



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO



**INGENIERÍA DE
SOFTWARE**

Índice

UNIDAD I. INGENIERÍA DE SOFTWARE

UNIDAD II. PROCESO DE GESTIÓN DEL PROYECTO

UNIDAD III. METODOLOGÍAS

UNIDAD IV. CALIDAD Y NORMAS DE CALIDAD

UNIDAD V. MODELOS DE MADUREZ

UNIDAD VI. TEMAS SELECTOS

UNIDAD I. INGENIERÍA DE SOFTWARE



1.1 Conceptos



1.2 Atributos y características de software



1.3 Importancia y aplicación del software

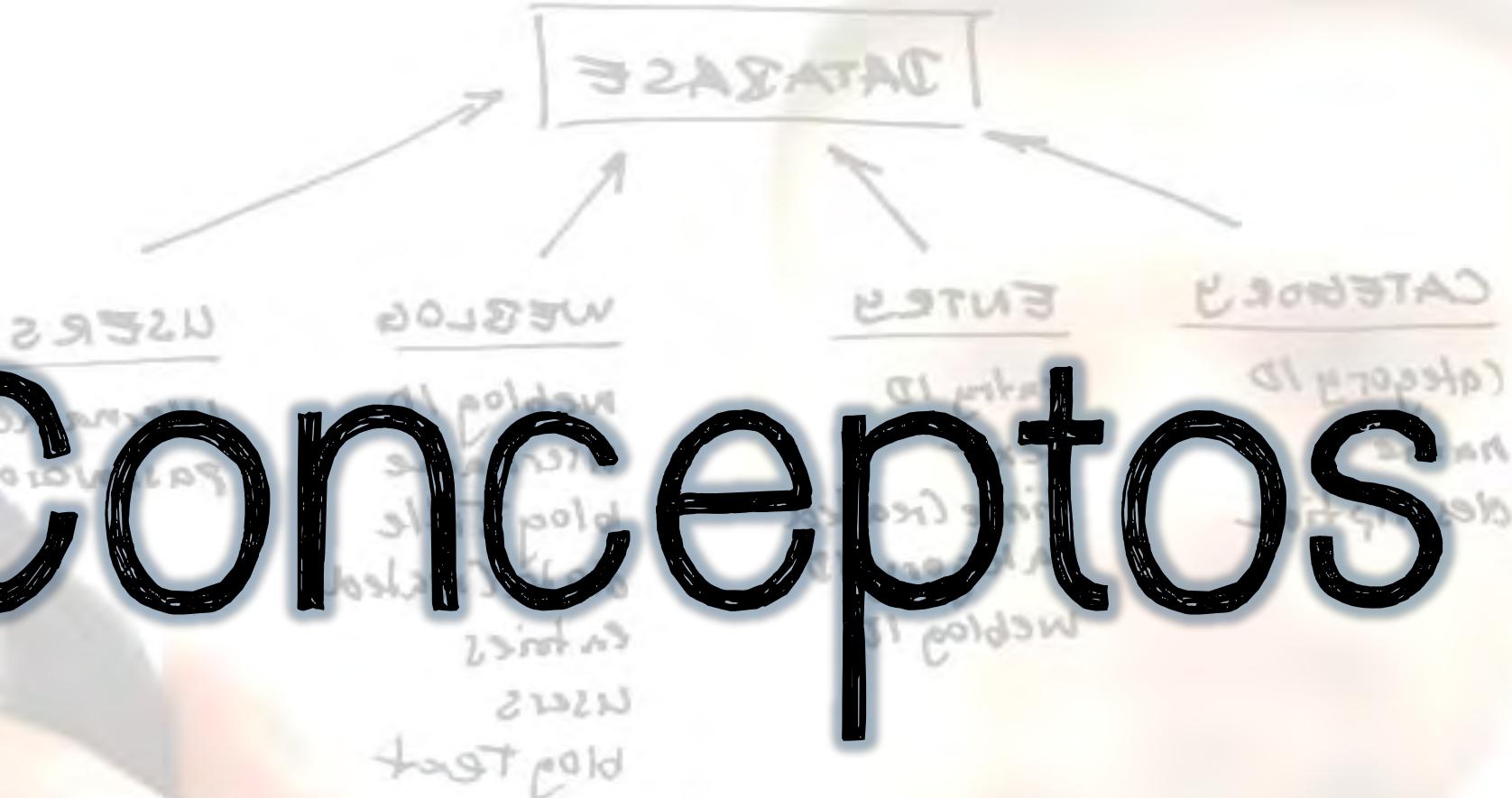


1.4 Ciclo de vida del software



1.5 Modelos de procesos

Conceptos



¿Qué es software?

Programas de cómputo y su documentación asociada: requerimientos, modelos de diseño y manuales de usuario, puede ser creado desarrollando nuevos programas, configurando sistemas de software genérico o reutilizando software existente.

¿Qué es la Ingeniería de Software?

Una disciplina de la Ingeniería que concierne a todos los aspectos de la producción de software, utilizando las herramientas y técnicas apropiadas para resolver el problema planteado, de acuerdo a las restricciones de desarrollo y a los recursos disponibles.

¿Qué es un proceso de software?

Un conjunto estructurado de actividades cuya meta es el desarrollo o evolución de un software

Algunas actividades genéricas en todos los procesos de software son:

Especificación: qué debe hacer el software y cuáles son sus especificaciones de desarrollo.

Desarrollo: producción del sistema de software Validación, verificar que el software cumple con lo solicitado por el cliente.

Evolución: cambiar/adaptar el software a las nuevas demandas.

¿Qué es un modelo de proceso de software?

Representación formal y simplificada de un proceso de software, presentada desde una perspectiva específica

Modelos Genéricos:

- Cascada, separar en distintas fases de especificación y desarrollo.
- Desarrollo Iterativo, la especificación, desarrollo y validación están interrelacionados
- Prototipo, un modelo sirve de prototipo para la construcción del sistema final
- Basado en componentes, asume que partes del sistema ya existen y se enfoca a su integración

¿Cuáles son los costos de la Ingeniería de Software?

El costo total de un software esta dividido aproximadamente de la siguiente forma:

- 60% costos de desarrollo
- 40% costos de pruebas
- Los costos dependen del tipo de sistema que se desarrolla y de los requerimientos del mismo tales como desempeño y confiabilidad, la distribución de los costos depende del modelo de desarrollo empleado.

¿Qué es CASE?

CASE es Computer-Aided Software Engineering son programas que son usados para dar soporte automatizado a las actividades del proceso de software como:

- Las herramientas CASE son comúnmente usadas para dar soporte a los métodos de software
- Módulos de análisis que verifican que las reglas del método se cumplan
- Generadores de reportes que facilitan la creación de la documentación del sistema
- Generadores de código a partir del modelo del sistema

Atributos y características del software

Características.

Para poder comprender lo que es el software (y consecuentemente la ingeniería del software), es importante examinar las características del software que lo diferencian de otras cosas que los hombres pueden construir.

1. El software se desarrolla, no se fabrica en un sentido clásico.

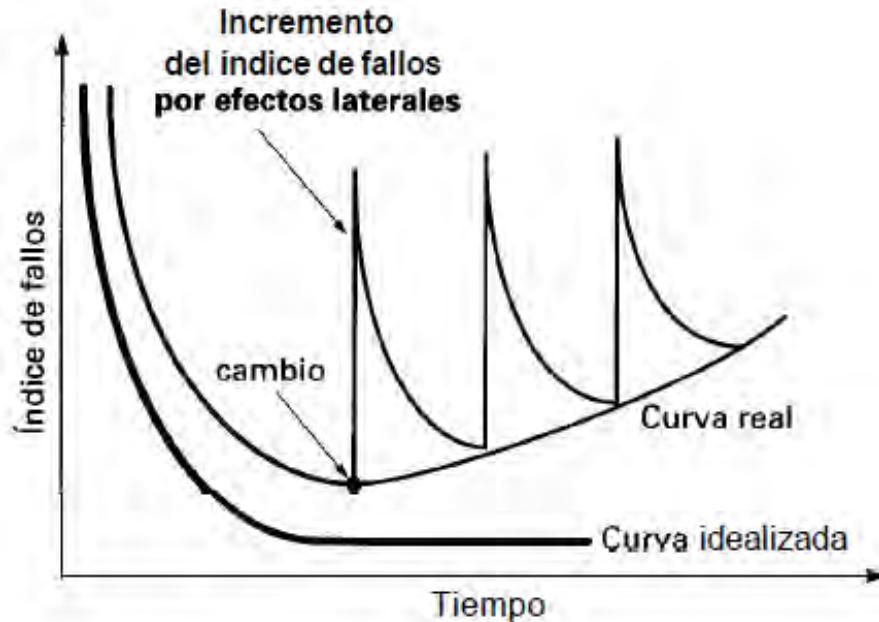
Aunque existen similitudes entre el desarrollo del software y la construcción del hardware, ambas actividades son fundamentalmente diferentes. En ambas actividades la buena calidad se adquiere mediante un buen diseño, pero la fase de construcción del hardware puede introducir problemas de calidad que no existen (o son fácilmente corregibles) en el software.

Ambas actividades dependen de las personas, pero la relación entre las personas dedicadas y el trabajo realizado es completamente diferente para el software. Ambas actividades requieren la construcción de un producto pero los enfoques son diferentes.

Los costes del software se encuentran en la ingeniería esto significa que los proyectos de software no se pueden gestionar como si fueran proyectos de fabricación.

2. El software no se estropea.

El software no es susceptible a los males del entorno que hacen que el hardware se estropee. Por tanto, en teoría, la curva de fallos para el software tendría la forma que muestra la figura. Los defectos no detectados harán que falle el programa durante las primeras etapas de su vida. Sin embargo, una vez que se corrigen (suponiendo que no se introducen nuevos errores) la curva se aplana, como se muestra. La curva idealizada es una gran simplificación de los modelos reales de fallos del software. Sin embargo la implicación es clara, el software no se estropea. ¡Pero se deteriora!



3. Aunque la industria tiende a ensamblar componentes, la mayoría del software se construye a medida.

La mayoría de ocasiones el software es construido a medida pero en los años 60, se construyeron bibliotecas de subrutinas científicas reutilizables en una amplia serie de aplicaciones científicas y de ingeniería. Esas bibliotecas de subrutinas reutilizaban de forma efectiva algoritmos bien definidos, pero tenían un dominio de aplicación limitado.

Los componentes reutilizables modernos encapsulan tanto datos como procesos que se aplican a los datos, permitiendo al ingeniero del software crear nuevas aplicaciones a partir de las partes reutilizables.

