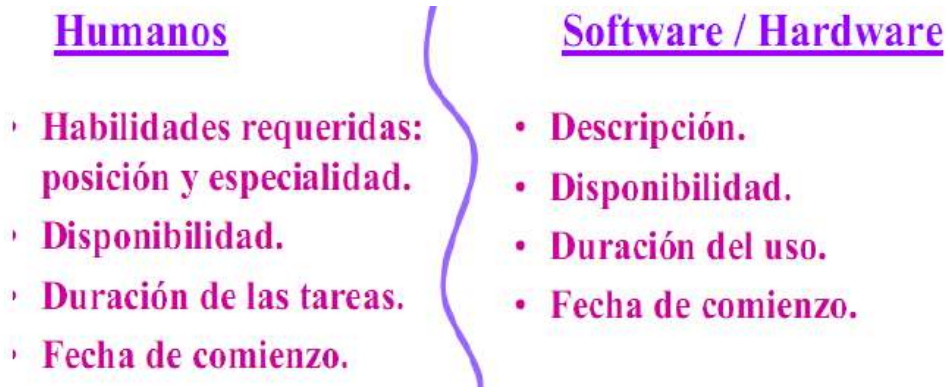
- Ambito del Software:

El ámbito del software describe las funciones y características que se entregan a los usuarios finales; los datos que son entrada y salida; el "contenido" que se presenta a los usuarios como consecuencia de usar el software y el desempeño, las restricciones, las interfaces y la confiabilidad que se ligan al sistema.

- Recursos

La segunda tarea en la planificación es la estimación de los recursos requeridos para lograr el esfuerzo de desarrollo del software. Las tres principales categorías de los recursos de la ingeniería de software: personal, componentes de software reutilizables y entorno de desarrollo (herramientas de hardware y software). Cada recurso se especifica con cuatro características: descripción del recurso, un enunciado de disponibilidad, momento en el que se requerirá el recurso y duración del tiempo que se aplicará el recurso.

- Recursos Humanos: El planificador comienza por evaluar el ámbito del software y seleccionando las habilidades requeridas para completar el desarrollo. Se especifican tanto la posición organizacional como la especialidad. El número de personas requeridas para un proyecto de software puede determinarse sólo después de hacer una estimación del esfuerzo de desarrollo.
- Recursos de Software Reutilizables: Existen cuatro categorías de recursos de software que deben considerarse conforme avanza la planificación: Componentes comerciales, Componentes de experiencia completa, Componentes de experiencia parcial y Componentes nuevos.
- Recursos Ambientales: El entorno que soporta a un proyecto de software, con frecuencia llamado entorno de ingeniería de software (EIS), incorpora hardware y software. El hardware proporciona una plataforma que soporta las herramientas (software) requeridas para producir los productos operativos que son resultado de la buena práctica de la ingeniería de software.



ESTIMACIÓN DE PROYECTOS DE SOFTWARE

La estimación de costo y esfuerzo del software nunca será una ciencia exacta. Demasiadas variables (humanas, técnicas, ambientales, políticas) pueden afectar el costo final del software y el esfuerzo aplicado para su desarrollo.

Sin embargo, la estimación del proyecto de software puede transformarse de un arte oscuro a una serie de pasos sistemáticos que proporcionen estimaciones con riesgo aceptable. Para lograr estimaciones confiables de costo y esfuerzo, surgen algunas opciones:

- Retrase la estimación hasta avanzado el proyecto: esta opción no es práctica ya que las estimaciones de costo deben proporcionarse por anticipado.
- Base las estimaciones en proyectos similares que ya estén completos: puede funcionar razonablemente bien si el proyecto actual es muy similar a esfuerzos anteriores y otros factores que influyen en el proyecto son aproximadamente equivalentes.
- Use técnicas de descomposición relativamente simples para generar estimaciones de costo y esfuerzo de proyecto.
- Use uno o más modelos empíricos para estimación de costo y esfuerzo de software.
- Adquirir herramientas automáticas de estimación.

Algunas de los métodos utilizados para la estimación de proyectos son:

- Basados en la experiencia.
- Basado exclusivamente en los recursos.
- Método basado exclusivamente en el mercado.
- Basado en los componentes del producto o en el proceso de desarrollo.
- Métodos algorítmicos

ESTIMACIÓN POR DESCOMPOSICIÓN

Las técnicas de descomposición tienen un enfoque de "divide y vencerás" para la estimación del proyecto. Al descomponer un proyecto en funciones principales y actividades de ingeniería de software relacionadas, la estimación de costo y esfuerzo puede realizarse en forma escalonada.

Las técnicas de descomposición son:

Estimación basada en el problema

Las estimaciones de LDC y PF son distintas técnicas de estimación, aunque ambas tienen varias características en común. El planificador comienza con un enfoque acotado del ámbito del software y a partir de ahí intenta descomponer el software en funciones problema que puedan estimarse individualmente. Entonces se estiman las LDC o PF para cada función. De manera alternativa, el planificador puede elegir otro componente para tamaño, como clases u objetos, cambios o procesos de negocio afectados. Entonces se aplican las métricas de la línea base de productividad a la variable de estimación apropiada, y se deriva el costo o esfuerzo de la función.

Las técnicas de estimación LDC y PF difieren en cuanto al detalle requerido para descomposición y el objetivo de la partición. Al emplear LDC como variable de estimación la descomposición es absolutamente esencial y con frecuencia se lleva a grados considerables de detalle. Mientras mayor sea el grado de partición más probable que se desarrolle una estimación razonablemente precisa de LDC.

En la estimación PF la descomposición funciona de manera diferente. Más que enfocarse sobre la función se estima cada una de las cinco características de dominio de información.

Estimación basada en el proceso

La técnica más común para estimar un proyecto es basar la estimación en el proceso que se empleará. El proceso se descompone en un conjunto relativamente pequeño de tareas y se estima el esfuerzo requerido para lograr cada tarea.

MODELOS EMPÍRICOS DE ESTIMACIÓN

Un modelo empírico se basa en la experiencia (datos históricos) y toma la forma: d = f (vi) donde d es uno de los valores estimados (por ejemplo, esfuerzo, costo, duración del proyecto) y vi son parámetros independientes seleccionados (por ejemplo, LOC o PF estimadas).

Un modelo de estimación para software de computadora utiliza fórmulas obtenidas empíricamente para predecir el esfuerzo como una función de LDC o PF. Los datos empíricos que apoyan la mayoría de los modelos de estimación proceden de una muestra limitada de proyectos. Por esta razón, ningún modelo de estimación es apropiado para todas las clases de software ni en todos los entornos de desarrollo. En consecuencia, los resultados obtenidos a partir de tales modelos se deben emplear juiciosamente. Un modelo de estimación debe calibrarse para reflejar las condiciones locales. El modelo debe probarse mediante la aplicación de los datos recopilados a partir de proyectos completados, colocar los datos en el modelo y luego comparar los resultados reales con los predichos. Si la concordancia es insuficiente, el modelo debe afinarse y ponerse a prueba nuevamente antes de que se pueda utilizar.

Algunos de los métodos basados únicamente en la experiencia encontramos:

Juicio experto

Se consultan varios expertos en las técncas de desarrollo de software propuestas y en el dominio de la aplicación. Cada uno de ellos estima el coste del proyecto. Estas estimaciones se comparan y discuten. El proceso de estimación se itera hasta que se llega a un concenso. La técnica de estimación Juicio Experto tiene dos variantes:

- Puro: Se basa fundamentalmente en los conocimientos del experto. Aquín experto estudia las especificaciones y hace su estimación, y en caso de desaparecer el experto, la empresa deja de estimar.
- Wideband Delphi: Se basa en la idea de que las estimaciones en grupo suelen ser mejores que las individuales. Aquí un grupo de personas son informadas y tratan de adivinar lo que costara el desarrollo tanto en esfuerzo, como su duración. El método de trabajo consiste en:
    - Se dan las especificaciones a un grupo de expertos.
    - Se les reúne para que discutan tanto el producto como la estimación.
    - Remiten sus estimaciones individuales al coordinador.
    - Cada estimador recibe información sobre su estimación, y las ajenas pero de forma anónima.
    - Se reúnen de nuevo para discutir las estimaciones.
    - Cada uno revisa su propia estimación y la envía al coordinador.
    - Se repite el proceso hasta que la estimación converge de forma razonable.

Analogia

Consiste en comparar las especificaciones de un proyecto, con las de otros proyectos. Analogía, pueden variar los siguientes factores:

- Tamaño: ¿mayor o menor?
- Complejidad: ¿Más complejo de lo usual?
- Usuarios: Si hay más usuarios habrán más complicaciones.
- Otros factores:
    - Sistema Operativo, entornos (la primera vez más).
    - Hardware, ¿Es la primera vez que se va a utilizar?
    - Personal del proyecto, ¿nuevos en la organización?

Distribucion de la utilizacion de recursos en el ciclo de vida

Usualmente las organizaciones tienen una estructura de costes similar entre proyectos. Si en un proyecto ya hemos realizado algunas fases, es de esperar que los costes se distribuyan de manera proporcional.

Estudio Viabilidad	Planificación y Requisitos	Diseño General	Diseño Detallado	Desarrollo	Prueba
10%	17%	15%	15%	33%	10%

2 m. ?

MÉTODO BASADO EXCLUSIVAMENTE EN LOS RECURSOS

En la estimación consiste en ver de cuanto personal y durante cuanto tiempo se dispone de él, haciendo esa estimación. En la realización: "El trabajo se expande hasta consumir todos los recursos disponibles" (Ley de Parkinson).

MÉTODO BASADO EXCLUSIVAMENTE EN EL MERCADO

Lo importante es conseguir el contrato. Aquí el precio se fija en función de lo que creemos que esta dispuesto a pagar el cliente.

Si se usa en conjunción con otros métodos puede ser aceptable, para ajustar la oferta. El peligro está si es el único método utilizado.

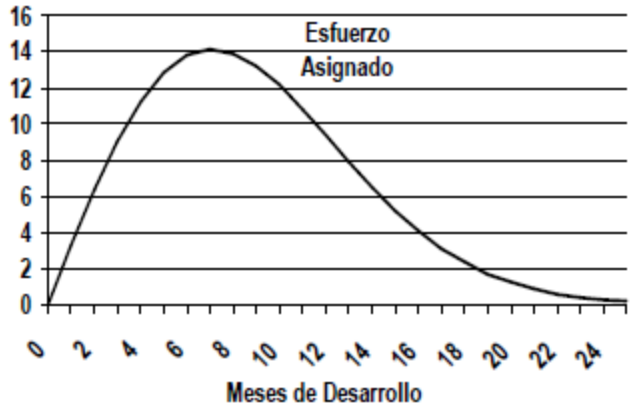
MÉTODO BASADO EN LOS COMPONENTES DEL PRODUCTO O EN EL PROCESO DE DESARROLLO

Bottom-up

Se descompone el proyecto en las unidades lo menores posibles, se estima cada una y se calcula el coste total.

Top-Down

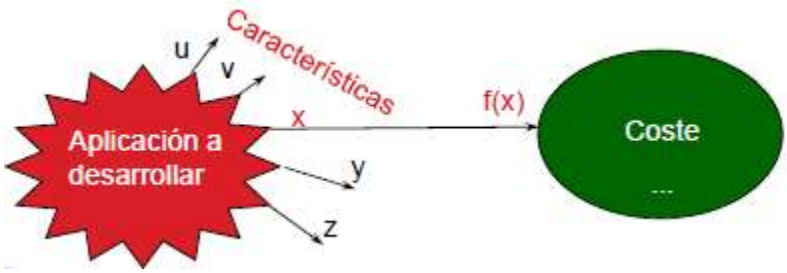
Se ve todo el proyecto, y se descompone en grandes bloques o fases. Luegp se estima el coste de cada componente.



MÉTODOS ALGORÍTMICOS

Se basan en la utilización de fórmulas que aplicadas sobre modelos top-down o bottom-up y producen una estimación de coste del proyecto.

Aquí se desarrolla un modelo utilizando información histórica de coste que relaciona alguna métrica de software, por lo general el tamaño, con el coste del proyecto. Se hace una estimación de esa métrica y el modelo predice el esfuerzo requerido.



MODELO DE PUTNAM

Es un modelo empírico que relaciona cantidad de personas-mes y la duración del proyecto: $Y= 2Kate^{(-at^2)}$

Y = Personas-mes en cada punto

K = Esfuerzo total del proyecto, (Área bajo la curva)

a = Cte. asociada a la aceleración de entrada de personas en el proyecto

t = instante del tiempo.

Se puede observar que a pequeños cambios en el tiempo de desarrollo, provocan grandes modificaciones en el esfuerzo (al expandir el tiempo, el esfuerzo disminuye). Como así también podemos ver que aumentar rápidamente el personal de desarrollo tiene mayor impacto sobre el coste y el esfuerzo, que sobre el tiempo.