Atributos

El software debe proveer la funcionalidad y desempeño requeridos por el usuario y debe ser mantenible, confiable, eficiente y aceptable.

- Mantenible, el software debe poder evolucionar para continuar cumpliendo con las especificaciones
- Confiable, el software no debe causar daños físicos o económicos en el caso de que falle
- Eficiente, el software no debe desperdiciar los recursos del sistema
- Aceptable, el software debe ser aceptado por los usuarios para los que fue diseñado.
- Debe ser entendible, utilizable y compatible con otros sistemas.

Importancia y aplicación del software

Importancia del Software

Cada software desarrolla funciones específicas dentro de una diversa gama de aplicaciones, y sin duda alguna uno de los programas que mayor utilidad representa dentro de una empresa, son los denominados Sistemas de Soporte a la Decisión (DSS).

"Las herramientas DSS se pueden definir como un sistema integrado de planificación y tratamiento de la información que incorpora la habilidad de interrogar al sistema de información de la empresa en una forma determinada, analizar la información obtenida y predecir, con base a unos modelos determinados, el impacto de las futuras decisiones antes de llevarlas a la práctica. Normalmente se trata de conjuntos integrados de programas (cálculos estadísticos, consultas de bases de datos, modelización, programación matemática, etc.) que pueden compartir los mismos datos e informaciones (tanto de fuentes internas como externas)".

De esta manera, la toma de decisiones se convierte en una variable crítica de éxito dentro de las empresas, y es aquí donde radica la importancia de un DSS.

"En la actualidad los sistemas de información juegan un papel fundamental para el éxito de las empresas. Es un gran reto el que una organización pueda tener controladas las variables de planeación, organización, control y dirección de la empresa. Para poder tener un control de las mismas, se requiere de tomar decisiones acertadas que puedan llevar a la organización en la dirección deseada."

Aplicaciones del software

El software puede aplicarse en cualquier situación en la que se haya definido previamente un conjunto específico de pasos procedimentales.

El contenido y el determinismo de la información son factores importantes a considerar para determinar la naturaleza de una aplicación de software. El contenido se refiere al significado y a la forma de la información de entrada y salida. El determinismo de la información se refiere a la predictibilidad del orden y del tiempo de llegada de los datos.

Las siguientes áreas del software indican la amplitud de las aplicaciones potenciales:

- Software de sistemas
- Software de tiempo real
- Software de gestión
- Software de ingeniería y científico
- Software empotrado
- Software de computadoras personales
- Software basado en Web
- Software de inteligencia emocional

Software de sistemas

El software de sistemas es un conjunto de programas que han sido escritos para servir a otros programas. Algunos programas de sistemas (por ejemplo: compiladores, editores y utilidades de gestión de archivos) procesan estructuras de información complejas pero determinadas.

Otras aplicaciones de sistemas (por ejemplo: ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones) procesan datos en gran medida indeterminados.

Se caracteriza por una fuerte interacción con el hardware de la computadora; una gran utilización por múltiples usuarios; una operación concurrente que requiere una planificación, una compartición de recursos y una sofisticada gestión de procesos; Unas estructuras de datos complejas y múltiples interfaces externas.

Software de tiempo real

El software que coordina, analiza, controla sucesos del mundo real conforme ocurren, se denomina de tiempo real. Entre los elementos del software de tiempo real se incluyen: un componente de adquisición de datos que recolecta y da formato a la información recibida del entorno externo, un componente de análisis que transforma la información según lo requiera la aplicación, un componente de control/salida que responda al entorno externo, y un componente de monitorización que coordina todos los demás componentes, de forma que pueda mantenerse la respuesta en tiempo real (típicamente en el rango de un milisegundo a un segundo).

Software de gestión

El proceso de la información comercial constituye la mayor de las áreas de aplicación del software. Los sistemas discretos (por ejemplo: nóminas, cuentas de haberes-débitos, inventarios, etc.) han evolucionado hacia el software de sistemas de información de gestión (SIG) que accede a una o más bases de datos que contienen información comercial.

Las aplicaciones en esta área re estructuran los datos existentes para facilitar las operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamientos de datos, las aplicaciones de software de gestión también realizan cálculo interactivo (por ejemplo: el procesamiento de transacciones en puntos de ventas).

Software de ingeniería y científico

El software de ingeniería y científico está caracterizado por los algoritmos de manejo de números. Las aplicaciones van desde la astronomía a la vulcanología, desde el análisis de la presión de los automotores a la dinámica orbital de las lanzaderas espaciales y desde la biología molecular a la fabricación automática. Sin embargo, las nuevas aplicaciones del área de ingeniería/ciencia se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora (del inglés CAD), la simulación de sistemas y otras aplicaciones interactivas, han comenzado a coger características del software de tiempo real e incluso del software de sistemas.

Software empotrado

Los productos inteligentes se han convertido en algo común en casi todos los mercados de consumo e industriales. El software empotrado reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo. El software empotrado puede ejecutar funciones muy limitadas y curiosas (por ejemplo: el control de las teclas de un horno de microondas) o suministrar una función significativa y con capacidad de control (por ejemplo: funciones digitales en un automóvil, tales como control de la gasolina, indicadores en el salpicadero, sistemas de frenado, etc.).

Software de computadoras personales

El mercado del software de computadoras personales ha germinado en las pasadas dos décadas. El procesamiento de textos, las hojas de cálculo, los gráficos por computadora, multimedia, entretenimientos, gestión de bases de datos, aplicaciones financieras, de negocios y personales y redes o acceso a bases de datos externas son algunas de los cientos de aplicaciones.

Software basado en Web

Las páginas Web buscadas por un explorador son software que incorpora instrucciones ejecutables (por ejemplo, CGI, HTML, Perl, o Java), y datos (por ejemplo, hipertexto y una variedad de formatos de audio y visuales). En esencia, la red viene a ser una gran computadora que proporciona un recurso software casi ilimitado que puede ser accedido por cualquiera con un modem.

Software de inteligencia artificial

El software de inteligencia artificial (IA) hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo. Los sistemas expertos, también llamados sistemas basados en el conocimiento, reconocimiento de patrones (imágenes y voz), redes neuronales artificiales, prueba de teoremas, y los juegos son representativos de las aplicaciones de esta categoría.

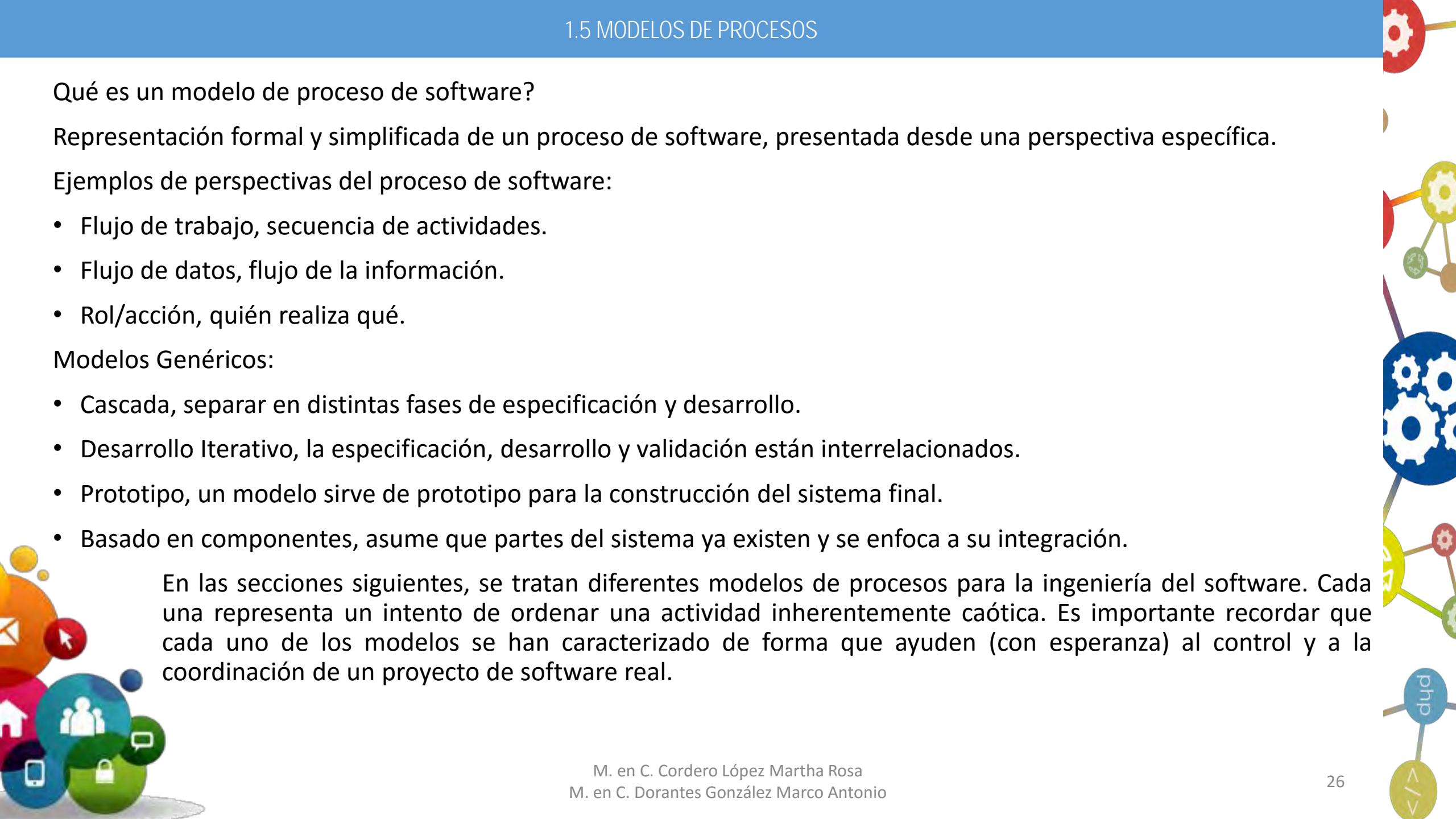
Ciclo de vida del software

Ciclo de vida del Software

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

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

Modelos de procesos



Qué es un modelo de proceso de software?

Representación formal y simplificada de un proceso de software, presentada desde una perspectiva específica.

Ejemplos de perspectivas del proceso de software:

- Flujo de trabajo, secuencia de actividades.
- Flujo de datos, flujo de la información.
- Rol/acción, quién realiza qué.

Modelos Genéricos:

- Cascada, separar en distintas fases de especificación y desarrollo.
- Desarrollo Iterativo, la especificación, desarrollo y validación están interrelacionados.
- Prototipo, un modelo sirve de prototipo para la construcción del sistema final.
- Basado en componentes, asume que partes del sistema ya existen y se enfoca a su integración.