MODELO COCOMO

COCOMO (COnstructive COst MOdel), es el modelo de estimación de costes más utilizado propuesto por Barry Boehm

Es un modelo empírico que se obtuvo recopilando datos de varios proyectos grandes. Estos datos fueron analizados para descubrir las formulas que mejor se ajustaban a las observaciones. Estas fórmulas vinculan el tamaño del sistema y del producto, factores del proyecto y del equipo con el esfuerzo necesario para desarrollar el sistema.

El principal cálculo en el modelo COCOMO es el uso de la ecuación del esfuerzo para estimar el número de personas o de meses necesarios para desarrollar el proyecto.

ESFUERZO = A * TAMAÑO^B

A es un factor constante que depende de las prácticas organizacionales y del tipo de software que se esta desarrollando. B es un valor que generalmente se encuentra entre 1 y 1,5. TAMAÑO es la valoración del tamaño del código de software. Como podemos apreciar, se parte de conocer el número de líneas que tendrá la futura aplicación (tamaño medido en SLOC).

En COCOMO es fundamental el término A que hace referencia al modo de desarrollo del proyecto, que influye directamente en el costo y duración del mismo, y que puede ser:

- Modo orgánico: cuando el proyecto es desarrollado en un ambiente familiar y estable, y en el que el producto es similar a otros desarrollados previamente.
- Modo empotrado: para proyectos caracterizados por unos requerimientos y restricciones poco flexibles, que requieren un gran esfuerzo de innovación
- Modo semiacoplado: es un modelo para proyectos que presentan características intermedias entre el orgánico y el empotrado, conun equipo formado por miembros de distintos niveles de experiencia, que trabajan sobre un conjunto de requisitos más o menos flexibles.

COCOMO está definido en términos de tres modelos diferentes: modelo básico, intermedio y detallado, que reflejan el nivel de detalle considerado a la hora de realizar la estimación del coste. El modelo básico provee una estimación inicial poco refinada, el intermedio la modifica utilizando una serie de multiplicadores relacionados con el proyecto y el proceso, y el detallado realiza diferentes estimaciones para cada fase del proyecto.

- Modelo básico: La ecuación básica del esfuerzo queda definida de la siguiente forma:

MODO DE DESARROLLO	ECUACIÓN BÁSICA DEL ESFUERZO
Orgánico	Esfuerzo = $2.4 * KSLOC^{1.05}$
Semiacoplado	Esfuerzo = $3.0 * KSLOC^{1.12}$
Empotrado	Esfuerzo = $3.6 * KSLOC^{1.20}$

Y las estimaciones sobre el tiempo son:

MODO DE DESARROLLO	ECUACIÓN BÁSICA DEL TIEMPO
Orgánico	Tiempo de desarrollo = $2.5 * Esfuerzo^{0.38}$
Semiacoplado	Tiempo de desarrollo = $2.5 * Esfuerzo^{0.35}$
Empotrado	Tiempo de desarrollo = $2.5 * Esfuerzo^{0.32}$

- Modelo intermedio y detallado: La ecuacion de esfuerzo quedan definidas como:

MODO DE DESARROLLO	ECUACIÓN BÁSICA DEL ESFUERZO
Orgánico	Esfuerzo = $3.2 * KSLOC^{1.05}$
Semiacoplado	Esfuerzo = $3.0 * KSLOC^{1.12}$
Empotrado	Esfuerzo = $2.8 * KSLOC^{1.20}$

Y por ejemplo la estimación del tiempo para el orgánico es: Tiempo de desarrollo= 2.5 * esfuerzo^0.38

En este modelo se debe

- Determinar los multiplicadores del esfuerzo: Tamaño B.D., experiencia analistas, herramientas, … (15 en total, varían de 0.75-1.66)
- Estimación esfuerzo con las correcciones.
- Estimación de factores relacionados ($, duración fases,…)

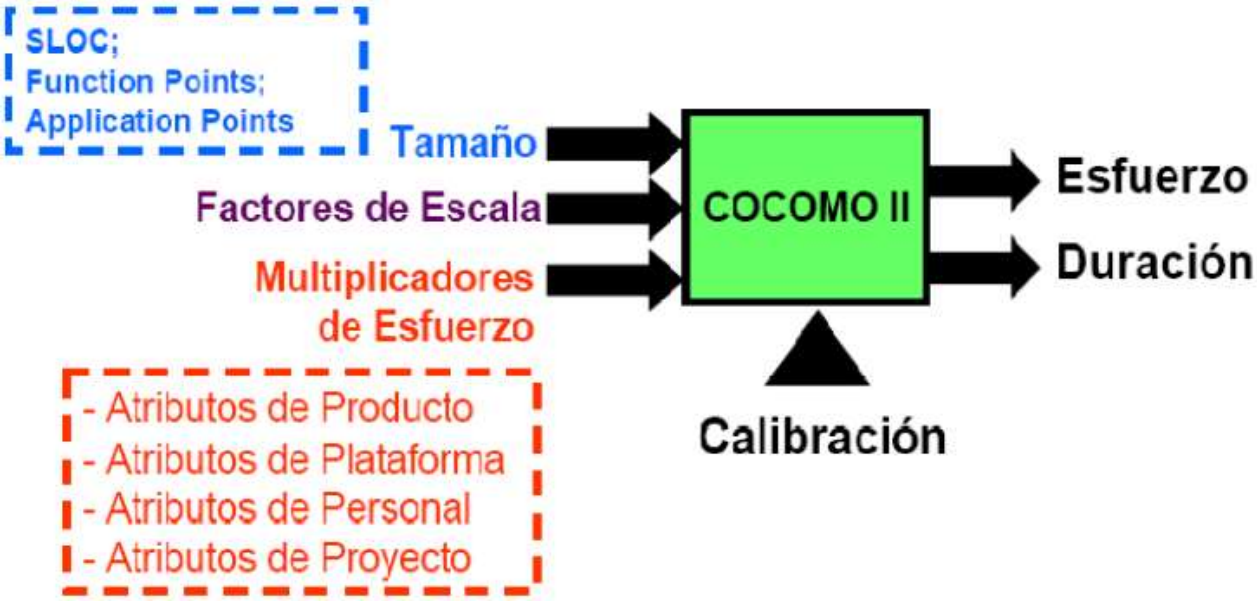
MODELO COCOMO II

COCOMO II es un modelo que permite estimar el coste, el esfuerzo y el tiempo cuando se planifica una nueva actividad de desarrollo software, y está asociado a los ciclos de vida modernos. Fue desarrollado a partir de COCOMO, incluyendo actualizaciones y nuevas extensiones más adecuadas a los requerimientos de los ingenieros software.

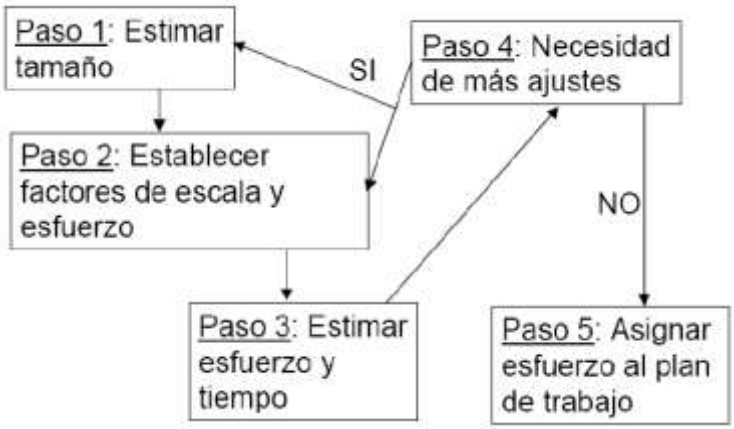
COCOMO II en realidad es una jerarquía de modelos de estimación que aborda las áreas siguientes:

- Modelo de composición de aplicación. Se usa durante las primeras etapas de la ingeniería de software, cuando son primordiales la elaboración de prototipos de las interfaces de usuario, la consideración de la interacción del software y el sistema, la valoración del rendimiento y la evaluación de la madurez de la tecnología.
- Modelo de etapa temprana de diseño. Se usa una vez estabilizados los requisitos y establecida la arquitectura básica del software.
- Modelo de etapa postarquitectónica. Se usa durante la construcción del software.

Concepto operacional de COCOMO II



Proceso de estimación con COCOMO II



Los modelos de COCOMO II usan la siguiente Ecuación de Cálculo del Esfuerzo:

Esfuerzo = A * EAF * TAMAÑO^B

Donde EAF es la productoria de Multiplicadores de Esfuerzos:

1- Atributos de Producto
    1- Confiabilidad requerida
    2- Tamaño de la base de datos
    3- Complejidad del producto
    4- Documentación requerida
    5- Reuso requerido
2- Atributos de Plataforma
    1- Limitaciones de tiempo de ejecución
    2- Limitaciones de almacenamiento
    3- Volatilidad de la plataforma
    4- Tiempo de respuesta promedio
3- Atributos de Proyecto
    1- Prácticas de programación moderna
    2- Uso de herramientas de software
    3- Limitaciones de tiempo de desarrollo
    4- Desarrollo multisitio
4- Atributos de Personal
    1- Capacidad de los analistas
    2- Experiencia en aplicaciones similares
    3- Capacidad de los programadores
    4- Experiencia en la plataforma
    5- Experiencia en lenguaje y herramienta
    6- Continuidad del personal

B es un factor que toma valores comprendidos entre 1.01 y 1.26, que viene dado por B = 1.01 + Σwi, siendo Σwi la suma de cinco Factores de Escala que causan efecto en la economía del proyecto, y que son:

- Existencia de precedentes al proyecto.
- Flexibilidad de la espeificación.
- Resolución de riesgos.
- Cohesión del equipo.
- Madurez del proceso.

Y el TAMAÑO es la estimación del tamaño del proyecto expresada en SLOC, Puntos de Función o Puntos de Objeto.

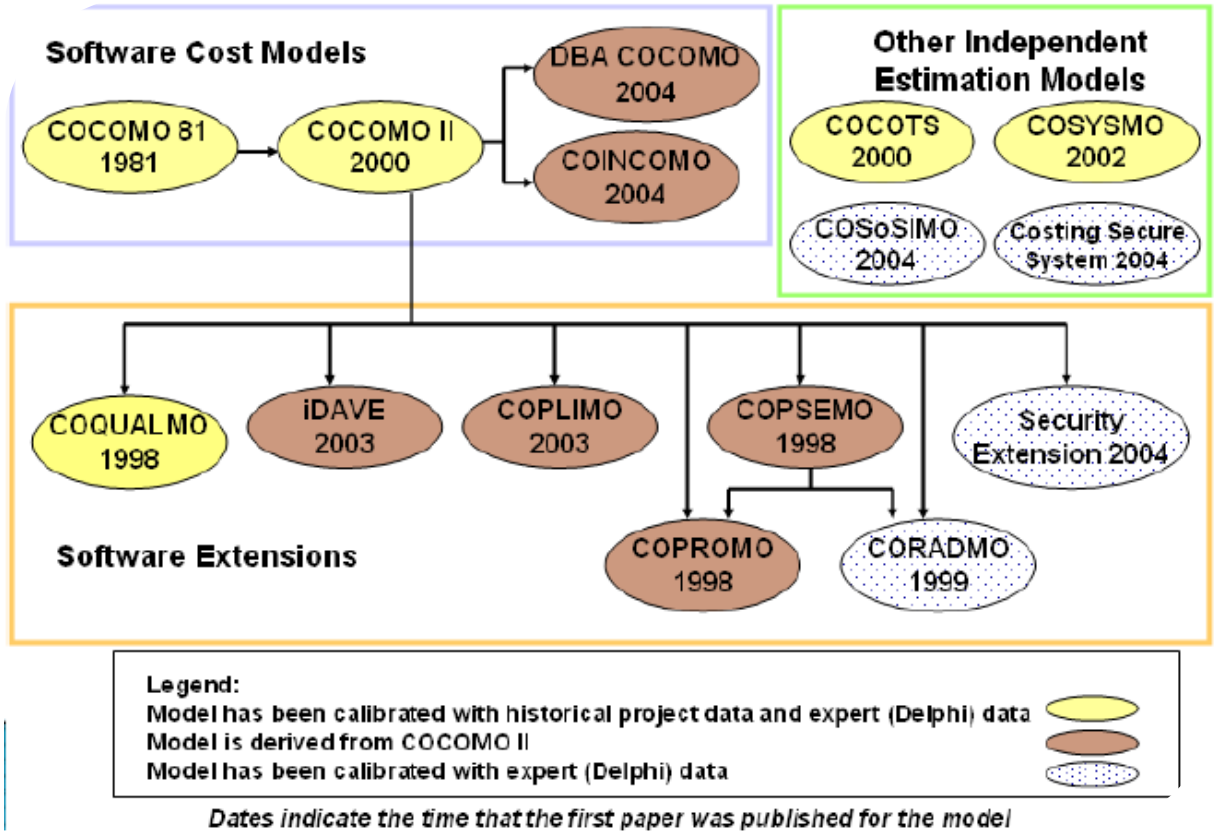
- Líneas de Código (LOC): Es factible si se tiene código desarrollado.

- Puntos de Función (PF): Permite medir la funcionalidad del sistema desde la perspectiva del usuario. Existen algunos métodos de conteo (Albrecht IFPUG, MKII, NESMA, COSMIC-FFP.)

Limitaciones de COCOMO II

- Está basado en un ciclo de vida en cascada
- El modelo NO sirve para proyectos pequeños, esfuerzo < 16 PM, menos de 2 programadores por año.

Alternativas derivadas de COCOMO



ESTIMACIÓN PARA PROYECTOS ORIENTADOS A OBJETOS

Vale la pena complementar los métodos de estimación de costo de software convencional con una técnica que se diseñó explícitamente para software OO. Lorenz y Kidd [Lor94] sugieren el siguiente enfoque:

1- Desarrollar estimaciones usando descomposición de esfuerzo, análisis PF y cualquier otro método que sea aplicable para aplicaciones convencionales.
2- Usar el modelo de requisitos, desarrollar casos de uso y determinar un conteo. Reconocer que el número de casos de uso puede cambiar conforme avance el proyecto.
3- A partir del modelo de requisitos, determinar el número de clases clave.
4- Categorizar el tipo de interfaz para la aplicación y desarrollar un multiplicador para clases de apoyo:

Tipo de interfaz	Multiplicador
No GUI	2.0
Interfaz de usuario basada en texto	2.25
GUI	2.5
GUI compleja	3.0

Multiplique el número de clases clave (paso 3) por el multiplicador a fin de obtener una estimación para el número de clases de apoyo.

5- Multiplicar el número total de clases (clave + apoyo) por el número promedio de unidades de trabajo por clase. Lorenz y Kidd sugieren 15 a 20 persona-días por clase.
6- Comprobación cruzada de la estimación basada en clase, multiplicando el número promedio de unidades de trabajo por caso de uso.

PUNTOS DE FUNCION (PF)

Existían muchos problemas con las Líneas de Códigos (LOC) entre ellos:

- No existe definición estándar para todos los lenguajes.
- Variaciones de tamaño por estilos de programación.
- Funciones de software liberadas sin producir código.

El objetivo de los PF era medir la productividad de varios proyectos en distintos lenguajes.

Objetivos de una métrica funcional

1- Facilidades externas, visibles del software.
2- Factores que son importantes para los usuarios.
3- Aplicable temprano en el ciclo de vida.
4- Relación con la productividad económica.
5- Independiente del código fuente o lenguaje. Fácil de aplicar y calcular.
6- Predicción del tamaño de todas las salidas (particularmente del código fuente para cualquier lenguaje).
7- Cálculo reverso sobre software existente (factor de expansión).
8- Proyectos de mantenimiento y mejora.
9- Todo tipo de aplicaciones (incluyendo MIS, software de base, sistemas de tiempo real, software embebido, etc.).
10- Recolección de datos “hard” del proyecto (fechas, personal, esfuerzo, costo, etc.), por medio de un estándar de conteo.
11- Recolección de datos “soft” del proyecto (habilidades, experiencias, métodos, herramientas, etc.), en forma no ambigua.

Métrica de los PF

Es una métrica que se puede aplicar en las primeras fases de desarrollo. Se basa en características fundamentalmente “Externas” de la aplicación a desarrollar.

Mide dos tipos de características: los elementos de función (entradas, salidas, ficheros, etc.) y los factores de Complejidad.

Estimación del esfuerzo requerido

Partimos de los datos históricos de la Organización Esfuerzo = PFA * Promedio (Lenguaje)

Nombre Proyecto	Puntos de Función	Lenguaje	Esfuerzo en horas
Sénia	200	COBOL	5.017
Mijares	300	PASCAL	5.410
Paláncia	150	PASCAL	2.569
Turia	375	4GL	3.011
Albufera	500	PASCAL	9.479
Magro	425	4GL	3.342
Cabriel	800	PASCAL	13.349
Júcar	180	PASCAL	2.800
Serpis	325	4GL	2.541
Montnegre	225	PASCAL	4.528
Vinalopó	310	PASCAL	5.628
Segura	470	COBOL	13.218

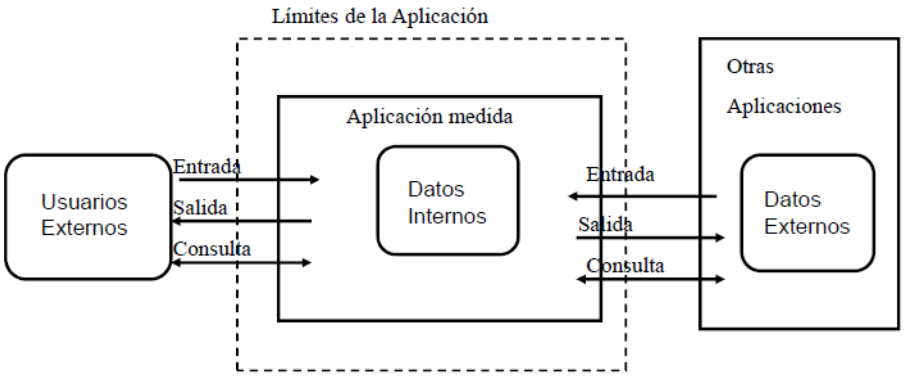
El método de estimación de costes mediante los puntos de función ha sido denominado FPA o Análisis de Puntos de Función. Este método se basa no en las LDC sino en una métrica que cuantifica la funcionalidad que hay que entregar al usuario al construir una aplicación. Dicha métrica se denomina puntos de función.

Se caracterizan por:

- Tener un componente empírico, basado en la experiencia de muchosproyectos.
- Tener en cuenta la complejidad, aunque es muy difícil de determinar en un proyecto
- Ser independientes del entorno tecnológico y de las metodologías aplicadas.
- Utilizar medidas indirectas, que se caracterizan por ser subjetivas y difíciles de calcular, sin embargo el resultado obtenido es fácilmente comparable

<u>Puntos de Función</u>

Métrica que considera totales ponderados de entradas, salidas, consultas, archivos lógicos e interfaces pertenecientes a una aplicación.



- Unidad abstracta del "valor económico" de un producto de software
- No incluye calidad del producto
- Suma ponderada de cuatro atributos observables:
  - Nro. de Entradas (x4)
  - Nro. de Salidas (x5)
  - Nro. de Consultas (x4)
  - Nro. de Archivos Maestros (x10)
- Ajustables por un "factor de complejidad" en + - 25%
- Problemas: complejidad totalmente subjetiva, rango de variación insuficiente

El cálculo de los PF se realiza de la siguiente manera:

- Identificación de parámetros y su complejidad. Esto da los PFNA

Parámetros	Cantidad	Complejidad			PF
		simple	*media*	*alta*	
Nº de entradas de usuario		x 3	x 4	x 6	
Nº de salidas de usuario		x 4	x 5	x 7	
Nº de peticiones de usuario		x 3	x 4	x 6	
Grupos lógicos de datos internos		x 7	x 10	x 15	
Grupos lógicos de datos externos		x 5	x 7	x 10	
					Σ

- Una vez calculado este valor se debe ajustar a las características del proyecto mediante un factor de complejidad (FA).
- Existen 14 factores que contribuyen a la complejidad de una aplicación.
- Se debe valorar cada uno de ellos dentro de una escala del cero al cinco

<u>Definición de tipos de elementos</u>

- Archivos lógicos internos (ALI o ILF): Grupo de datos lógicos o de información de control, identificables por el usuario, mantenidos a través de procesos elementales de la aplicación dentro de los límites de la misma.
- Archivo de interface externa (AIE o EIF): Grupo de datos lógicos o de información de control, identificables por el usuario, referenciados por la aplicación pero mantenidos a través de procesos elementales de una aplicación diferente.

Cantidad de archivos referenciados	Cantidad de elementos de datos 1 - 4	5 -15	> 15
1	Baja	Baja	Promedio
2	Baja	Promedio	Alta
> 2	Promedio	Alta	Alta

- Entradas externas (EE o EI): Proceso elemental de la aplicación que procesa datos o información de control que entran desde el exterior del sistema. Los datos procesados mantienen uno o más ALIs, las información de control puede o no mantener un ALI.
- Salidas externas (SE o EO): Es un proceso elemental de la aplicación que genera datos o información de control que es enviado fuera de los límites del sistema. Los datos provienen de uno o más ALIs
- Consultas externas (CE o EQ): Es un proceso elemental de la aplicación que consiste de una combinación de entrada/salida que resulta de un recupero de información desde los ALIs. La parte de la entrada es información de control la cual detalla el requerimiento, especificando qué y/o cómo los datos deben ser recuperados. La parte de la salida contiene los datos recuperados. No se actualizan ALIs durante el proceso.
- Tipos elementales de datos (TED o DET): Son campos/atributos únicos reconocibles por el usuario, no recursivos, y pueden incluir claves foráneas.
- Tipos elementales de registros (TER o RET): Son subgrupos de elementos de datos (mandatorios o no) reconocidos por el usuario, contenidos dentro de los ALIs y de los AIEs.
- Tipos de archivos referenciados (TAR o FTR): Son el total de ALIs mantenidos, leídos o referenciados, y el total de AIEs leídos o referenciados por una transacción de entrada o salida.

<u>Matriz de complejidad</u>

Entradas externas:

Salidas externas:

Factores que contribuyen a la complejidad de una aplicación:

Factores de complejidad (FC)
Comunicación de datos
Rendimiento
Frecuencia de transacciones
Requisitos de manejo del usuario final
Procesos complejos
Facilidad de mantenimiento
Instalación en múltiples lugares

Cantidad de archivos referenciados	Cantidad de elementos de datos 1 - 5	6 -19	> 19
1	Baja	Baja	Promedio
2 -3	Baja	Promedio	Alta
> 3	Promedio	Alta	Alta

A cada factor se le asigna un pero entre una escala de 0 a 5:

- 0 Factor no presente o sin influencia
- 1 Influencia insignificante
- 2 Influencia moderada
- 3 Influencia promedio
- 4 Influencia significativa
- 5 Influencia alta

Ejemplo: Para el factor Comunicación de Datos:

- 0 Aplicaciones batch o en PC standalone

- 1 Impresión o ingreso de datos remota
- 2 Impresión e ingreso de datos remota
- 3 Front end con teleprocesamiento en la aplicación
- 4 Aplicaciones con teleprocesamiento significativo
- 5 Aplicaciones con teleprocesamiento dominante

Otro ejemplo: Para el factor Funciones Distribuidas:

- 0 Aplicaciones totalmente monolíticas
- 1 Aplicaciones que preparan da datos para otras componentes
- 2 Aplicaciones distribuidas en pocas componentes
- 3 Aplicaciones distribuidas en más componentes
- 4 Aplicaciones distribuidas en muchas componentes
- 5 Aplicaciones dinámicamente ejecutadas en muchas componentes

Luego para obtener los PF ajustados:

- Sumar los factores de complejidad (FC)
- Multiplicar la suma por 0,01
- Sumarle 0,65
- Multiplicar por el valor PF "sin ajustar"
- PF Ajustados = PF * ((FC * 0,01) + 0,65)

<u>Datos históricos: base de mediciones</u>

- Tamaño
    - PF y LOC
    - Cálculo directo y cálculo reverso
- Estimados
    - Esfuerzo
    - Duración
- Reales (insumidos)
    - Esfuerzo
    - Duración
- Apertura por actividades del ciclo de vida
    - Requerimientos, desarrollo, implantación (como mínimo)
- Staff
    - Tamaño medio, pico, características del equipo
- Características del proyecto
    - Tipo, Dominio de aplicación, Tecnología
- Relación tamaño con CU
    - PF por módulo
    - PF promedio por CU
    - PF-UCP
- Testing
    - Cantidad de casos de prueba diseñados por PF
    - Cantidad de defectos
- Retrabajo
    - Proporción sobre insumido
    - Por actividad del ciclo de vida

<u>Vender o comprar</u>

- Especificación de la función y el rendimiento del software y definición de las características que se puedan medir.
- Estimación del coste interno de desarrollo y de la fecha de entrega.

- Selección de candidatos:
    - Selección de componentes de software reutilizables que puedan ayudar en el desarrollo de la aplicación.
    - Selección de las tres o cuatro aplicaciones que mejor cumplan las especificaciones.
    - Comparación, una a una, de todas las funciones clave entre los diferentes candidatos.
- Evaluación de cada paquete de software o componente según la calidad de productos anteriores, soporte del vendedor, reputación del producto, etc.
- Contacto con otros usuarios de dicho software y valoración de las opiniones.

Subcontratación

Ventajas:

- Económicas: reducción del número de personas y de la infraestructura necesaria.
- Mayor calidad, por la enorme especialización de las empresas.

Desventajas:

- Pérdida de control sobre el software a desarrollar.