En las secciones siguientes, se tratan diferentes modelos de procesos para la ingeniería del software. Cada una representa un intento de ordenar una actividad inherentemente caótica. Es importante recordar que cada uno de los modelos se han caracterizado de forma que ayuden (con esperanza) al control y a la coordinación de un proyecto de software real.

Lineal secuencial

- El *modelo lineal secuencial* sugiere un enfoque sistemático, secuencial, para el desarrollo del software
- Comprende el análisis, diseño, codificación, pruebas y mantenimiento.
- Propuesto por Royce en 1970
- Dirigido a establecer orden en el desarrollo de grandes productos

Modelado: según el ciclo de ingeniería convencional, el modelo lineal secuencial comprende las siguientes actividades:

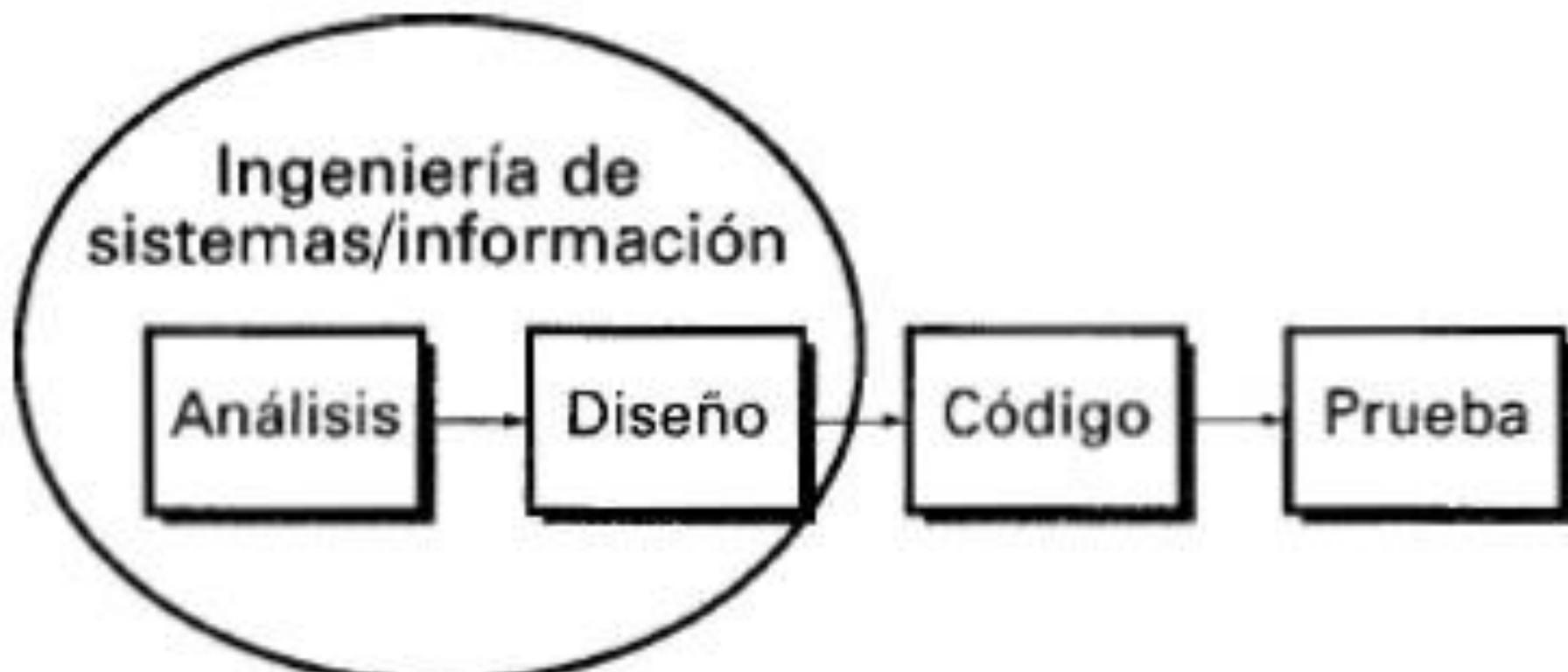
- **Ingeniería y modelado de Sistemas/Información.**

Como el software siempre forma parte de un sistema más grande (o empresa), el trabajo comienza estableciendo requisitos de todos los elementos del sistema y asignando al software algún subgrupo de estos requisitos.

- **Aplicación**

Esta visión del sistema es esencial cuando el software se debe interconectar con otros elementos como hardware, personas y bases de datos.

EL modelo lineal secuencial



- **Ingeniería de sistemas-información**

La ingeniería de información abarca los requisitos que se recogen en el nivel de empresa estratégico y en el nivel del área de negocio.

La ingeniería y el análisis de sistemas comprende los requisitos que se recogen en el nivel del sistema con una pequeña parte de análisis y de diseño

- **Análisis de los requisitos de software**

El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza del (los) programa(s) a construirse, el ingeniero («analista») del software debe comprender el dominio de información del software , así como la función requerida, comportamiento, rendimiento e interconexión.

- **Diseño**
- Es un proceso de muchos pasos que se centra en cuatro atributos distintos de programa:
- estructura de datos,
- arquitectura de software,
- representaciones de interfaz
- y detalle procedural (algoritmo).
- El proceso del diseño traduce requisitos en una representación del software donde se pueda evaluar su calidad antes de que comience la codificación.
- **Generación de código**

El diseño se debe traducir en una forma legible por la máquina. El paso de generación de código lleva a cabo esta tarea. Si se lleva a cabo el diseño de una forma detallada, la generación de código se realiza mecánicamente.

- **Pruebas**

El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales; es decir, realizar las pruebas para la detección de errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos.

- **Mantenimiento**

El software sufrirá cambios después de ser entregado al cliente. Se producirán cambios porque se han encontrado errores, porque el software debe adaptarse para acoplarse a los cambios de su entorno externo (por ejemplo: se requiere un cambio debido a un sistema operativo o dispositivo periférico nuevo), o porque el cliente requiere mejoras funcionales o de rendimiento. El soporte y mantenimiento del software vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo.



VENTAJAS

- Proporciona una plantilla en la que se encuentran métodos para análisis, diseño, codificación, pruebas y mantenimiento.
- Puede acoplar interacción, lo hace indirectamente.
- Sigue siendo el modelo de procesos mas extensamente utilizado por la ingeniería de software.
- Hace detección de errores



LINEAL SECUENCIAL



DESVENTAJAS

- Los proyectos reales raras veces siguen el modelo secuencial que propone el modelo.
- El modelo lineal secuencial requiere que el cliente exponga explícitamente todos los requisitos.
- Un grave error puede ser desastroso si no se detecta hasta que se revisa el programa.
- Alto costo.



¿Por qué falla algunas veces el modelo lineal?

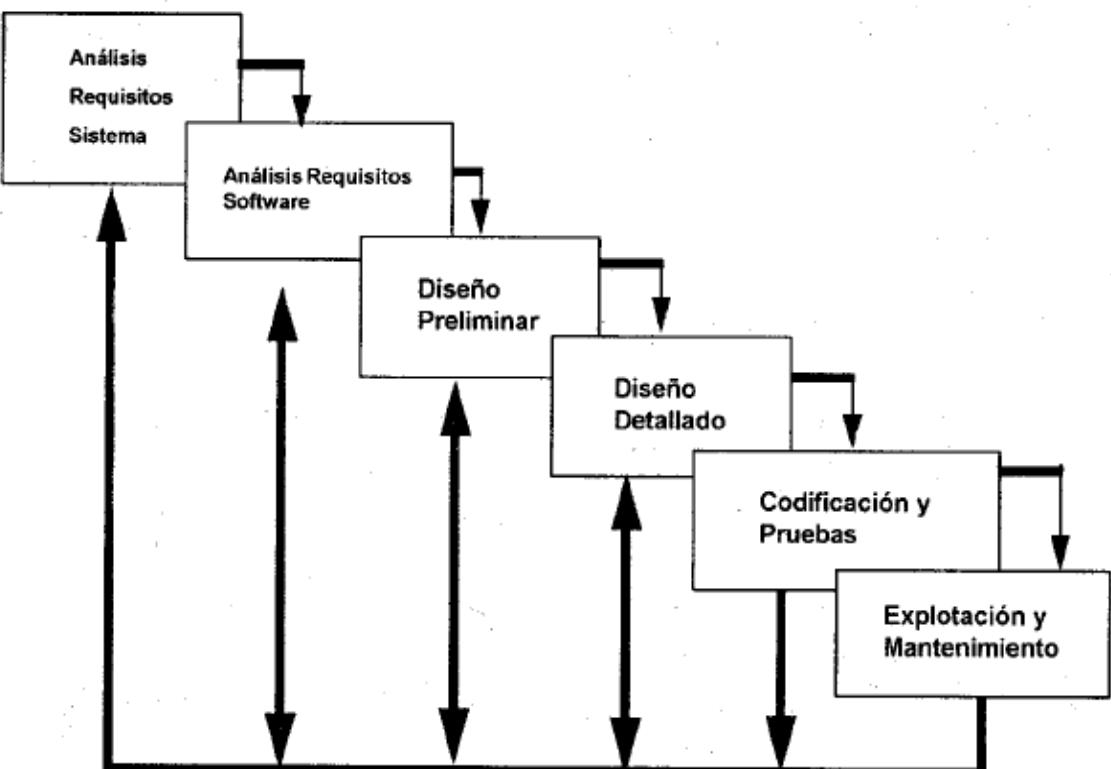
- 1.- Los proyectos reales raras veces siguen el modelo secuencial que propone el modelo. Aunque el modelo lineal puede acoplar interacción, lo hace indirectamente. Como resultado, los cambios pueden causar confusión cuando el equipo del proyecto comienza.
- 2.- A menudo es difícil que el cliente exponga explícitamente todos los requisitos. El modelo lineal secuencial lo requiere y tiene dificultades a la hora de acomodar la incertidumbre natural al comienzo de muchos proyectos.
- 3.- El cliente debe tener paciencia. Una versión de trabajo del (los) programa(s) no estará disponible hasta que el proyecto esté muy avanzado. Un grave error puede ser desastroso si no se detecta hasta que se revisa el programa.