USE CASE POINT

- Karner, 1993
- Enfoque semejante a PF
    - Cuenta aspectos claves de los requerimientos para obtener puntos no ajustados
    - Usa varios conjuntos de cuestiones sobre el equipo y su entorno para crear un factor de ajuste
    - Multiplica la cuenta original por el factor para obtener puntos ajustados, que traduce en una estimación de esfuerzo en horas/personas(LOE, Level of effort)
- Ajuste de Kraner
    - Muy parecido a los factores de PF
    - LOE 20 horas /persona por UCP

Cálculo

- Ranquer actores: Simple (1pto), Medio (2pto), Complejo (3pto)
    - Simple: una maquina con una api programable
    - Medio: un humano con una interfaz de línea comando o maquina con algún protocolo
    - Complejo un humano con una GUI
- Ranquer caso de usos: Simple(5pto) Medio(10pto) Complejo (15pto)
    - Simple: <4 escenarios claves o caminos de ejecución
    - Medio :4 o + escenarios pero < 8
    - Complejo: 8 o+ escenarios claves
- Calcular UCP no ajustados(UUCP) el factor de ajuste y UCP ajustados (AUCP)
    - Sumar todos los puntos
    - Multiplicar por factor de ajuste técnico y de entorno
- Convertir el toda AUCP en LOE
    - Utilizar la calibración propia de su equipo/organización
    - Usar 20 horas persona por AUCP para comenzar
    - Sum 30 horas persona por aucp

Problemas

- Que es un escenario clave
    - Camino principal que una instancia de un CU puede ejecuta
    - Correspondencia con un flujo alternativo principal
    - Varios flujos alternativos se combinan en un escenario clave
    - Un flujo de excepción particular muy complejo

- Se asume los CU están nivelados
    - Nivel de detalle no demasiado descompuestos, igualmente importancia de un alto nivel
    - CU el sistema, no del negocio
- Los escenarios clave pueden realizar por colaboraciones 7+- 2 clases de análisis
- Evaluación de complejidad de los casos de uso
- Granularidad, que es un CU y hasta donde debe refinarse

El Personal es Decisivo: Experimentos realizados demuestran que el personal tiene diferencias de rendimiento en una proporción de 1:26 o más.

9. GESTIÓN DE CONFIGURACIÓN DE SOFTWARE

CONCEPTOS

Por **cambio** se entiende toda alteración de algún componente del negocio que surge como una necesidad interna, una exigencia del mercado, una obligación legal o una innovación tecnológica

El cambio como problema

- Problema de coordinación entre grupos de trabajo
- Mala comunicación entre áreas comerciales y operativas
- Mala actualización de procesos y documentos
- Gran esfuerzo de retrabajo y gran cantidad de errores
- Baja productividad personal
- Mala calidad frente al cliente

Para entender el cambio es necesario conocer sus particularidades

- Un cambio impacta en más de un componente del negocio
- Un cambio trivial se puede expandir en forma descontrolada
- Un cambio puede tener relación con otros cambios
- Un cambio mal aplicado puede afectar productos no involucrados en su implementación

Origen de cambios: Son muy variados cuales son los orígenes del cambio, pero los fundamentales son:

- Nuevos negocios o condiciones comerciales que dictan los cambios en los requerimientos del producto o en las normas comerciales.
- Nuevas necesidades del cliente que demanda la modificación de los datos, funcionalidades o servicios.
- Reorganización y/o reducción del volumen comercial que provoca cambios en las prioridades del proyecto o en la estructura del equipo.
- Restricciones presupuestarias o de planificación que provocan redefinición del sistema o producto.

Una **configuración** es una combinación de versiones particulares de los componentes que forman un sistema consistente. Desde el punto de vista de evolución, es el conjunto de las versiones de los objetos componentes en un instante dado.

La Primera Ley de la Ingeniería de Sistemas establece: "Sin importar dónde se esté en el ciclo de vida del sistema, el sistema cambiará, y el deseo por cambiar persistirá a lo largo del ciclo de vida."

GESTIÓN DE CONFIGURACIÓN DEL SOFTWARE (SCM)

- Disciplina que se ocupa de identificar la configuración de un sistema en puntos discretos en el tiempo con el propósito de controlar sistemáticamente los cambios a esta configuración y mantener la integridad y traceability de esta configuración a través del ciclo de vida del sistema
- La disciplina dedicada a identificar y documentar las características físicas y funcionales de un ítem de configuración, controlar cambios a esas características, registrar y reportar cambios y estados de la configuración y verificar el cumplimiento de los requerimientos
- Es una actividad de garantía de calidad del software que se aplica en todas las fases del proceso de ingeniería del software.

Propósitos

- Mantener el seguimiento y la integridad de los activos de proyecto, durante su evolución en el proceso de desarrollo de software.
- Se ocupa de identificar los artefactos, versiones y dependencias entre artefactos, así también como la identificación de las configuraciones que forman conjuntos consistentes de artefactos interrelacionados.
- Organiza el trabajo de forma tal de los desarrolladores individuales del equipo no estén constantemente pisándose entre ellos.

Los miembros del equipo de proyecto deben ser capaces de identificar y ubicar los artefactos, seleccionar la versión adecuada de cada uno de ellos, revisar su historia de cambios para entender su estado actual y las razones de los cambios, como así también quien es el responsable actual de los mismos.

Problemas del Desarrollo de Software sin SCM	Importancia del SCM
La versión actual del código se sobrescribe por una anterior	Mejora la productividad
Una actualización crítica se descarta de la versión final	Costos de mantenimiento más bajos
Se hacen cambios a una versión incorrecta del código	Reducción de defectos
Reaparecen errores ya corregidos	Independencia de personas
No se logra determinar qué versiones van en una entrega	
Las construcciones no son reproducibles	
Muchas posibilidades de error cuando se mantienen múltiples versiones	
No hay historia de los cambios	
Los jefes de proyecto no pueden medir el avance	
Pobre comunicación del equipo	

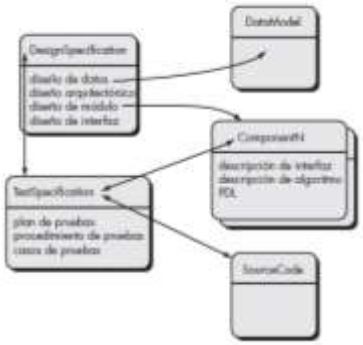
Los req. Del sistema siempre cambian durante su desarrollo y su uso, y se tienen que incorporar estos requerimientos en nuevas versiones del sistema. Es necesario gestionar estos sistemas que evolucionan, porque es fácil perder la pista de los cambios que se han incorporado dentro de cada versión. Las versiones incorporan propuestas de cambios, correcciones de fallos y, adaptaciones para hardware y sistemas operativos diferentes. Pueden existir varias versiones en desarrollo y en uso al mismo tiempo. Si no se tienen unos procedimientos de gestión de configuraciones adecuados, se puede hacer un esfuerzo inútil modificando la versión errónea de un sistema, entregar una versión incorrecta a los clientes o perder la pista de dónde ha sido guardado el código fuente. Los **procedimientos de gestión de configuraciones** definen cómo registrar y procesar los cambios propuestos al sistema, cómo relacionar éstos con los componentes del sistema y los métodos utilizados para identificar las diversas versiones del sistema. Las **herramientas de gestión de configuraciones** se utilizan para almacenar las

versiones de los componentes del sistema, construir sistemas a partir de estos componentes y llevar el registro de entregas de las versiones del sistema a los clientes.
Las principales actividades del proceso de GCS son:

- Identificación de elementos en la configuración del software
- Control de versiones
- Control de cambios
- Auditorías de configuración
- Generación de Informes

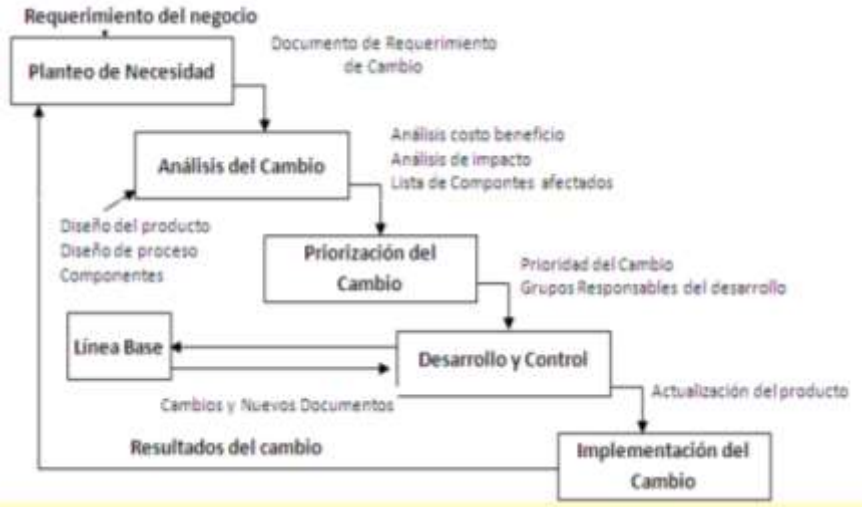
Ítems de configuración del software (ICS): un elemento de información nominado que puede ser tan pequeño como un solo diagrama UML o tan grande como el documento de diseño completo.
Un ICS se puede convertir en un **sistema controlado o línea base**, que es un sistema donde los cambios han sido acordados y registrados formalmente antes de ser implementados. Estos se convierten en el punto de inicio para la evolución controlada.
Eventos que conducen a una línea de referencia: Las tareas de la ingeniería de software producen uno o más ICS. Después de revisar y aprobar los ICS, se colocan en una base de datos del proyecto (librería de proyecto o repositorio de software). Cuando un miembro de un equipo de ingeniería de software quiere hacer una modificación a un ICS que se ha convertido en línea de referencia, se copia de la base de datos del proyecto en el espacio de trabajo privado del ingeniero. Sin embargo, este ICS extraído puede modificarse solamente si se siguen ciertos controles ACS.
Lineas de Referencia:
El cambio es un hecho de vida en el desarrollo de software. Una línea de referencia es un concepto de administración de la configuración del software que le ayuda a controlar el cambio sin impedir seriamente cambios justificados. El IEEE define una línea de referencia como: Una especificación o producto que se revisó formalmente y con el que se estuvo de acuerdo, que a partir de entonces sirve como base para un mayor desarrollo y que puede cambiar sólo a través de procedimientos de control de cambio formal.
Antes de que un ítem de configuración del software se convierta en línea de referencia, los cambios pueden realizarse rápida e informalmente. No obstante, una vez establecida la línea de referencia, pueden realizarse cambios, pero debe aplicarse un procedimiento formal específico para evaluar y verificar cada uno de ellos.
Objetivos de la línea base:

- Sin confusión: En la base de componentes del software
- Sin Ambigüedad: Provee el estándar oficial sobre el que se basa todo trabajo posterior
- Esta controlada: Solo se pueden realizar sobre ella cambios autorizados



Los ICS se organizan para formar objetos de configuración que puedan catalogarse con un solo nombre en la base de datos del proyecto. **Un objeto de configuración** tiene un nombre y atributos, y está "conectado" con otros objetos mediante relaciones. Por ejemplo en la figura, los objetos de configuración DesignSpecification (especificación de diseño), DataModel (modelode datos), ComponentN (componente n), SourceCode (código fuente) y TestSpecification (especificación de prueba) se definen cada uno por separado. Sin embargo, cada uno de los objetos se relaciona con los demás, como se muestra mediante las flechas. Una flecha curva indica una relación composicional, es decir, DataModel y ComponentN son parte del objeto DesignSpecification. Una flecha con doble punta indica una interrelación. Si se realizara un cambio al objeto SourceCode, las interrelaciones permiten determinar qué otros objetos (e ICS) pueden resultar afectados.

La base de datos de configuraciones o repositorio: Se utiliza para registrar toda la información relacionada con las configuraciones y sus elementos. Sus funciones principales son ayudar a la evaluación del impacto de los cambios en el sistema y proveer información de la gestión acerca del proceso de la CM. No incluye información acerca de los elementos de configuración, sino que se define como un metamodelo.
Administración del cambio



-Planteo de necesidad: Transforma un requerimiento del negocio en un planteo formal de Cambio. El documento de necesidad debe poseer:

- Objetivo del requerimiento
- Área productora
- Producto/s afectado/s
- Detalle del cambio a realizar
- Beneficio esperados por el cambio
- Tiempo de liberación al mercado.

-Análisis de cambio: En base al documento de necesidad, se elabora;

- Análisis costo-beneficio.
- Análisis de la relación del cambio con otros en ejecución o espera de desarrollo.
- Un plan preliminar de trabajo
- Análisis de impacto del cambio: es el conjunto de actividades que permiten determinar los efectos de un cambio en los componentes de un producto, proceso u organización (Componentes del negocio afectados y esfuerzo estimado de desarrollo). Se evalúan los posibles riesgos asociados con un cambio a un determinado conjunto de componentes del negocio, incluyendo las estimaciones de sus efectos en recursos, tiempos y actividades. Para entender y hacer un efectivo análisis de impacto es necesario:
  - Disponer de un proceso documentado para el desarrollo de Productos
  - Disponer de los procesos y productos elaborados bien documentados y estandarizados
  - Establecer interrelaciones entre cada uno de los componentes del negocio
  - Es fundamental definir un documento que contenga:
  - Identificación del cambio
  - Descripción del cambio
  - Lista de componentes afectadas
  - Lista de las partes de cada componente afectado
  - Ponderación del impacto en cada componente,
  - Entre otras.