CASCADA

Tiene las mismas características que el modelo lineal como se había mencionado anteriormente pero en este se tiene la capacidad de regresar si se detecta un error en cualquiera de las etapas de planeación, desarrollo, diseño, pruebas y mantenimiento del software; Lo cual se observa en el siguiente diagrama:

- Críticas:
- No refleja realmente el proceso de desarrollo del software
- Se tarda mucho tiempo en pasar por todo el ciclo
- Perpetúa el fracaso de la industria del software en su comunicación con el usuario final
- El mantenimiento se realiza en el código fuente
 - Las revisiones de proyectos de gran complejidad son muy difíciles

Impone una estructura de gestión de proyectos





INCREMENTAL

El modelo incremental combina elementos del modelo lineal secuencial con la filosofía interactiva de construcción de prototipos. Como muestra en la figura, el modelo incremental aplica secuencias lineales de forma escalonada mientras progresá el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Cuando se utiliza un modelo incremental, el primer incremento a menudo es un producto esencial. Es decir, se afrontan requisitos básicos, pero muchas funciones suplementarias (algunas conocidas, otras no) quedan sin extraer. El cliente utiliza el producto central (o sufre la revisión detallada). Como un resultado de utilización y/o de evaluación, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales.

Paradigma

El software se ve como una integración de resultados sucesivos obtenidos después de cada interacción



Autor

Meir Manny Lehman fue un científico e investigador en el área de la computación; miembro de la Real Academia de Ingeniería.

Sus contribuciones en investigación incluyen el estudio del fenómeno de la evolución del software y la propuesta de sus conocidas leyes, que llevan su nombre, las Leyes de Lehman de la evolución del software. [1]

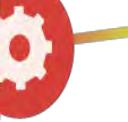
Definido por Lehman en 1984.

Surge porque en los primeros desarrollos se podía esperar largo tiempo hasta que el software estuviese listo. Las reglas del negocio de hoy no lo permiten. [2]

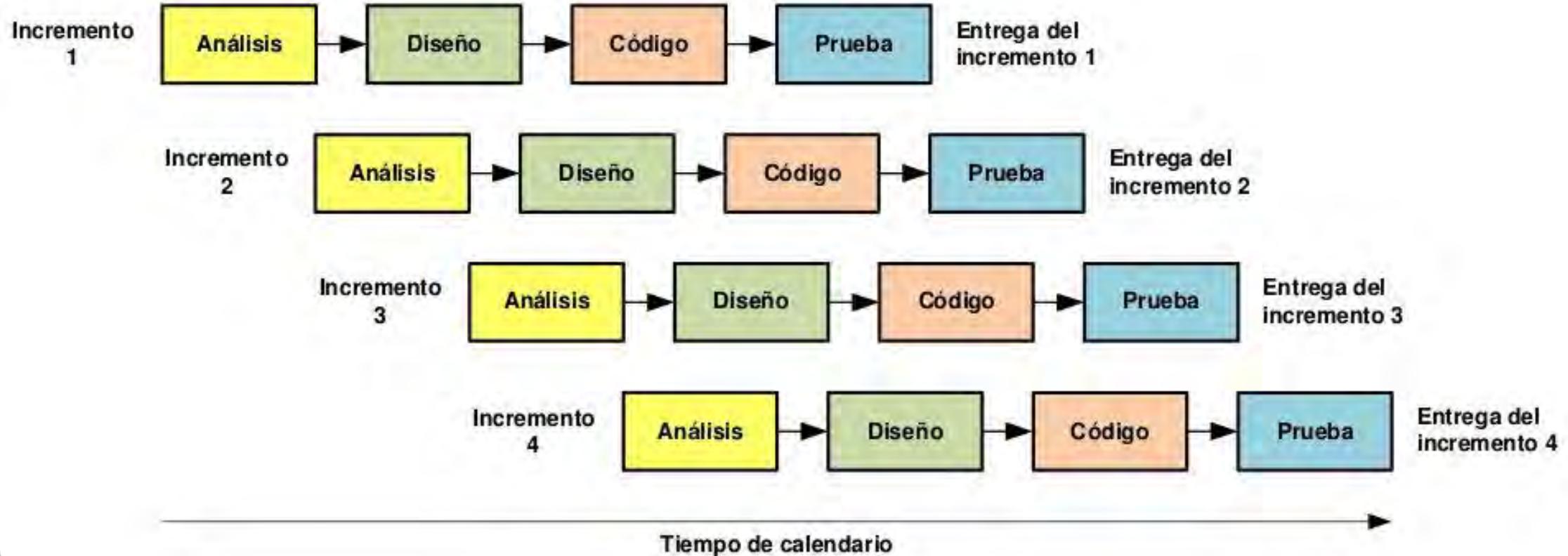


Características

- Constituye una de las variantes del modelo en cascada puro.
- El primer incremento a menudo es un producto esencial (núcleo).
- Se va creando el Software añadiendo componentes funcionales al sistema (incrementos).
- El cliente utiliza el producto central (sufre una revisión detallada) y como resultado de utilización, de evaluación, se desarrolla un plan para el incremento siguiente.

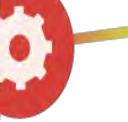


INCREMENTAL



<http://osc.co.cr/wp-content/uploads/2011/06/incremental.jpeg>





Ventajas

- Con un paradigma incremental se reduce el tiempo de desarrollo inicial, ya que se implementa la funcionalidad parcial.
- También provee un impacto ventajoso frente al cliente, que es la entrega temprana de partes operativas del Software.
- El modelo proporciona todas las ventajas del modelo en cascada realimentado, reduciendo sus desventajas sólo al ámbito de cada incremento.
- Permite entregar al cliente un producto más rápido en comparación del modelo de cascada.
- Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos.
 - Por su versatilidad requiere de una planeación cuidadosa tanto a nivel administrativo como técnico.
 - El resultado puede ser muy positivo.

Desventajas

- El modelo Incremental no es recomendable para casos de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos.
- Requiere de mucha planeación, tanto administrativa como técnica.
- Requiere de metas claras para conocer el estado del proyecto.
- Los errores en los requisitos se detectan tarde.

INCREMENTAL



Problemas que resuelve

- Corrige la necesidad de una secuencia no lineal de pasos de desarrollo.
- Éste modelo se adecúa a entornos de alta incertidumbre
- Es útil cuando la dotación del personal no está disponible para una implementación completa en la fecha límite que se haya establecido para el proyecto
- No es recomendable para sistemas de tiempo real, de alto nivel de seguridad, procesamiento distribuido y alto índice de riesgos.



DESARROLLO RÁPIDO DE APLICACIONES (DRA)



El Desarrollo Rápido de Aplicaciones (DRA) es un modelo de proceso del desarrollo del software lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto.

El modelo DRA es una adaptación a alta velocidad del modelo lineal secuencial en el que se logra el desarrollo rápido utilizando una construcción basada en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un sistema completamente funcional dentro de períodos cortos de tiempo (por ejemplo: de 60 a 90 días). Cuando se utiliza principalmente para aplicaciones de sistemas de información, el enfoque DRA comprende las siguientes fases:

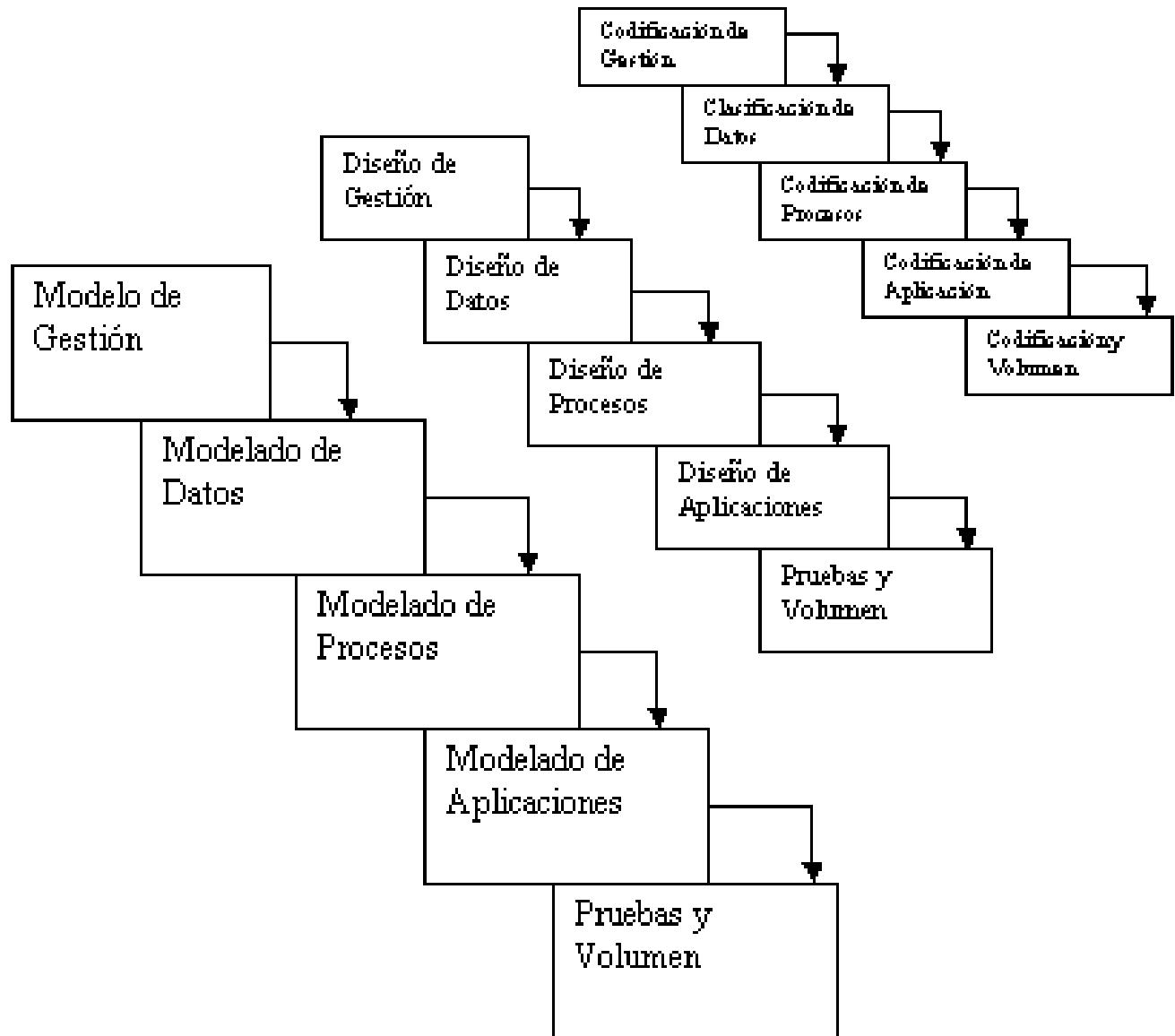
Modelado de Gestión: El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión?, ¿Qué información se genera?, ¿Quién la genera?, ¿A dónde va la información?, ¿Quién la procesa?.

Modelado de Datos: El flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.

Modelado de Proceso: Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir o recuperar un objeto de datos.

Generación de Aplicaciones: El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.

Pruebas y Entrega: Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.



Técnicas de cuarta generación

Las técnicas de cuarta generación son un conjunto muy diverso de métodos y herramientas que tienen por objetivo el de facilitar el desarrollo del software, facilitan al que desarrolla el software la propiedad de especificar algunas características del mismo a alto nivel, mas tarde, la herramienta genera automáticamente el código fuente a partir de esta especificación.

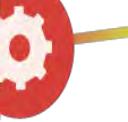


Al igual que otros paradigmas, T4G comienza con el paso de reunión de requisitos. Idealmente, el cliente describe los requisitos, que son, a continuación, traducidos directamente a un prototipo operativo. Estado actual de los enfoques de T4G:

1. El uso de T4G ha crecido considerablemente en la última década y ahora es un enfoque viable para muchas de las diferentes áreas de aplicación. Junto con las herramientas de ingeniería de software asistida por computadora (CASE) y los generadores de código, T4G ofrece una solución fiable a muchos problemas de software.
2. Los datos recogidos en compañías que usan T4G parecen indicar que el tiempo requerido para producir software se reduce mucho para aplicaciones pequeñas y de tamaño medio, y que la cantidad de análisis y diseño para las aplicaciones pequeñas, también se reduce.
3. Sin embargo, el uso de T4G para grandes trabajos de desarrollo de software exige el mismo o más tiempo de análisis, diseño y prueba (actividades de ingeniería del software), perdiéndose así un tiempo sustancial que se ahorra mediante la eliminación de la codificación.

Características mas importantes de T4G.-

1. Acceso a base de datos: utilizando lenguajes de consulta de alto nivel.
2. Generadores de códigos: a partir de una especificación de los requisitos se genera automáticamente toda la aplicación
3. Generación de pantallas: permitiendo diseñar la pantalla dibujándola directamente, incluyendo además el control del cursor y la gestión de los errores de los datos de entrada.
4. Gestión de entornos gráficos. Manejo de un entorno agradable al usuario.
5. Generación de informes: Como otros paradigmas, T4G comienza con el paso de recolección de requerimientos. En el mejor de los casos el cliente debería describir los requerimientos y éstos traducirse directamente a un prototipo operacional pero en general esto no es así.



Ventajas

- Reducción en el tiempo de desarrollo en el software.
- Mejora significativa en la productividad de la gente que construye el software.
- Recolección y análisis de grandes colecciones de datos.



Desventajas

- Los lenguajes de programación, ya que el código fuente producido con la herramientas es ineficiente
- El mantenimiento de grandes sistemas de software desarrollando usan T4G está abierta a discusión.
- Limitada a las aplicaciones y análisis de sistemas de información comerciales, análisis de información y de la obtención de grandes bases de datos.

INCREMENTAL



Prototipos

- Comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un «diseño rápido».
- El diseño rápido se centra en una representación de esos aspectos del software que serán visibles para el usuario/cliente (por ejemplo: enfoques de entrada y formatos de salida). El diseño rápido lleva a la construcción de un prototipo.
- El prototipo lo evalúa el cliente/usuario y se utiliza para refinar los requisitos del software a desarrollar. La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo al mismo tiempo que el desarrollador comprenda mejor lo que se necesita hacer.

Características

- Se espera que sirva como un mecanismo para identificar los requisitos del software.
- El desarrollador intenta hacer uso de los fragmentos del programa ya existentes o aplica herramientas (por ejemplo: generadores de informes, gestores de ventanas, etc.) que permiten generar rápidamente programas de trabajo.
- Plan rápido.
- Modelado, diseño rápido.
- Construcción del Prototipo.
- Desarrollo, entrega y retroalimentación.
 - Comunicación.
 - Entrega del desarrollo final.

Propuesta de Brooks

- En la mayoría de los proyectos, el primer sistema construido apenas se puede utilizar. Puede ser demasiado lento, demasiado grande o torpe en su uso, o las tres a la vez. No hay otra alternativa que comenzar de nuevo, aunque nos duela pero es más inteligente, y construir una versión rediseñada en la que se resuelvan estos problemas.
- Cuando se utiliza un concepto nuevo de sistema o una tecnología nueva, se tiene que construir un sistema que no sirva y se tenga que tirar, porque incluso la mejor planificación no es omnisciente como para que esté perfecta la primera vez.
- Por lo tanto la pregunta de la gestión no es si construir un sistema piloto y tirarlo. Tendremos que hacerlo. La Única pregunta es si planificar de antemano construir un desechable, o prometer entregárselo a los clientes.

Lo ideal sería que el prototipo sirviera como un mecanismo para identificar los requisitos del software. Si se construye un prototipo de trabajo, el desarrollador intenta hacer uso de los fragmentos del programa ya existentes o aplica herramientas (por ejemplo: generadores de informes, gestores de ventanas, etc.) que permiten generar rápidamente programas de trabajo.

El prototipo puede servir como primer sistema, es verdad que a los clientes y a los que desarrollan les gusta el paradigma de construcción de prototipos. A los usuarios les gusta el sistema real y a los que desarrollan les gusta construir algo inmediatamente. Sin embargo, la construcción de prototipos también puede ser problemática por las siguientes razones:

1. El cliente ve lo que parece ser una versión de trabajo del software, sin saber que con la prisa de hacer que funcione no se ha tenido en cuenta la calidad del software global o la facilidad de mantenimiento a largo plazo. Cuando se informa de que el producto se debe construir otra vez para que se puedan mantener los niveles altos de calidad, el cliente no lo entiende y pide que se apliquen pequeños ajustes para que se pueda hacer del prototipo un producto final.
2. El desarrollador, a menudo, hace compromisos de implementación para hacer que el prototipo funcione rápidamente. Se puede utilizar un sistema operativo o lenguaje de programación inadecuado simplemente porque está disponible y porque es conocido; un algoritmo eficiente se puede implementar simplemente para demostrar la capacidad.

Tipos de prototipos.

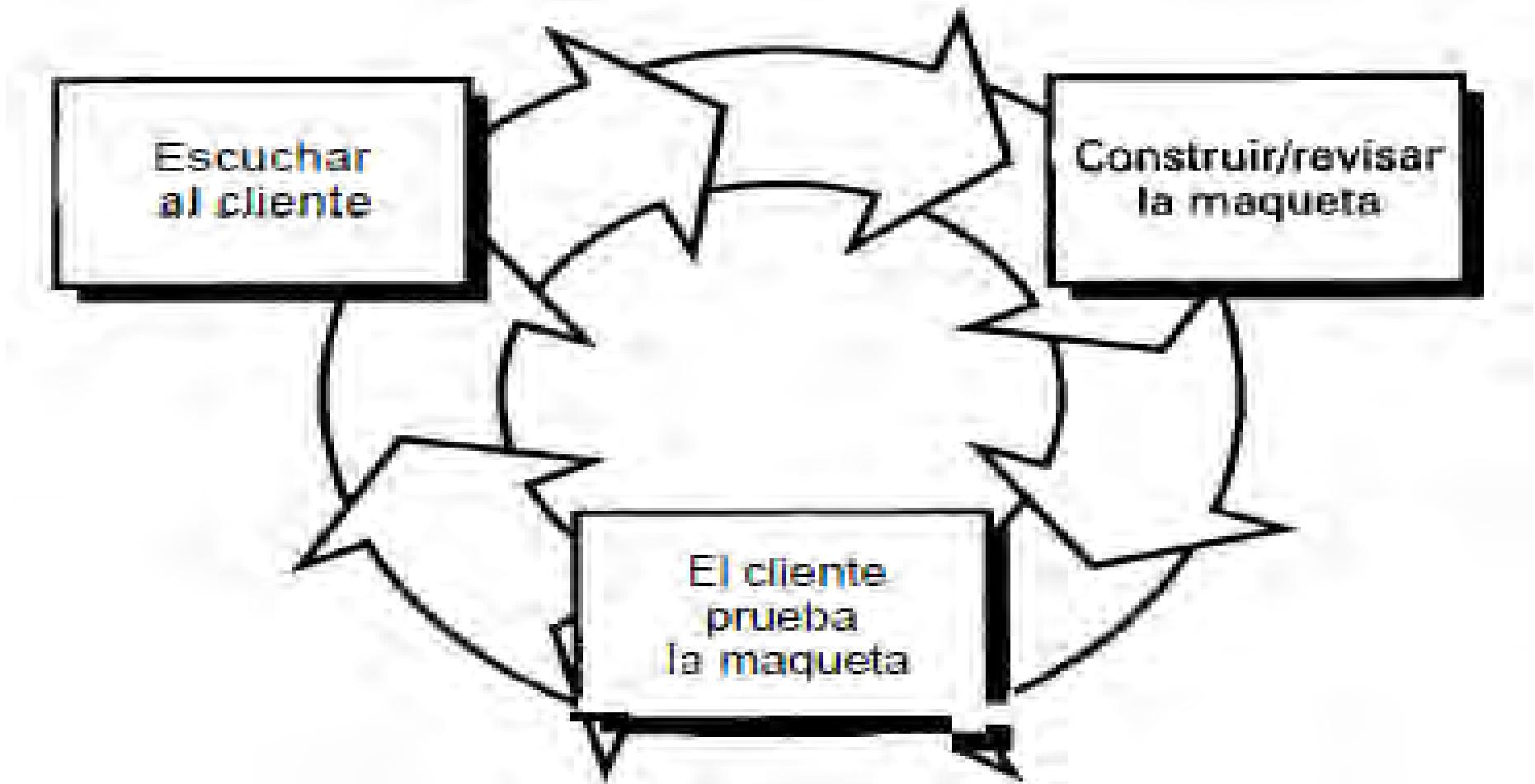
- Prototipo Parchado

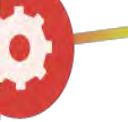
Un prototipo en sistemas de información es un modelo operable que tiene todas las características necesarias, pero que es ineficiente debido a que los programas fueron hechos a la carrera con el objetivo de ser funcionales en vez de ser eficientes. Un prototipo parchado es un sistema de información que tiene todas las características propuestas pero es realmente un modelo básico que eventualmente será mejorado.

- Prototipo No Operacional

Un modelo a escala no funcional de un sistema de información puede ser hecho cuando la codificación requerida por las aplicaciones es muy amplia para hacer el prototipo. Este modelo a escala no funcional es utilizado con el objeto de aspecto de diseño y no funcionalidad.