-Priorización del cambio: Esta actividad, en base al resultado del Análisis del Cambio, asigna una prioridad, y asigna los recursos necesarios para su desarrollo.

- Se deben ajustar planes y lista de cambios
- Esta actividad se puede integrar con el Análisis del Cambio.
- Facilita las actividades de coordinación y uso de los recursos

-Desarrollo del control

- Plasma el cambio dentro de cada uno de los componentes afectados, a partir de diseñar una solución adecuada.

Montenegro, Micaela Soledad

- Produce todos los materiales necesarios para efectuar revisiones del diseño del cambio.
- Genera la nueva versión de cada documento, producto y proceso.
- Arma el plan de implantación de los cambios.

-Los procedimientos de control de los componentes deben asegurar:
- Que un componente particular afectado por el cambio este en un solo lugar a la vez
- Que se esté trabajando con la versión correcta
- La identificación de la versión necesaria para el cambio
- La facilidad de almacenamiento de cada componente
- El control de calidad debe asegurar:
  - Que el cambio satisface los requerimientos
  - Que el plan de desarrollo se cumple dentro de los presupuestos
  - Que se respeten todos los estándares del proceso
  - Que los componentes afectados están bajo control

-Implementación del cambio
- Instala el cambio en la organización
- Capacita a los recursos afectados por el cambio
- Renueve documentos obsoletos
- Apoya la instalación de tecnología
- Monitoree el nuevo proceso
- Recopila las sugerencias y la satisfacción del cliente

PLANIFICACIÓN DE LA GESTIÓN DE LA CONFIGURACIÓN DE SOFTWARE

Un **plan de SCM** describe los estándares y procedimientos utilizados para la gestión de la configuración. Se documenta **que actividades** de SCM van a ser llevadas a cabo, **quiénes** son responsables de realizar las distintas actividades, **cuándo** van a suceder, y qué **recursos** son requeridos.
- El plan de SCM describe las **políticas y prácticas** a ser utilizadas en el proyecto para implementar el SCM.
- El plan de SCM define las reglas y responsabilidades.
- El plan de SCM es parte de plan de desarrollo de software.
- No existen reglas estrictas: Hay que discutir qué es lo que mejor se ajusta en general, y para cada producto en particular.

El punto de inicio para desarrollar el plan es un conjunto de estándares generales de gestión de la configuración de toda la compañía adaptables a cada proyecto específico. **El plan de la CM se organiza en varios capítulos que incluyen:**
- La definición de **lo que se debe gestionar** (los elementos de configuración) y el esquema formal para **identificar estas entidades**.
- Un enunciado de **quién toma la responsabilidad** de los procedimientos de gestión de configuraciones y quién envía las entidades controladas al equipo de gestión de configuraciones.
- **Las políticas** de gestión de configuraciones utilizadas para gestionar el control de los cambios y las versiones.
- Una **descripción de las herramientas a utilizar** para la gestión de configuraciones y el proceso a aplicar cuando se utilizan estas herramientas.
- Una **definición de la base de datos de la configuración** que se utilizará para registrar la información de la configuración.

En el plan de la CM se incluye información adicional de la gestión del software por parte de los proveedores externos y los procesos de auditoría para el proceso de la CM.

ELEMENTOS DE CONFIGURACIÓN DE SOFTWARE

Identificación de objetos de configuración de software

Durante el proceso de planificación de la gestión de configuraciones, se decide exactamente qué elementos (o clases de elementos) se van a controlar. Los documentos o grupos de documentos relacionados del control de la

configuración **son documentos formales o elementos de la configuración**. Normalmente, **los planes del proyecto, las especificaciones, los diseños, los programas y los conjuntos de datos de prueba son elementos de la configuración**. Todos los documentos que son necesarios para el mantenimiento futuro del sistema deben ser controlados por el sistema de control de configuraciones.

Sin embargo, esto **no** significa que **todos los documentos o archivos deban estar bajo el control de configuraciones**. Documentos como, por ejemplo, documentos técnicos de trabajo que presentan la captura de ideas para el posterior desarrollo, minutas de las reuniones de grupo, bosquejo del plan y propuestas, no tendrán relevancia a largo plazo y no serán necesarios para el futuro mantenimiento del sistema.

El **esquema de asignación de nombres** a los documentos debe asignar un nombre único a todos los documentos de control de la configuración. Este nombre debe reflejar el tipo de elemento, la parte del sistema en la que se utiliza y el creador del elemento, entre otros. En su esquema de nombres, también deberá reflejar las relaciones entre elementos para asegurar que los documentos relacionados tengan una raíz común de su nombre. Esto conduce a un esquema de asignación de nombres jerárquico.

Tipos de objetos

- Elementos básicos (un listado fuente de un módulo, un conjunto de casos de prueba que se usan para probar el código)
- Elementos compuestos (colección de objetos básicos y otros objetos compuestos)

Control de versiones

La gestión de las versiones es el proceso de identificar y mantener los registros de las diversas versiones de un sistema. Los gestores de las versiones diseñan procedimientos para asegurar que las diversas versiones de un sistema se puedan recuperar cuando se requieran y que no se cambien de forma accidental por parte del equipo de desarrollo.

Una **versión** de un sistema es una instancia de un sistema que difiere, de alguna manera, de otras instancias. Las nuevas versiones de un sistema tienen diferente funcionalidad, mejor rendimiento o incorporan reparaciones de los fallos del sistema. Algunas versiones son funcionalmente equivalentes pero diseñadas para diferentes configuraciones de hardware y software. Si sólo existen pequeñas diferencias entre las versiones, éstas se denominan **variantes**. Una entrega de un sistema es una versión que se distribuye a los clientes.

Control de cambios

Un cambio es el paso de una versión de la línea base a la siguiente. Puede incluir modificaciones del contenido de algún componente, y/o modificaciones de la estructura del sistema, añadiendo o eliminando componentes.

En un proyecto de desarrollo de software, el cambio no controlado lleva rápidamente al caos.

El control de cambios incluye:

- Control de acceso: gobierna los derechos de los ingenieros del software a acceder y modificar objetos de configuración concretos.
- Control de sincronización: asegura que los cambios en paralelo, realizados por personas diferentes, no se sobreescriben mutuamente.

Desplegar ("Check-out"): Un despliegue crea una copia de trabajo local desde el repositorio. Se puede especificar una revisión concreta, y por defecto se suele obtener la última.

"Publicar" o "Enviar" ("commit", "check-in"): Un **commit** sucede cuando una copia de los cambios hechos a una copia local es escrita o integrada sobre repositorio.

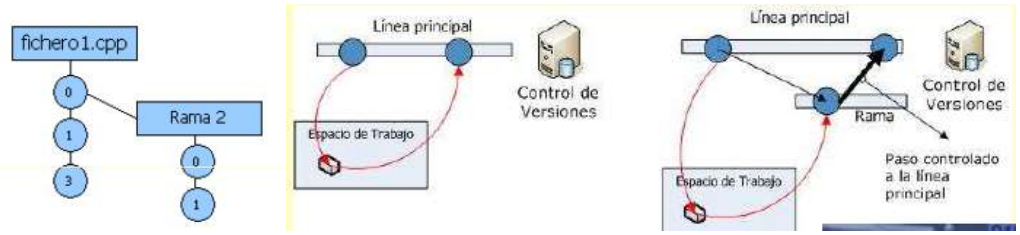
Actualización ("sync" ó "update", "rebase"): Una **actualización** integra los cambios que han sido hechos en el repositorio (por ejemplo por otras personas) en la **copia de trabajo** local.

Abrir rama ("branch") o ramificar: Un módulo puede ser **branched** o **bifurcado** en un instante de tiempo de forma que, desde ese momento en adelante se tienen dos copias (ramas) que evolucionan de forma independiente siguiendo su propia línea de desarrollo. El módulo tiene entonces 2 (o más) "ramas". Las ramas también son conocidas como "lineas de desarrollo". Incluso cuando un proyecto no tiene ramas específicas se considera que el desarrollo se esta produciendo en la rama principal, también conocida como "línea primaria" o "trunk".

Rotular ("tag","etiqueta"): Los tags permiten identificar de forma fácil revisiones importantes en el proyecto.

Integración (Merge): Mover un cambio de una rama a otra, lo que incluye unir desde la rama principal a otra rama o vice versa. También tiene otro significado: es lo que hace el sistema de control de versiones cuando se encuentra con que dos personas han realizado cambios en un mismo fichero sin relación alguna. Ya que estos cambios no interfieren entre ellos, cuando alguna de estas personas actualiza su copia del fichero (el cual ya contiene los

cambios) los cambios de la otra persona serán unidos automáticamente. Cuando dos cambios diferentes están relacionados, el resultado es un "conflicto".



Patrones de creación de ramas

Tipos de patrones

- Patrones de política de trabajo (cuando hacer check-in, propagar)
- Patrones de creación de ramas (subproyecto)
- Patrones de estructuración (rama para desarrollo externo)

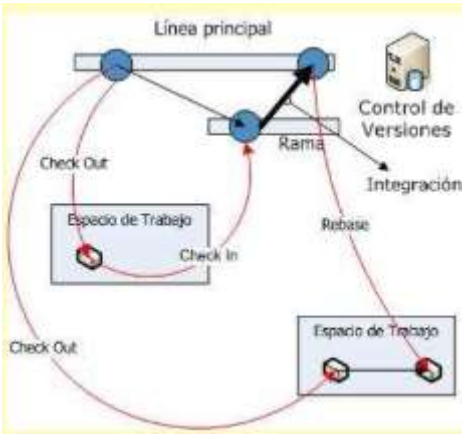
Ejemplos de patrones

- Rama de proyecto
- Rama de mantenimiento
- Ramificar por tarea. Ventajas:
  - Mayor paralelismo
  - Productividad
  - Blindaje de la estabilidad del producto
  - Trazabilidad total
- Soporte de diferentes plataformas
- Rama de desarrollo mayor: ventajas:
  - Mejora el aislamiento
  - Programadores con un *servicio* adicional
  - Mayor control

Dimensiones de branching

- Física: se ramifican ficheros, componentes y subsistemas
- Funcional: para unidades entregables (correcciones, construcciones)
- De entorno: para distintas plataformas, sistemas operativos
- Organizativa: estructurar el trabajo, coordinar
- Procedimental: comportamiento del equipo

AUDITORIA DE CONFIGURACIÓN



¿Cómo podemos asegurar que el cambio se ha implementado correctamente?

1- Revisiones técnicas formales: se centran en la corrección técnica del elemento de configuración que ha sido modificado.
2- Auditorias de configuración del software**:** complementa la revisión técnica formal

OBJETIVO Verificar que el producto de SW integrado satisface los requerimientos estándares o acuerdos contractuales y que los componentes que se integran corresponden con las versiones vigentes.
META Verificar que todos los productos de SW han sido producidos descritos e identificados correctamente y que todas las solicitudes de cambio han sido procesadas

REPORTES O INFORMES DE ESTADO

Se generan Informes de Estado cada vez que:

- Se asigna una nueva identificación a un elemento
- La Autoridad de Control de Cambios aprueba un cambio
- Se lleva a cabo una auditoría
- Regularmente, a fin de mantener al equipo al tanto de los cambios importantes

Responde a:

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

Todo release cae dentro de una de estas dos categorías, dependiendo del tipo de modificaciones realizadas:

- **Release mayor:** Implica un cambio de arquitectura y/o una modificación mayor en un conjunto de procesos o funcionalidades.
- **Release menor*:*** Modificación en un componente o funcionalidad particular.

Cualquiera de las categorías anteriores puede incluir además ***bug fixing*** (corrección de errores). Una corrección de errores siempre se considera una modificación menor.
Se considera un tipo de release denominado **parche**, cuyo objetivo es introducir en la release actual la corrección de errores sin la necesidad de generar una nueva release. En general, los parches se incluyen de forma automática en la release inmediata posterior.
Dado que esta clasificación **no es formal**, la categorización de una release debe ser consensuada y analizada.
La nomenclatura de los releases será la siguiente: ***Rmm-nn*** donde:
R: es una letra fija. Indica que se trata de un release.
mm: Número de release mayor. Se utiliza numeración sucesiva a partir de 01, siempre con dos dígitos.
Nn: Número de modificación menor. Se utiliza numeración sucesiva a partir de 00, siempre con dos dígitos.
Ejemplo: R01-00 R01.01 R01.02 R02.00

AMBIENTE DE DESARROLLO

- El ambiente de desarrollo es el ámbito de codificación, compilación y test unitario. Es propiedad exclusiva del desarrollador.
- La creación del ambiente de desarrollo es responsabilidad del desarrollador.
- Un ambiente de desarrollo debe ser inicializado con la correspondiente versión de release que será modificada.
- Todo componente a modificar debe ser previamente reservado mediante la operación de check-out y descargado sobre el ambiente de desarrollo.

AMBIENTE DE INTEGRACIÓN

- Es donde el responsable de realizar la integración (en gral. Es mismo desarrollador) recompila todos los componentes modificados que formarán parte de una entrega de software (generalmente una nueva release).
- El resultado de esta compilación será probada en integración y posteriormente entregada al cliente.
- Es condición necesaria para que un componente sea incluido en el test de integración, que esté incluido en la herramienta de control de versiones (CVS).
- El conjunto de archivos/revisiones que forman parte cada modificación, debe quedar registrado.