- Prototipo Primero De Una Serie





Ventajas

- El prototipo puede servir como «primer sistema».
- El cliente se relaciona directamente con el desarrollo del prototipo del proyecto.
- Cuando el cliente tiene una necesidad legítima, pero está desorientado sobre los detalles, el primer paso es desarrollar un prototipo.
- Este modelo es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
- También ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina.



Desventajas

- El cliente ve lo que parece ser una versión de trabajo del software, sin tener conocimiento de que el prototipo.
- Cuando se informa de que el producto se debe construir otra vez para que se puedan mantener los niveles altos de calidad, el cliente no lo entiende y pide que se apliquen algunos pequeños ajustes.
- De forma demasiado frecuente la gestión de desarrollo del software es muy lenta.
- Se puede utilizar un sistema operativo o lenguaje de programación inadecuado simplemente porque está disponible y porque es conocido.
- Después de algún tiempo, el desarrollador debe familiarizarse con estas selecciones, y olvidarse de las razones por las que son inadecuadas. La selección menos ideal ahora es una parte integral del sistema.

INCREMENTAL



Modelo en espiral

- Creado por por Barry Boehm en 1976
- La meta del modelo espiral del proceso de producción del software es proporcionar un marco para diseñar tales procesos
- Esta dirigido por los niveles de riesgo en el proyecto actual.
- En comparación con los actuales modelos, el modelo espiral se puede ver como metamodelo. (acomoda cualquier modelo de proceso del desarrollo)
- Usándolo como referencia, se puede elegir el modelo de desarrollo más apropiado.

Comunicación con el cliente: Las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.

Planificación: Las tareas requeridas para definir recursos, el tiempo y otra información relacionadas con el proyecto.

Ánalysis de riesgos: Las tareas requeridas para evaluar riesgos técnicos y de gestión.

Ingeniería: Las tareas requeridas para construir una o más representaciones de la aplicación.

Construcción y adaptación: Las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario (por ejemplo: documentación y práctica)

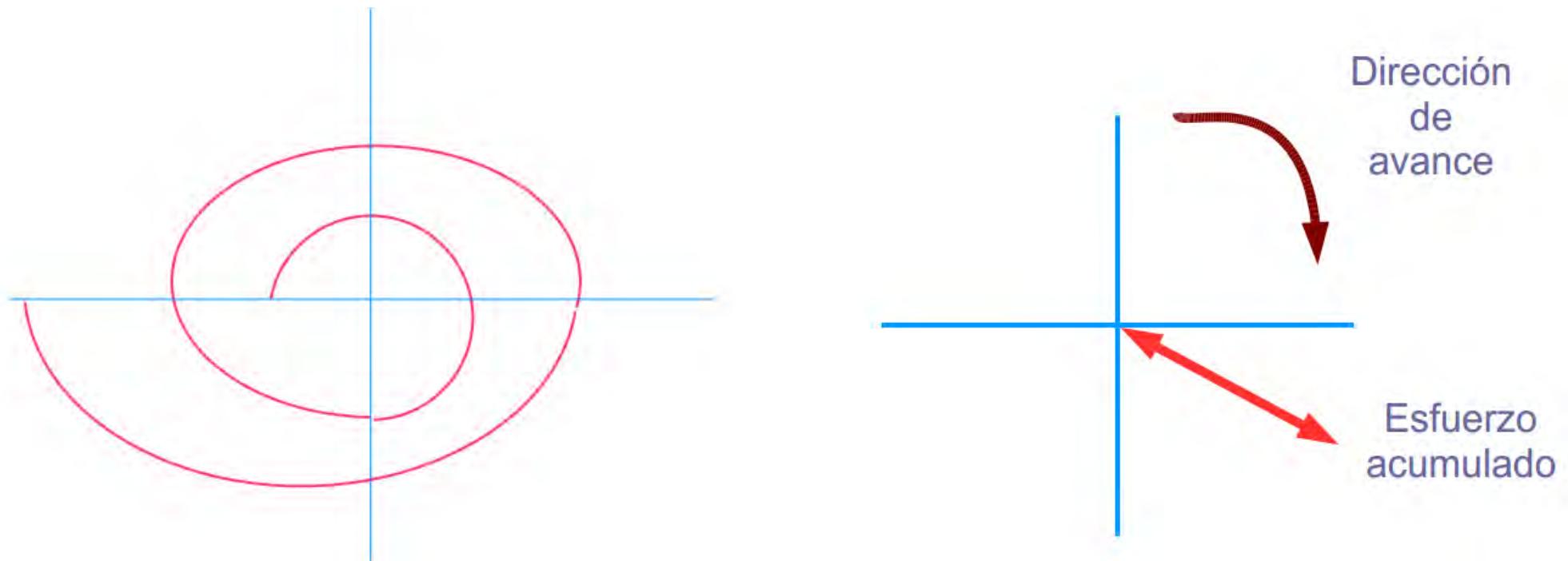
Evaluación del cliente: Las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación.

Las regiones están compuestas por un conjunto de tareas del trabajo, llamado conjunto de tareas, que se adaptan a las características del proyecto que va a emprenderse. Para proyectos pequeños, el número de tareas de trabajo y su formalidad es bajo. Para proyectos mayores y más críticos cada región de tareas contiene tareas de trabajo que se definen para lograr un nivel más alto de formalidad. En todos los casos, se aplican las actividades de protección (por ejemplo: gestión de configuración del software y garantía de calidad del software).

Esquema



Espiral (Una representación)





ESPIRAL

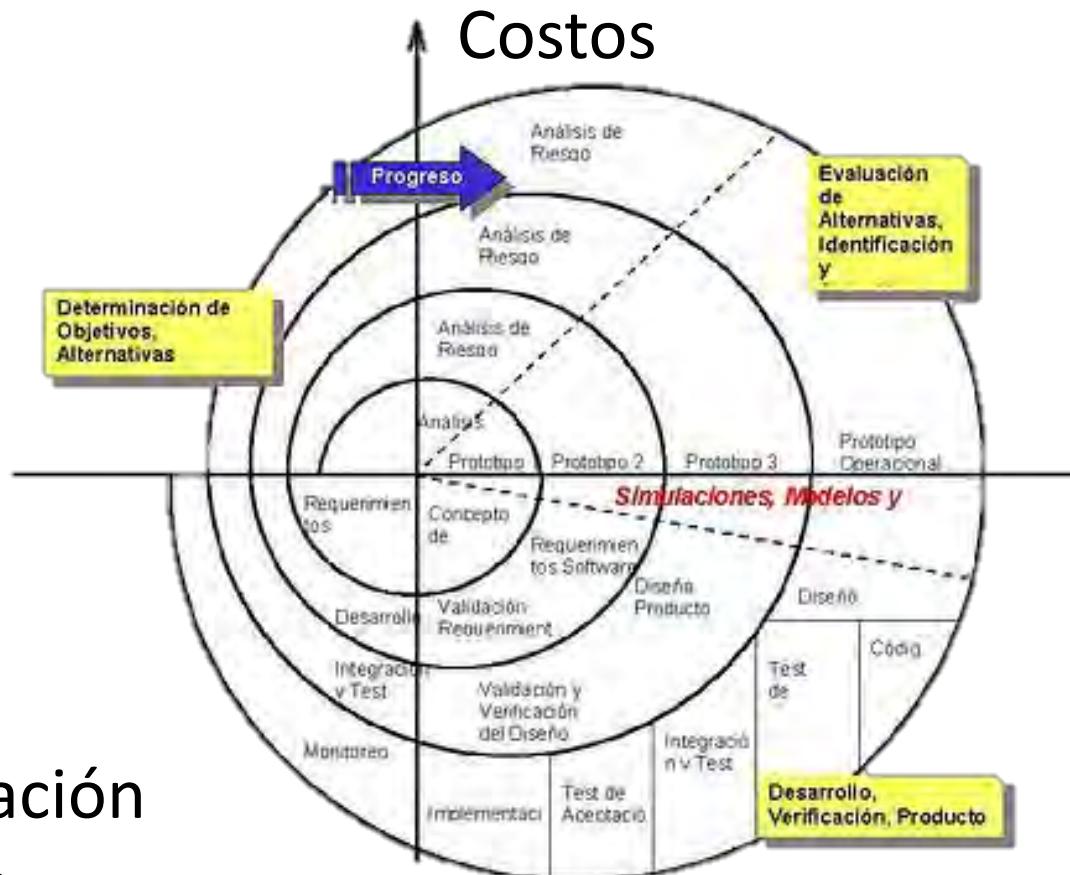
Estrategia para resolución de problemas

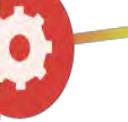


Esquema

Revisión

Planificación
Próxima





INCREMENTAL



Ventajas

- Mientras los costos del modelo aumentan, los riesgos del proyecto disminuyen.
- Buen control sobre el desarrollo del proyecto.
- Evaluación en cada fase que permite cambios de objetivos.
- Funciona bien en proyectos de innovación

Desventajas

- Es un modelo complicado.
- Exige desarrolladores con mucha experiencia para manejar la complejidad de los problemas.
- La evaluación de riesgos es compleja.
- Excesiva flexibilidad para algunos proyectos .



Aplicaciones

- Para proyectos más pequeños, el concepto de desarrollo ágil de software se está convirtiendo en una alternativa viable.
- Proyectos que involucren fuentes externas del software
- Proyectos donde los riesgos sean menores
- Proyectos en los que se requiera encontrar soluciones viables, o en proyectos en los que este previsto abandonar el proyecto a tiempo.
- También es razonable utilizar el modelo en espiral en proyectos en los objetivos de negocio son inestables pero la arquitectura

PARADIGMAS EXTRAS

El concepto de métodos formales involucra una serie de técnicas lógicas y matemáticas con las que es posible especificar, diseñar, implementar y verificar los sistemas de información (Monin, 2003).

Los métodos formales surgieron como puntos de vista analíticos con los que es posible verificar el desarrollo de sistemas mediante la lógica y las matemáticas, lo que aporta grandes ventajas para mejorar la calidad de los programas y por tanto la Ingeniería de Software.

En este campo del conocimiento, la especificación formal es una de las más importantes fases del ciclo de vida, labor que requiere mucho cuidado ya que su función es garantizar que tanto el funcionamiento como el desempeño del programa sean correctos, bajo cualquier situación. En el futuro, los métodos formales

deberían estar presentes como principios esenciales en el desarrollo de software, ya que se convierten en la base para aplicar las técnicas de prueba y, dado su principio matemático, en potencialmente automatizables.

Su importancia reside en que a medida que los sistemas informáticos crecen en complejidad las pérdidas causadas por fallas son cada vez mayores, por esta razón las técnicas formales agregan algo conocido como “corrección certificada” lo cual evita fallas y se traduce en menos perdidas de dinero, los métodos formales atraen a la industria, ya que su aplicación ayuda a lograr los estándares de calidad que la sociedad exige.

La importancia de los métodos formales en la Ingeniería de Software se incrementó en los últimos años: se desarrollan nuevos lenguajes y herramientas para especificar y modelar formalmente, y se diseñan metodologías maduras para verificar y validar. Los modelos que se diseñan y construyen de esta forma, desde las fases iniciales del desarrollo de software, son esenciales para el éxito del futuro proyecto; ya que en la actual Ingeniería de Software constituyen la base que sustenta las subsiguientes fases del ciclo de vida, y porque los errores surgidos en ella tienen gran impacto en los costos del proyecto (Perry, 2006).

Los formalismos matemáticos de los métodos formales son:

- Lógica de primer orden y teoría de conjuntos (Manna & Waldinger, 1985). Se utilizan para especificar el sistema mediante estados y operaciones, de tal forma que los datos y sus relaciones se describan detalladamente, sus propiedades se expresen en lógica de primer orden, y la semántica del lenguaje tenga como base la teoría de conjuntos.
- Algebraicos y de especificación ecuacional (Ehrig & Mahr, 1990). Describen las estructuras de datos de forma abstracta para establecer el nombre de los conjuntos de datos, sus funciones básicas y propiedades mediante fórmulas ecuacionales, en las que no existe el concepto de estado modifiable en el tiempo.
- Redes de Petri (Murata, 1989), (Reising, 1991), (Manna & Pnueli, 1992), (Bause & Kritzinger, 2002), (Desel & Esparza, 2005), (Juhás et al., 2007). Establecen el concepto de estado del sistema mediante lugares que pueden contener marcas, y hacen uso de un conjunto de transiciones con pre y post condiciones, para describir cómo evoluciona el sistema, y cómo produce marcas en los puntos de la red.

Mitos sobre los métodos formales:

1. Garantizan la perfección del software y hacen innecesaria su verificación.
2. Solo sirven para demostrar que los programas son correctos.
3. Sólo es necesario aplicarlos a sistemas en donde la seguridad es crítica.
4. Sólo pueden ser aplicados por expertos en matemáticas.
5. No se aplican en sistemas reales de gran tamaño.
6. Retrasan el desarrollo del software.

Paradigma de desarrollo basado en componentes

Es un proceso que concede particular importancia al diseño y a la construcción de sistemas basados en computadoras que utilizan “componentes” de software reutilizables.

- El desarrollo de software basado en componentes permite reutilizar piezas de código pre-elaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión.
- Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio.

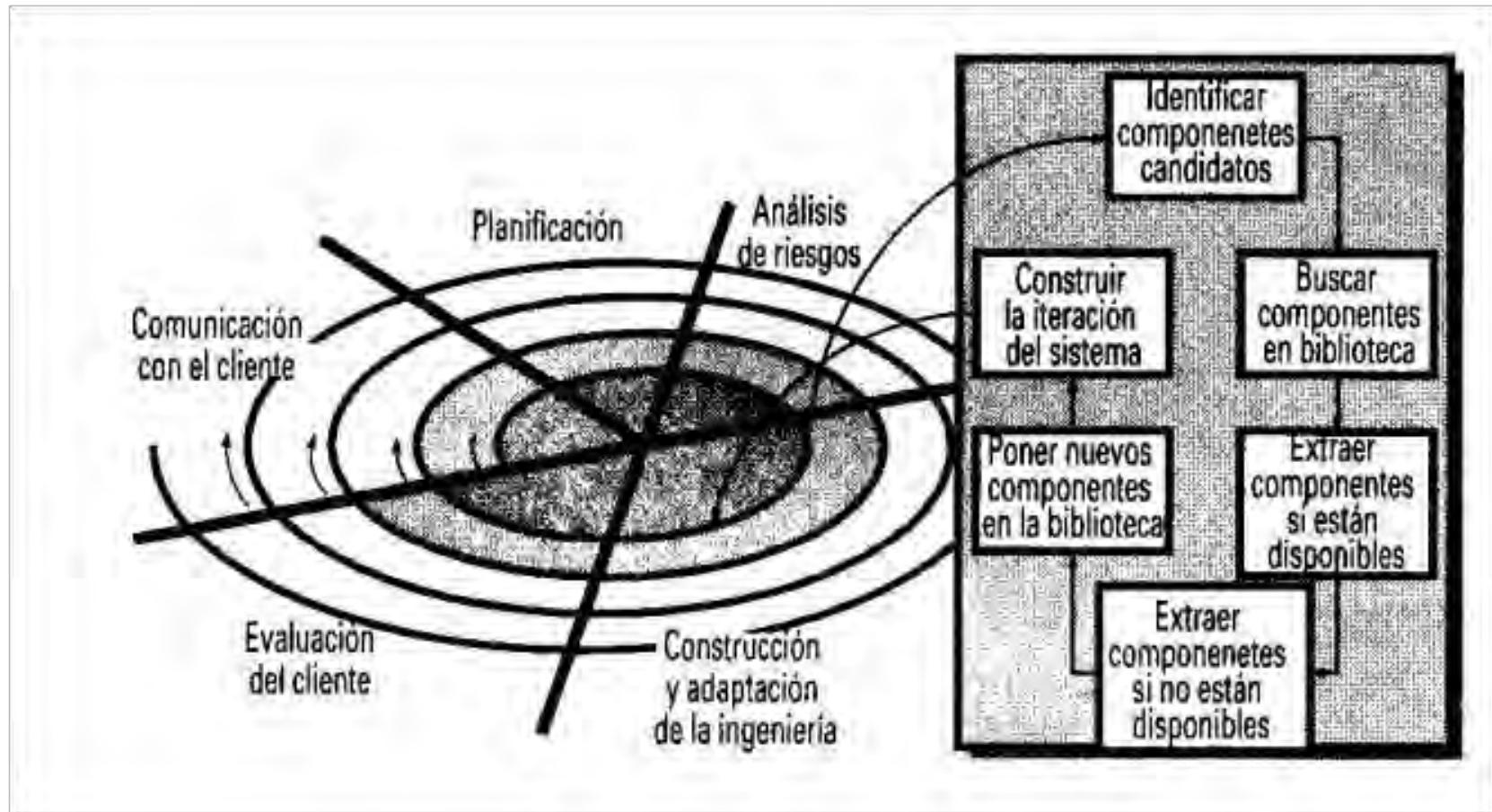
Características

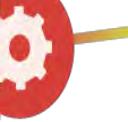
- Identifiable
- Auto contenido
- Puede ser remplazado por otro componente
- Con acceso solamente a través de su interfaz
- Sus servicios no varían
- Bien Documentado
- Es genérico
- Reutilizado dinámicamente
 - Independiente de la plataforma

- Proporcionan el marco de trabajo técnico para un modelo de proceso basado en componentes.
- Enfatiza la creación de clases que encapsulan tanto los datos como los algoritmos que se utilizan para manejar los datos.
- El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo en espiral.
- Las clases creadas en los proyectos de ingeniería del software anteriores, se almacenan en una biblioteca de clases o diccionario de datos.
- Según estudios de reutilización, QSM Associates, Inc. Informa que el ensamblaje de componentes lleva a una reducción del 70% de tiempo de ciclo de desarrollo, un 84% del coste del proyecto y un índice de productividad del 26.2, comparado con la norma de industria del 16.9. Aunque estos resultados están en función de la robustez de la biblioteca de componentes.



Esquema





INCREMENTAL



Ventajas

- Reutilización del software
- Simplifica las pruebas
- Simplifica el mantenimiento del sistema
- Mayor calidad
- Ciclos de desarrollo más cortos
- Mejor ROI
- Funcionalidad mejorada



Aplicaciones

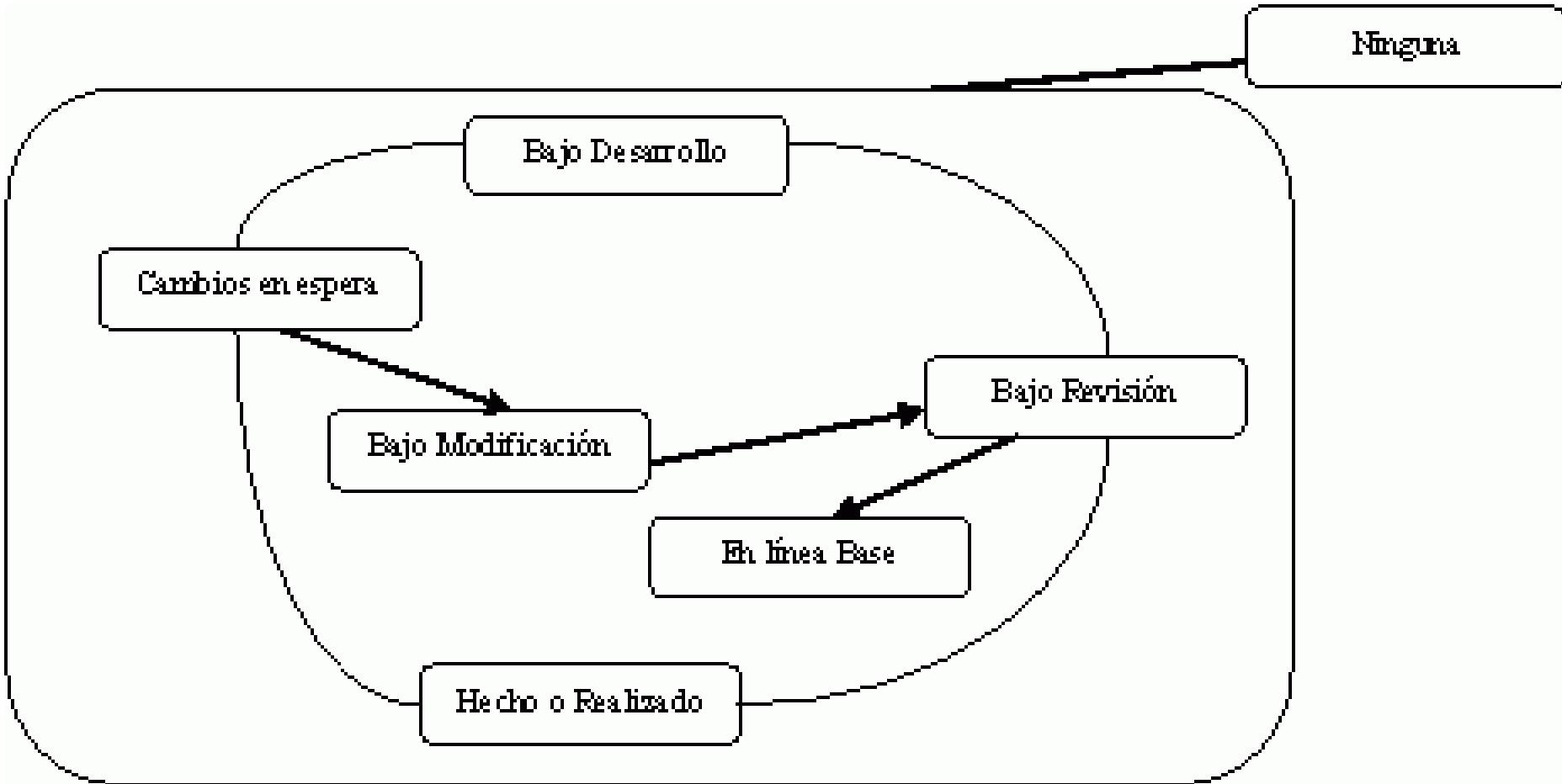
Este paradigma es poco recomendable para grupos de trabajo donde no se cuenta con un equipo experto en análisis de riesgos, o para proyectos donde los riesgos sean menores y no se justifique la flexibilidad y gestión de riesgos que ofrece este paradigma.