10. VERIFICACIÓN Y VALIDACIÓN

Durante y después del proceso del proceso de implementación, el programa que se está desarrollando debe ser comprobado para asegurar que satisface su especificación y entrega la funcionalidad especificada por las personas que pagan por el software. Verificación y validación (V & V) es el proceso de verificar que un sistema de software cumple con las especificaciones y que cumple su propósito, y tienen lugar en cada etapa del proceso de software.

La verificación y la validación no son lo mismo:

- Validación: ¿Estamos construyendo el producto correcto?La validación tiene como objetivo asegurar que el sistema de software satisface las expectativas del cliente.
    - Evalúa un componente al final de cada fase del desarrollo del software.
    - Se realiza durante cada fase el desarrollo de sw.
- Verificación: ¿Estamos construyendo el producto correctamente? La verificación implica comprobar que el software está de acuerdo con su especificación, es decir que satisface sus requerimientos funcionales y no funcionales.
    - Típicamente incluye el testeo, y tiene lugar luego que las verificaciones han sido completadas.
    - Ocurre al final con las pruebas de aceptación del cliente.

De acuerdo con el Modelo de Madurez de Capacidades (CMMI-SW v1.1),

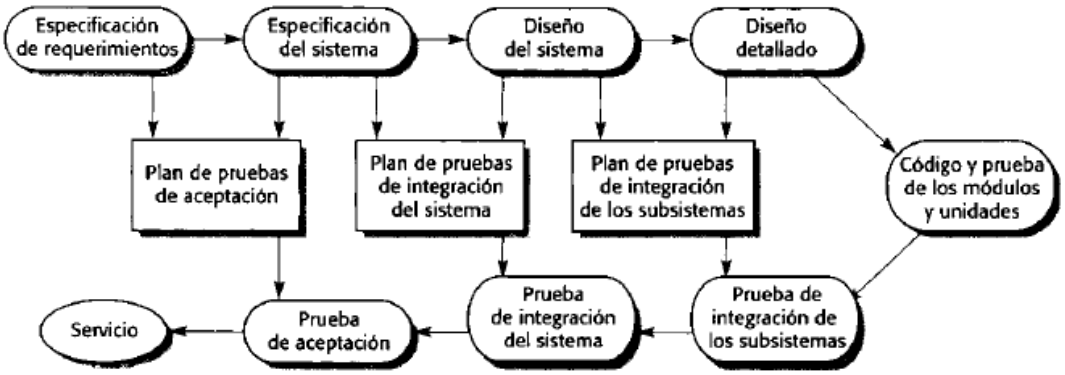
- Verificación: El proceso de evaluación de software para determinar si los productos de una determinada fase de desarrollo cumplan las condiciones impuestas en el inicio de esa fase.
- Validación: El proceso de evaluación de software durante o al final del proceso de desarrollo para determinar si satisface los requisitos especificados.

En otras palabras, la validación asegura que el producto realmente cumple con las necesidades del usuario, y que las especificaciones eran correctas, en primer lugar, mientras que la verificación es garantizar que el producto ha sido construido de acuerdo con los requisitos y especificaciones de diseño. La validación asegura que "se construyó lo correcto". Verificación asegura que "se construyó bien". Validación confirma que el producto, como proporcionado, cumplirá con su uso previsto.

PROCESO, PLANEAMIENTO Y ESTRATEGIAS TESTING

La verificación y validación es un proceso caro, por ello es necesario una planificación cuidadosa.

Debería comenzarse la planificación de la verificación y validación del sistema en etapas tempranas del proceso de desarrollo. El modelo de proceso de desarrollo del software se denomina a veces modelo V, y muestra que los planes de pruebas deberías derivarse a partir de las especificación y diseño del sistema. Este modelo también divide la V & V del sistema en varias etapas cada una de las cuales esta conducida por pruebas que tienen que definirse para comprobar la conformidad del programa con su diseño y especificación.



La planificación de las pruebas está relacionada con el establecimiento de estándares para el proceso de las pruebas, no sólo con la descripción de los productos de prueba. Los planes de prueba ademas de ayudar a los gestores a asignar recursos y estimar el calendario de rpuebas, ayudan en el diseño y en la realización de las pruebas de sistema.

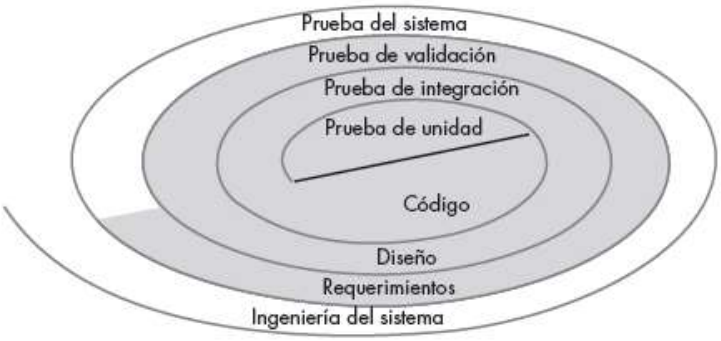
Los principales componentes de un plan de pruebas para un sistema grande y complekp son:

- Proceso de prueba: descripción de las principales fases del proceso de prueba.
- Trazabilidad de los requerimientos: Las pruebas deberían planificarse para que todos los requerimientos se prueben individualmente.
- Elementos probados: Especificación de los elementos del proceso de software que tienen que ser probados.
- Calendario de pruebas: Calendario de todas las pruebas y la asignación de recursos para este calendario se enlaza con la agenda del desarrollo del proyecto.
- Procedimientos de registros de prubas: Los resultados de las pruebas deben ser registrados sistemáticamente.
- Requerimientos de hardware y software: Determinación de las herramientas de software requeridas y la utilización estimada del hardware.
- Restricciones: Anticipación de las restricciones que afectan al proceso de pruebas.

Los planes de pruebas no son documentos estáticos, sino que evolucionan durante el proceso de desarrollo (Ej. Cambian debido a retrasos en otras etapas del proceso de desarrollo).

Estrategia de prueba del software

El proceso de software puede verse como la espiral. Inicialmente, la ingeniería de sistemas define el papel del software y conduce al análisis de los requerimientos del mismo. Al avanzar hacia adentro a lo largo de la espiral, se llega al diseño y finalmente a la codificación. Para desarrollar software de computadoras, se avanza en espiral hacia adentro a lo largo de una línea que reduce el nivel de abstracción en cada vuelta.



Una estrategia para probar el software también puede verse en el contexto de la espiral.

Existen 4 niveles de pruebas

- Pruebas de Unidad: Garantiza que cada componente funciona adecuadamente de forma individual, es decir como unidad. Estas pruebas enfocan los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software. Las pruebas de unidad se enfocan en la lógica de procesamiento interno y de las estructuras de datos dentro de las fronteras de un componente. Este tipo de pruebas puede realizarse en paralelo para múltiples componentes.
  A continuación, los componentes deben ensamblarse o integrarse para formar el paquete de software completo.
- Pruebas de Integración: Consiste en probar el sistema resultante de la integración de sus componentes para encontrar problemas debidos a dicha integración. Procura exponer fallas en la integración a nivel de módulos o fallas en las interfaces y la interacción entre sistemas. Tiene en cuenta :
  - La interacción aumenta la complejidad.
  - Convivencia entre los módulos.
  - ¿Interfiere con otros sistemas?
  - ¿Cómo respondería en un entorno de Producción real?
  - ¿Qué plataformas soporta? ¿Cómo responde en cada una? ¿Cómo se comunican entre ellas?
  - Considerar la coordinación de cada cambio
- Pruebas de Sistema: verifica que todos los elementos se mezclan de manera adecuada y que se logra el funcionamiento/rendimiento global del sistema. Implican integrar dos o más componentes que implementan funciones del sistema o características y a continuación se preuba este sistema integrado. Se ocupan de probar el sistema completo, o un incremento a ser entregado al cliente en el caso de un enfoque iterativo.
- Pruebas de Aceptación de Usuario: Asegura que el sistema cumpla con los requerimientos de negocio y, consecuentemente, que la lógica del mismo funcione correctamente.

Pruebas

Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo.

El objetivo de las pruebas es presentar información sobre la calidad del producto a las personas responsables de este.

Tipos de pruebas: Es un grupo de actividades de prueba dirigidas a probar un componente o un sistema con un objetivo de prueba específico. Los tipos de pruebas buscan evaluar alguna de los atributos de calidad delproducto de software. Ejemplos de tipos de pruebas son:

- Usabilidad
- Mantenimiento
- Funcionales
- Confiabilidad

Montenegro, Micaela Soledad

- Performance
- Escalabilidad
- Carga
- Utilización de recursos
- Estrés
- Instalación
- Portabilidad

Pruebas de Funcionalidad: Verifican la capacidad del producto de software para proveer funciones que cumplan con las necesidades, establecidas e implícitas, cuando el software es utilizado en condiciones normales a cualquier nivel del sistema. Las pruebas se realizan en base al análisis de la especificación de la funcionalidad del componente o sistema.

Las pruebas consisten en accionar(incluso de forma inválida) con el objeto de prueba y verificar que las respuestas obtenidas del mismo sean las correctas. Evalúan las siguientes características del objeto de prueba:

- Precisión: las respuestas esperadas cumplan el grado de precisión necesario.
- Interoperabilidad: pueda interactuar con uno o más componentes o sistemas necesarios.
- Seguridad: impida el acceso no autorizado, accidental o deliberado, a funcionalidad o datos del mismo.
- Adecuación: provea el conjunto apropiado de funcionalidad para la tarea especificada y los objetivos de su usuario.

Pruebas de Confiabilidad: •Tienen por objetivo medir la madurez del software durante eltiempo y compararla con la confiabilidad deseada. Las tomadas incluyen tiempo medio entre fallas (MTBF) y tiempo medio de reparación (MTTR)

Las pruebas toman un tiempo considerable para que sean significativas y suelen continuarse en producción.

Estas pruebas también se especifican en términos de perfiles operacionales

- Prueba de robustez: Orientadas a evaluar la tolerancia de un componente o sistema a fallas que ocurren fuera del objeto de prueba.
- Prueba de recuperación: Orientadas a evaluar la habilidad del sistema de software a recuperarse de fallas en hardware o software del modo predeterminado que permite reanudar las operaciones normales.

Pruebas de Performance: Pruebas enfocadas en la habilidad del componente o sistema de responder al usuario o sistema dentro del tiempo establecido y bajo las condiciones establecidas. Las medidas pueden variar de ciclos de CPU a tiempo de respuesta.

Pruebas de Carga: Enfocadas en la habilidad del sistema de manejar crecientes niveles de carga real y previsible resultante de pedidos de transaciones generadas por usuarios en simultaneo. Las pruebas pueden conducirse con un número realista de ususarios(multiusuario) o con una gran cantidad de ellos (volumen). Las medidas deben realizarse sobre tiempo de respuesta promedio y tasa de transferencia de información.

- Fuertes cargas de trabajo como tráfico excesivo o cargas elevadas detransacciones simultáneas.
- Tráfico de red artificial.
- Múltiples usuarios simultáneos.
- Generar transacciones con herramientas de automatización.

Pruebas de Estrés: Enfocadas en la habilidad del sistema de manejar picos de carga en el límite o superiores de su capacidad máxima. La performance del sistema debe degradarse lentamente y de forma predecible sin fallas a medida que los niveles de estrés aumentan.

Las pruebas deben verificar que el sistema no pierda su integridad funcional mientras este se encuentra en condiciones de stress.

- Encontrar errores debidos a la escasez de recursos

- Falta de memoria
- Falta de espacio en disco
- Encontrar errores debidos a la existencia de recursos compartidos
- Recursos del sistema
- Bloqueos de la base de datos
- Ancho de banda de la red

¿Cómo deben hacerse las pruebas?

- Evitar pruebas espontáneas y que no dejan registros.
- Sobre la base de una metodología formal fácilmente repetible en múltiples circunstancias
- Proveyendo un encuadre formal para todas las actividades de pruebas
- Midiendo exhaustivamente
- Ejecutándolo en paralelo al ciclo de desarrollo de sistemas, empezando lo más tempranamente posible.

¿Qué pasa si NO hay pruebas sistemáticas?

- No hay una identificación adecuada de los requerimientos de pruebas.
- Probable indefinición respecto de cuán completas son pruebas.
- Desconocimiento de la tasa actual de errores.
- No podemos decir cuando liberar un producto.

Requerimiento de Prueba

Es un aspecto o un evento de un componente o sistema que pudiera ser verificado por uno o más casos de pruebas. Ej. Función, transacción, característica, o elementos estructurales.

¿Qué debe contener una Metodología de Pruebas?

- Identificación de requerimientos de Prueba
- Planeamiento de la prueba
- Ejecución de la prueba
- Clasificación de los errores
- Mediciones
- Seguimiento de los errores
- Administración del proyecto de pruebas

Prueba habitual vs. Prueba sistemática

- En la prueba habitual, el objetivo dominante es demostrar que el sistema funciona. “El camino Feliz”
- La prueba sistemática, por el contrario, se focaliza en la detección e identificación de defectos del sistema

Las pruebas sistemáticas requieren:

- Independencia: El Tester debe estar comprometido con la búsqueda de errores y no con la defensa del producto.
- Idoneidad: El Tester debe ser un profesional preparado en las funciones de probar y con los conocimientos necesarios de la metodología de pruebas y de las técnicas y métodos de prueba.

CASO DE PRUEBA

Un caso de prueba es un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y pos condiciones de ejecución, desarrollados para con un objetivo en particular, tal como ejercitar un camino particular de un programa o verificar el cumplimiento de un requerimiento específico.

- Precondición: Condiciones de entorno y de estado, las cuales deben ser cumplidas antes de que el componente o sistema puede ser ejecutado con un valor particular. Los datos de prueba son aquellos que deben existir antes de que una prueba pueda ser ejecutada y que afectan o son afectados por el

componente o sistema bajo prueba. El entorno de prueba son elementos diferentes del objeto de prueba que son necesarios para conducir la prueba. Ej. Hardware, Herramienta de Testing, etc.
- Entrada de Prueba: El dato recibido por un objeto a prueba, durante la ejecución de la prueba desde una fuente externa la cual puede ser HW, SW o una acción humana.
- Resultado Esperado: El Comportamiento predicho la especificación u otra fuente, del objeto de prueba, bajo condiciones específicas. Se debe también especificar cómo se obtiene el resultado esperado.
- Oráculo: Es una fuente para determinar resultados esperados de un caso de prueba. Un oráculo puede ser un sistema existente, un usuario manual, o el conocimiento especializado de un individuo. Nunca debería ser el código. Ej. Simulación, experto en el dominio.
- Criterio para el Veredicto pasa /falla: Reglas de decisión usadas para determinar si un ítem de prueba ha pasado o fallado una prueba
- Pos condición: Condiciones del entorno y estado que se deben cumplimentar después de la ejecución de una prueba o un caso de prueba, o un procedimiento de prueba.

El Resultado obtenido por el caso de prueba es s el comportamiento producido/observado de un objeto de prueba como resultado del procesamiento de entradas de prueba. Esto no es parte del caso de prueba, sino el resultado de la ejecución de la prueba, sobre una versión específica del objeto a prueba. En base a este resultado se da el veredicto de si la prueba pasó o falló.

El caso de prueba tiene como objetivo: "descubrir errores", como criterio: "en forma completa" y como restricción: "con el mínimo de esfuerzo y tiempo"

Tipos de casos de prueba

- Caso de prueba a alto nivel: Es un caso de prueba que no tiene una implementación concreta, o valores concretos para la entrada de datos y el resultado esperado
- Caso de prueba a bajo nivel: Es un caso de Prueba con valores concretos, que existen a nivel de implementación, para dato de entrada y resultado esperado. Los operadores lógicos de los casos de prueba a alto nivel son reemplazados por valores reales que corresponden a el objetivo de los operadores lógicos.

Nivel de detalle de especificación de un caso de prueba

- Es una decisión importante que se debe tomar cuando se piensa documentar casos de pruebas.
- Qué tan preciso tiene que ser especificadas las acciones, los datos, el resultado esperado, etc.?
- Mejora la reproducibilidad ya que nada es dejado al juicio de un tester en particular.
- Permite que los casos de prueba pueden ser revisados por desarrollo
- Implica mucho más trabajo en la especificación
- Permite que testers no expertos ejecuten los casos de prueba.

Conjunto de pruebas ideal

Es el menor subconjunto de todos los casos de prueba posibles que encuentra todos los defectos sobre un componente o sistema. En general, es imposible definir un conjunto ideal de casos de pruebas y pero se trata de aproximar aplicando criterios de selección de pruebas. El criterio de selección de las pruebas trata de aproximar esta noción, eligiendo el subconjunto de comportamientos a probar.

Prueba exitosa

- Aquella que detecta un defecto aún no descubierto.
- Es aquella que encuentra muchos defectos, no lo opuesto.
- Un producto sin defectos daría como resultado que testing no produzca pruebas exitosas y el consiguiente fracaso de testing.

Característica de un buen caso de prueba

- Fácil de repetir
- Teniendo un fijo y preciso punto de inicio. El estado del sistema cuando se introdujo la entrada es conocido.

- Conociendo exactamente que datos de entrada fueron introducidos y en que secuencia.
- Representativo
- Testing exhaustivo no es factible, con lo cual el caso de prueba debería ser representativo de un conjunto posible de casos de prueba.
- Ayuda a identificar fácilmente cual es el defecto.
- Característica de un buen caso de prueba
- Fácil de usar entender o de realizar
- Fácil de mantener
- Trazable

Prueba o conjunto de pruebas

Una prueba es un conjunto de uno o más casos de prueba, mientras que una suite de prueba es un conjunto de varios casos de prueba para un componente o sistema bajo prueba, donde la pos condición de una prueba es frecuentemente usada como la precondición del próximo caso de prueba. Esta última es usada para agrupar casos de prueba similares.

Procedimiento de Prueba

Es un documento especificando una secuencia de acciones para la ejecución de una prueba. También conocido como script de prueba o script manual. Es decir, es una descripción de cómo una prueba es realizada. Contiene acciones, verificaciones (checks) y casos de prueba relacionados e indica la secuencia de ejecución.

Derivación de Casos de Prueba

- En base a documentos del cliente
    - Útil para el diseño de casos de prueba de funciones de negocio.
    - Aconsejable para las pruebas de aceptación de usuario.
    - Puede ser muy pobre para un testing funcional.
- En base a documentos de relevamiento
    - Si se detectan omisiones o supuestos equivocados en el relevamiento, se debe informar para corregir.
    - El diseño de casos de pruebas generado de estos documentos tiene que servir al testing y no hay que confundir con el diseño del desarrollo.
- En base a casos de uso
- En base a especificaciones de programación
    - En área de Testing recibe las mismas especificaciones que se le envían al programador.
    - Si un programador puede codificar en base a estas especificaciones, Testing debe poder diseñar casos de prueba y testear.
    - Más completo que el diseño que se logra en base a documentos del cliente y de relevamiento.
    - Más detalle
    - Se puede lograr un diseño de casos de prueba de amplia cobertura.
    - Sirve para el testing funcional.
    - Si las especificaciones incluyen un prototipo de pantalla, se pueden diseñar casos para pruebas de usabilidad.
- En base a código
    - En Testing se recibe el código.
    - Adquiere las mismas ventajas y desventajas que esta técnica tiene.
    - Es complementario al diseño de casos de prueba en base a documentación.
    - Amplia cobertura del código y por supuesto del diseño.
    - Se usa para el diseño de pruebas unitarias o de componentes.
    - Permite completar los casos de prueba para asegurar una mayor cobertura.

TÉCNICAS DE PRUEBA

Técnicas de diseño de pruebas son:

- Métodos estandarizados para la derivación de casos de prueba.
- Fundamentales para el desarrollo del testing metodológico y profesional.
- Diferentes técnicas se enfocan en encontrar diferentes tipos de defectos
- Es necesario usar una combinación de técnicas para prevenir y encontrar defectos.
- Facilitan comprender la calidad y cobertura de las pruebas

Enfoque recomendado es:

1. Diseñar un conjunto inicial de casos de pruebas con técnicas de caja negra
2. Medir cobertura lógica lograda por las pruebas
3. Diseñar más casos de prueba hasta lograr la cobertura lógica deseada
4. Medir la cobertura estructural lograda sobre el objeto de prueba
5. Diseñar más casos de prueba hasta lograr la cobertura estructural deseada

Cualquier producto sometido a ingeniería pueden probarse en una de dos formas:

- Al conocer la función específica que se asignó a un producto para su realización, pueden llevarse a cabo pruebas que demuestren que cada función es completamente operativa mientras al mismo tiempo se buscan errores en cada función
- Al conocer el funcionamiento interno de un producto, pueden realizarse pruebas para garantizar que las operaciones internas se realizan de acuerdo con las especificaciones y que todos los componentes internos se revisaron de manera adecuada.

El primer enfoque de pruebas considera una visión externa y se llama prueba de caja negra. El segundo requiere una visión interna y se denomina prueba de caja blanca.

Prueba de caja blanca

La prueba de caja blanca también llamadas “técnica estructural”, es una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba.

La prueba de caja blanca del software se basa en el examen cercano de los detalles de procedimiento. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles.

Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que:

- Garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez
- Revisen todas las decisiones lógicas en sus lados verdadero y falso
- Ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas
- Revisen estructuras de datos internas para garantizar su validez

Razones para hacer Testing de Caja Blanca:

- Los errores lógicos y supuestos incorrectos son inversamente proporcionales a la probabilidad de ejecutar un camino en un programa.
- A menudo se cree que un camino lógico probablemente no se ejecutará cuando de hecho puede ser ejecutado en forma regular.
- Los errores tipográficos son aleatorios.

Una de las técnicas de caja blanca es la Prueba de Ruta Básica, y otra es la Prueba de Estructura de control que incluye prueba de condición, prueba de bucles, prueba de flujo de datos.

Prueba de caja negra

Las pruebas de caja negra, también llamadas pruebas de comportamiento, se enfocan en los requerimientos funcionales del software; es decir, las técnicas de prueba de caja negra le permiten derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa. Las

pruebas de caja negra no son una alternativa para las técnicas de caja blanca. En vez de ello, es un enfoque complementario que es probable que descubra una clase de errores diferente que los métodos de caja blanca.

La prueba de caja negra se refiere a las pruebas que se llevan a cabo en la interfaz del software. Una prueba de caja negra examina algunos aspectos fundamentales de un sistema con poca preocupación por la estructura lógica interna del software.

Las pruebas de caja negra intentan encontrar errores en las categorías siguientes:

- Funciones incorrectas o faltantes
- Errores de interfaz
- Errores en las estructuras de datos o en el acceso a bases de datos externas
- Errores de comportamiento o rendimiento
- Errores de inicialización y terminación.

A diferencia de las pruebas de caja blanca, que se realizan tempranamente en el proceso de pruebas, la prueba de caja negra tiende a aplicarse durante las últimas etapas de la prueba. Las pruebas se diseñan para responder a las siguientes preguntas:

- ¿Cómo se prueba la validez funcional?
- ¿Cómo se prueban el comportamiento y el rendimiento del sistema?
- ¿Qué clases de entrada harán buenos casos de prueba?
- ¿El sistema es particularmente sensible a ciertos valores de entrada?
- ¿Cómo se aíslan las fronteras de una clase de datos?
- ¿Qué tasas y volumen de datos puede tolerar el sistema?
- ¿Qué efecto tendrán sobre la operación del sistema algunas combinaciones específicas de datos?

Al aplicar las técnicas de caja negra, se deriva un conjunto de casos de prueba que satisfacen los siguientes criterios:

- Casos de prueba que reducen, por una cuenta que es mayor que uno, el número de casos de prueba adicionales que deben diseñarse para lograr pruebas razonables
- Casos de prueba que dicen algo acerca de la presencia o ausencia de clases de errores, en lugar de un error asociado solamente con la prueba específica a mano.

En este método interesa "el qué" realiza el programa, no "el cómo". El énfasis está en el control de las salidas teniendo en cuenta las entradas.

Los siguientes métodos son comúnmente utilizados: Particionamiento de equivalencias, Análisis de valores frontera y Adivinanza de defectos.

Los siguientes son menos utilizados: Diagrama de causa-efecto y Pruebas de transición de estado.

PRUEBA DEL SISTEMA

El software se incorpora con otros elementos del sistema (por ejemplo, hardware, personas, información), y se lleva a cabo una serie de pruebas de integración y validación del sistema.

Un problema clásico en la prueba del sistema es el “dedo acusador”. Esto ocurre cuando se descubre un error y los desarrolladores de diferentes elementos del sistema se culpan unos a otros por el problema. En lugar de abandonarse a tal sinsentido, deben anticiparse los potenciales problemas de interfaz y:

- Diseñar rutas de manejo de error que prueben toda la información proveniente de otros elementos del sistema
- Realizar una serie de pruebas que simulen los datos malos u otros errores potenciales en la interfaz del software
- Registrar los resultados de las pruebas para usar como “evidencia” si ocurre el dedo acusador

- Participar en planificación y diseño de pruebas del sistema para garantizar que el software se prueba de manera adecuada.

El Testing de Sistema es el proceso de verificar que un sistema integrado cumple con los requerimientos especificados.

Puede estar basado en :

- Requerimientos: surge del documento de requisitos (conviene dedicar tiempo suficiente para que las pruebas aseguren la inclusión correcta)
- Procesos de Negocio: basados en escenarios de negocio, definidos por el usuario (debería acercarse a su uso real)

La prueba del sistema es una serie de diferentes pruebas cuyo propósito principal es ejercitar por completo el sistema basado en computadora. Aunque cada prueba tenga un propósito diferente, todo él funciona para verificar que los elementos del sistema se hayan integrado de manera adecuada y que se realicen las funciones asignadas.

Tipos de pruebas del sistema

- Pruebas de recuperación: Ésta fuerza al software a fallar en varias formas y verifica que la recuperación se realice de manera adecuada.
- Pruebas de seguridad: Intenta verificar que los mecanismos de protección que se construyen en un sistema en realidad lo protegerán de cualquier penetración impropia.
- Pruebas de esfuerzo: Se diseñan para enfrentar los programas con situaciones anormales. En esencia, la persona que realiza las pruebas de esfuerzo pregunta: "¿cuánto podemos doblar esto antes de que se rompa?". La prueba de esfuerzo ejecuta un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales.
- Pruebas de rendimiento: Se diseña para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado.
- Pruebas de despliegue: también llamada prueba de configuración, ejercita el software en cada entorno en el que debe operar. Además, examina todos los procedimientos de instalación y el software de instalación especializado que usarán los clientes, así como toda la documentación que se usará para introducir el software a los usuarios finales.

11. AUDITORIA Y PERITAJE

DEFINICIÓN DE AUDITORÍA

Es un exámen crítico que se realiza con el objeto de evaluar la eficiencia y eficacia de una sección o de un organismo, y determinar cursos alternativos de acción para mejorar la organización y lograr los objetivos propuestos.

- Eficacia: Virtud, actividad, fuerza, para poder obrar. (lograr los objetivos)
- Eficiencia: Virtud y facultad para lograr un efecto determinado.(lograr los objetivos con los menores recursos posibles)

CONTROL INTERNO

El control interno comprende el plan de organización y todos los métodos y procedimientos que en forma coordinada se adoptan en un negocio para salvaguardar sus activos, verificar la razonabilidad y confiabilidad de su información financiera, promover la eficiencia operacional y provocar la adherencia a las polificas prescritas por la administración.

Objetivos básicos

- La protección de los activos de la empresa
- La obtención de información financiera veraz, confiable y oportuna.
- La promoción de la eficiencia en la operación del negocio.
- Lograr que en la ejecución de las operaciones se cumplan las políticas estabecidas por los administradores de la empresa.

Objetivos generales

- Autorización
- Procesamiento y clasificación de las transacciones
- Slvaguarda física
- Verificación y evaluación
- El área de informática puede interactuar de dos maneras en el ámbito del control interno. La primera es servir de herramienta para llevar a cabo un adecuado control interno, y la segunda es tener un control interno del área y del departamento de informática. En el primer caso se lleva el control interno pormedio de a evaluacion de una organización utilizando a computadora como herramienta que auxiliará en el logro de los abjetivos. En el segundo caso se lleva a cabo el control interno de informatica. Es decir se deben proteger adecuadamente los activos de la organización por medio del control para que se obtenga infrmación en forma veraz, oportuna y confiable, para que se mejora la eficiencia de la operación de la organización mediante la informatica y para que en la ejecución de las operaciones de la informática se cumplan las políticas establecidas por la administración.

TIPOS DE AUDITORÍA

Las auditorías se pueden clasificar según diversos criterios:

- Según cuándo se audita:
  - Auditoria Programada
  - Auditoría Extraordinaria
- Según su alcance:
  - Auditoría Parcial
  - Auditoría Global
- Según que se audita:
  - Auditoría Legal
  - Auditoría de un Proceso
  - Auditoría de Sistemas de Gestión
- Según quién audita:
  - Auditoría Interna
  - Auditoría Externa
- Según el área que se audita:
  - Auditoría Financiera
  - Auditoría Administrativa
  - Auditoría Operacional
  - Auditoría Gubernamental
  - Auditoría Integral
  - Auditoría de Sistemas

OBJETIVOS DE AUDITORÍA INFORMÁTICA

Los principales objetivos de la auditoría en informática son:

- Salvaguardar los activos. Se refiere a la protección del hardware, software y recursos humanos.
- Integridad de datos. Los datos deben mantener consistencia y no duplicarse.
- Efectividad de sistemas. Los sistemas deben cumplir con los objetivos de la organización.
- Eficiencia de sistemas. Que se cumplan con los objetivos con los menores recursos.
- Seguridad y confidencialidad.

¿QUÉ ES UNA AUDITORÍA INFORMÁTICA?

Es una función que ha sido desarrollada para asegurar la salvaguarda de los activos de los sistemas de computadoras, mantener la integridad de los datos y lograr los objetivos de la organización en forma eficaz y eficiente.
Auditoría en informática es la verificación de los controles en las siguientes tres áreas de la organización: aplicaciones, desarrollo de sistemas e instalación del centro de proceso.
Por tanto, podemos decir que auditoria en informática es la revisión y evaluación de los controles, sistemas y procedimientos de la informática; de los equipos de computo, su utilización, eficiencia y seguridad; de la organización que participa 1en el procesamiento de la información, a fin de que por medio del señalamiento de cursos alternativos se logre la utilización eficiente, confiable y segura de información que servirá para una adecuada toma de decisiones.
La auditoria en informática deberá comprender la evaluación de los equipos de computos o de un sistema o procedimiento especifico como asi también habrá de evaluar los sistemas de información en general desde su entradas, procedimientos, comunicación, controles, archivos, seguridad, personal y obtención de información.

AUDITORÍA INTERNA Y AUDITORÍA EXTERNA

Las auditorías se clasifican en función de quién audita en auditorías internas y auditorías externas.

- Auditorías Internas: Son las auditorías realizadas por una organización para evaluar su propio rendimiento, se llevan a cabo como consecuencia de una necesidad interna de dicha organización pudiendo ser auditores empleados de la organización, cuyas actividades y grado de adecuación al sistema de gestión de la prevención evalúan. Pueden se realizadas igualmente por auditores externos contratados para tal servicio, sin que ello suponga por dicha externalización que se considere una auditoria externa.
  La auditoria interna pretende verificar los diferentes procedimientos y sistemas de control interno, al objeto de saber si su funcionamiento es el previsto y el establecido. Pretende al mismo tiempo, sugerir cambios posibles o mejoras en los procesos, siendo una herramienta fundamental en el control de las empresas, dependiendo en la mayoría de los casos directamente de la gerencia.
- Auditorías externas: Son las auditorias realizadas por una organización, al objeto de evaluar las actividades de otras organizaciones. Se realizan como información del grado de gestión ante terceros, configurándose como un proceso de verificación de un periodo correspondiente, evaluando la correspondencia o cumplimiento de las disposiciones legales e internas vigentes. Normalmente, suelen llevarse a cabo a requerimiento de organismos oficiales, clientes u organismos de certificación. Ello no supone que puedan ser utilizadas como elemento de gestión por parte de la empresa.

ALCANCE DE LA AUDITORÍA

El campo de acción de la uditoría en informatica es:

- La evaluación administrativa del área de informática
- La evaluación de los sistemas y procedimientos, y de la eficiencia que se tiene en el uso de información. La evalución de la eficiencia y eficacia con la que se trabaja.
- La evaluación del proceso de datos, de los sistemas y de los equipos de cómputo.
- Seguridad y confidencialidad de la información.
- Aspectos legales de los sistemas y de la información.

El alcance ha de definir con precisión el entorno y los límites en que va a desarrollarse la auditoría informática, se complementa con los objetivos de ésta. El alcance ha de figurar expresamente en el Informe Final, de modo que quede perfectamente determinado no solamente hasta que puntos se ha llegado, sino cuales materias fronterizas han sido omitidas. Ejemplo: ¿Se someterán los registros grabados a un control de integridad exhaustivo*? ¿Se comprobará que los controles de validación de errores son adecuados y suficientes*? La indefinición de los alcances de la auditoría compromete el éxito de la misma.

INFORMÁTICA

Ciencia del tratamiento sistemático y eficaz, realizado especialmente mediante máquinas automáticas, de la información contemplada como vehiculo del saber humano y de la comunicación en los ámbitos técnico, económico y social.
Aplicación racional, sistemática de la información para el desarrollo económico, social y político.
Ciencia de los sistemas inteligentes de información.

TIPOS Y CLASES DE AUDITORÍAS

El departamento de Informática posee una actividad proyectada al exterior, al usuario, aunque el "exterior" siga siendo la misma empresa. He aquí, la Auditoría Informática de Usuario. Se hace esta distinción para contraponerla a la informática interna, en donde se hace la informática cotidiana y real. En consecuencia, existe una Auditoría Informática de Actividades Internas.
El control del funcionamiento del departamento de informática con el exterior, con el usuario se realiza por medio de la Dirección. Su figura es importante, en tanto en cuanto es capaz de interpretar las necesidades de la Compañía. Una informática eficiente y eficaz requiere el apoyo continuado de su Dirección frente al "exterior". Revisar estas interrelaciones constituye el objeto de la Auditoría Informática de Dirección. Estas tres auditorías, mas la auditoría de Seguridad, son las cuatro Areas Generales de la Auditoría Informática más importantes.
Dentro de las áreas generales, se establecen las siguientes divisiones de Auditoría Informática: de Explotación, de Sistemas, de Comunicaciones y de Desarrollo de Proyectos. Estas son las Areas Especificas de la Auditoría Informática más importantes.

Areas Específicas	**Areas Generales**			
	Interna	Dirección	Usuario	Seguridad
Explotación				
Desarrollo				
Sistemas				
Comunicaciones				
Seguridad				

Los servicios de auditoría pueden ser de distinta índole:

- Auditoría de seguridad interna. En este tipo de auditoría se contrasta el nivel de seguridad y privacidad de las redes locales y corporativas de carácter interno
- Auditoría de seguridad perimetral. En este tipo de análisis, el perímetro de la red local o corporativa es estudiado y se analiza el grado de seguridad que ofrece en las entradas exteriores.
- Test de intrusión. El test de intrusión es un método de auditoría mediante el cual se intenta acceder a los sistemas, para comprobar el nivel de resistencia a la intrusión no deseada. Es un complemento fundamental para la auditoría perimetral.
- Análisis forense. El análisis forense es una metodología de estudio ideal para el análisis posterior de incidentes, mediante el cual se trata de reconstruir cómo se ha penetrado en el sistema, a la par que se valoran los daños ocasionados. Si los daños han provocado la inoperabilidad del sistema, el análisis se denomina análisis postmortem.
- Auditoría de páginas web. Entendida como el análisis externo de la web, comprobando vulnerabilidades como la inyección de código sql, Verificación de existencia y anulación de posibilidades de Cross Site Scripting (XSS), etc.
- Auditoría de código de aplicaciones. Análisis del código tanto de aplicaciones páginas Web como de cualquier tipo de aplicación, independientemente del lenguaje empleado

Realizar auditorías con cierta frecuencia asegura la integridad de los controles de seguridad aplicados a los sistemas de información. Acciones como el constante cambio en las configuraciones, la instalación de parches, actualización de los softwares y la adquisición de nuevo hardware hacen necesario que los sistemas estén continuamente verificados mediante auditoría.

TÉCNICAS Y HERRAMIENTAS USADAS POR LA AUDITORÍA INFORMÁTICA

- Cuestionarios: Las auditorías informáticas se materializan recabando información y documentación de todo tipo. El trabajo de campo del auditor consiste en lograr toda la información necesaria para la emisión de un juicio global objetivo, siempre amparado en hechos demostrables. Para esto, suele ser lo habitual

comenzar solicitando la cumplimentación de cuestionarios preimpresos que se envían a las personas concretas que el auditor cree adecuadas, sin que sea obligatorio que dichas personas sean las responsables oficiales de las diversas áreas a auditar.

- Entrevistas: La entrevista es una de las actividades personales más importante del auditor; en ellas, éste recoge más información, y mejor matizada, que la proporcionada por medios propios puramente técnicos o por las respuestas escritas a cuestionarios.
- Checklist: El auditor profesional y experto es aquél que reelabora muchas veces sus cuestionarios en función de los escenarios auditados. El auditor conversará y hará preguntas "normales", que en realidad servirán para la cumplimentación sistemática de sus Cuestionarios, de sus Checklists. Salvo excepciones, las Checklists deben ser contestadas oralmente, ya que superan en riqueza y generalización a cualquier otra forma.
- Trazas y/o Huellas: Con frecuencia, el auditor informático debe verificar que los programas, tanto de los Sistemas como de usuario, realizan exactamente las funciones previstas, y no otras. Para ello se apoya en productos Software muy potentes y modulares que, entre otras funciones, rastrean los caminos que siguen los datos a través del programa. Muy especialmente, estas "Trazas" se utilizan para comprobar la ejecución de las validaciones de datos previstas.
- Software de Interrogación: En la actualidad, los productos Software especial para la auditoría informática se orientan principalmente hacia lenguajes que permiten la interrogación de ficheros y bases de datos de la empresa auditada. Estos productos son utilizados solamente por los auditores externos, por cuanto los internos disponen del software nativo propio de la instalación.

Finalmente, ha de indicarse la conveniencia de que el auditor confeccione personalmente determinadas partes del Informe. Para ello, resulta casi imprescindible una cierta soltura en el manejo de Procesadores de Texto, paquetes de Gráficos, Hojas de Cálculo, etc.