Paradigma de desarrollo concurrente

Davis y Sitaram han descrito el modelo de desarrollo concurrente, llamado algunas veces ingeniería concurrente, de la forma siguiente: Los gestores de proyectos que siguen los pasos del estado del proyecto en lo que se refiere a las fases importantes [del ciclo de vida clásico] no tienen idea del estado de sus proyectos. Estos son ejemplos de un intento por seguir los pasos extremadamente complejos de actividades mediante modelos demasiado simples. Tenga en cuenta que aunque un proyecto [grande] esté en la fase de codificación, hay personal de ese proyecto implicado en actividades asociadas generalmente a muchas fases de desarrollo simultáneamente. Por ejemplo: El personal está escribiendo requisitos, diseñando, codificando, haciendo pruebas y probando la integración [todo al mismo tiempo].

Se puede representar en forma de esquema como una serie de actividades técnicas importantes, tareas, y estados asociados a ellas.

Define una serie de acontecimientos que disparan transiciones de estado a estado para cada una de las actividades de la ingeniería de Software.

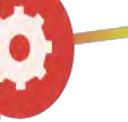


Ventajas

- Excelente para proyectos en los que se conforman grupos de trabajo independientes.
- Proporciona una imagen exacta del estado actual de un proyecto.

Desventajas

- Si no se dan las condiciones señaladas no es aplicable.
- Si no existen grupos de trabajo no se puede trabajar en este método



En realidad, el modelo de proceso concurrente es aplicable a todo tipo de desarrollo de software y proporciona una imagen exacta del estado actual de un proyecto.

El modelo de proceso concurrente se utiliza a menudo como el paradigma de desarrollo de aplicaciones Cliente/Servidor.

Un sistema cliente/servidor se compone de un conjunto de componentes funcionales. Cuando se aplica a cliente/servidor, el modelo de proceso concurrente define actividades en dos dimensiones

UNIDAD II. PROCESO DE GESTIÓN DE PROYECTO



2.1 Ámbito del software



2.2 Estudio de factibilidad



2.3 Análisis de riesgo



2.4 Recursos



2.5 Estimación



2.6 Planificación del proyecto



2.7 Supervisión y control del plan de proyecto

Ámbito de Software

La primera actividad de la planificación del proyecto de software es determinar su ámbito.

Se evalúan la función y el rendimiento que se asignaron al software durante la ingeniería del sistema, para establecer un ámbito de proyecto que no sea ambiguo, ni incomprensible para directivos y técnicos.

El ámbito del software describe el control y los datos a procesar, la función, el rendimiento, las restricciones, las interfaces y la fiabilidad; Se evalúan las funciones descritas en la declaración del ámbito, y en algunos casos se refinan para dar más detalles antes del comienzo de la estimación.

Dado que las estimaciones del coste y de la planificación temporal están orientadas a la función, muchas veces es útil llegar a un cierto grado de descomposición. Las consideraciones de rendimiento abarcan los requisitos de tiempo de respuesta y de procesamiento.

Las restricciones identifican los límites del software originados por el hardware externo, por la memoria disponible y por otros sistemas existentes.

Obtención de la información necesaria

Al principio de un proyecto de software las cosas siempre están un poco borrosas.

Se ha definido una necesidad y se han enunciado las metas y objetivos básicos, pero todavía no se ha establecido la información necesaria para definir el ámbito (prerrequisito para la estimación).

La técnica utilizada con más frecuencia para acercar al cliente y al desarrollador, y para hacer que comience el proceso de comunicación es establecer una reunión o una entrevista preliminar.

¿Cómo debería empezar la comunicación entre el desarrollador y el cliente?

Los autores Gause y Weinberg sugieren que el analista comience haciendo preguntas de contexto libre.

Es decir, una serie de preguntas que lleven a un entendimiento básico del problema, las personas que están interesadas en la solución, la naturaleza de la solución que se desea y la efectividad prevista del primer encuentro.

El primer conjunto de cuestiones de contexto libre se centran en el cliente, en los objetivos globales y en los beneficios.

El primer conjunto de cuestiones de contexto libre se centran en el cliente, en los objetivos globales y en los beneficios.

Por ejemplo, el analista podría preguntar:

¿Quién está detrás de la solicitud de este trabajo?

¿Quién utilizará la solución?

¿Cuál será el beneficio económico de una buena solución?

La última serie de preguntas se centra en la efectividad de la reunión. Gause y Weinberg las llaman metacuestiones y proponen la lista siguiente (abreviada):

¿Es usted la persona apropiada para responder a estas preguntas?

¿Son relevantes mis preguntas para su problema?

¿Estoy realizando muchas preguntas?

¿Hay alguien más que pueda proporcionar información adicional?

¿Hay algo más que debiera preguntarle?

Viabilidad

Una vez se ha identificado el ámbito (con la ayuda del cliente), es razonable preguntarse:

¿Podemos construir el software de acuerdo a este ámbito?

¿Es factible el proyecto?

Con frecuencia, las prisas de los ingenieros de software sobrepasan estas preguntas (o están obligados a pasárselas por los clientes o gestores impacientes).

Estudio de factibilidad

Los autores Putnam y Myers tratan este aspecto cuando escriben que: no todo lo imaginable es factible ni siquiera en el software.

La factibilidad del software tiene cuatro dimensiones sólidas:

Tecnología ¿Es factible un proyecto técnicamente? ¿Está dentro del estado actual de la técnica?

Financiamiento ¿Es factible financieramente? ¿Puede realizarse a un coste asumible por la empresa de software y por el cliente?

Tiempo ¿Pueden los proyectos adelantarse a los de la competencia?

Recursos ¿La organización cuenta con los recursos suficientes para tener éxito?

La respuesta es sencilla depende de la experiencia, ya que puede que se haya hecho antes algún proyecto de este tipo o puede que no se tenga experiencia en el proyecto y por lo tanto no son fáciles.

Un equipo puede pasarse varios meses descubriendo cuáles son los requisitos principales y cuáles son aquéllos difíciles de implementar para una nueva aplicación.

El equipo de factibilidad debe asumir la arquitectura y el diseño de los requerimientos de alto riesgo hasta el punto de poder responder todas estas preguntas. En algunos casos, cuando el equipo proporciona respuestas negativas, esto puede negociarse con una reducción de los requisitos.

Putnam y Myers sugieren, de forma acertada, que el estudio del ámbito no es suficiente. Una vez que se ha comprendido el ámbito, tanto el equipo de desarrollo como el resto deben trabajar para determinar si puede ser construido dentro de las dimensiones reflejadas anteriormente.

Análisis de riesgo

El riesgo se mide por el grado de incertidumbre en las estimaciones cuantitativas establecidas por recursos, coste y planificación temporal. Si no se entiende bien el ámbito del proyecto o los requisitos del proyecto están sujetos a cambios, la incertidumbre y el riesgo son peligrosamente altos.

Lo que es más importante, el cliente y el panificador, deben tener presente que cualquier cambio en los requisitos del software significa inestabilidad en el coste y en la planificación temporal.

Sin embargo, un gestor de proyecto no debería obsesionarse con la estimación.

Otro aspecto a estudiar es la complejidad del proyecto la cual tiene un gran efecto en la incertidumbre, que es inherente en la planificación, la complejidad es una medida relativa que se ve afectada por la familiaridad con esfuerzos anteriores.

Ejemplo:

Se podría considerar una aplicación sofisticada de comercio electrónico como excesivamente compleja para un desarrollador que haya realizado su primera aplicación. Sin embargo para un equipo de software que desarrolle su enésimo sitio web de comercio electrónico podría considerarse sumamente fácil (una de tantas).

Se han propuesto una serie de medidas cuantitativas de la complejidad del software, tales medidas se aplican en el nivel de diseño y de codificación, y son difíciles de utilizar durante la planificación del software (antes de que exista un diseño o un código), al comienzo del proceso de planificación se pueden establecer otras valoraciones de complejidad más subjetivas como factores de ajuste de la complejidad del punto de función.

El tamaño del proyecto es otro factor importante que puede afectar a la precisión y a la eficiencia de las estimaciones, a medida que el tamaño aumenta, crece rápidamente la interdependencia entre varios elementos del software.

El problema de la descomposición, un enfoque importante hacia la estimación, se hace más difícil porque los elementos descompuestos pueden ser todavía excesivamente grandes.

El grado de incertidumbre estructural tiene también efecto en el riesgo de la estimación, se refiere a los requisitos se han definido, la facilidad con la que pueden subdividirse funciones, y la naturaleza jerárquica de la información que debe procesarse.

La disponibilidad de información histórica tiene una fuerte influencia en el riesgo de la estimación, al mirar atrás, podemos emular lo que se ha trabajado y mejorar las áreas en donde surgieron problemas reduciendo el riesgo global.

Recursos

DE PROYECTOS

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

La figura siguiente ilustra los recursos de desarrollo en forma de pirámide; En la base de la pirámide de recursos se encuentra el entorno de desarrollo -herramientas de hardware y software- que proporciona la infraestructura de soporte al esfuerzo de desarrollo.

En un nivel más alto se encuentran los componentes de software reutilizables, los bloques de software que pueden reducir drásticamente los costes de desarrollo y acelerar la entrega.

En la parte más alta de la pirámide está el recurso primario el personal.

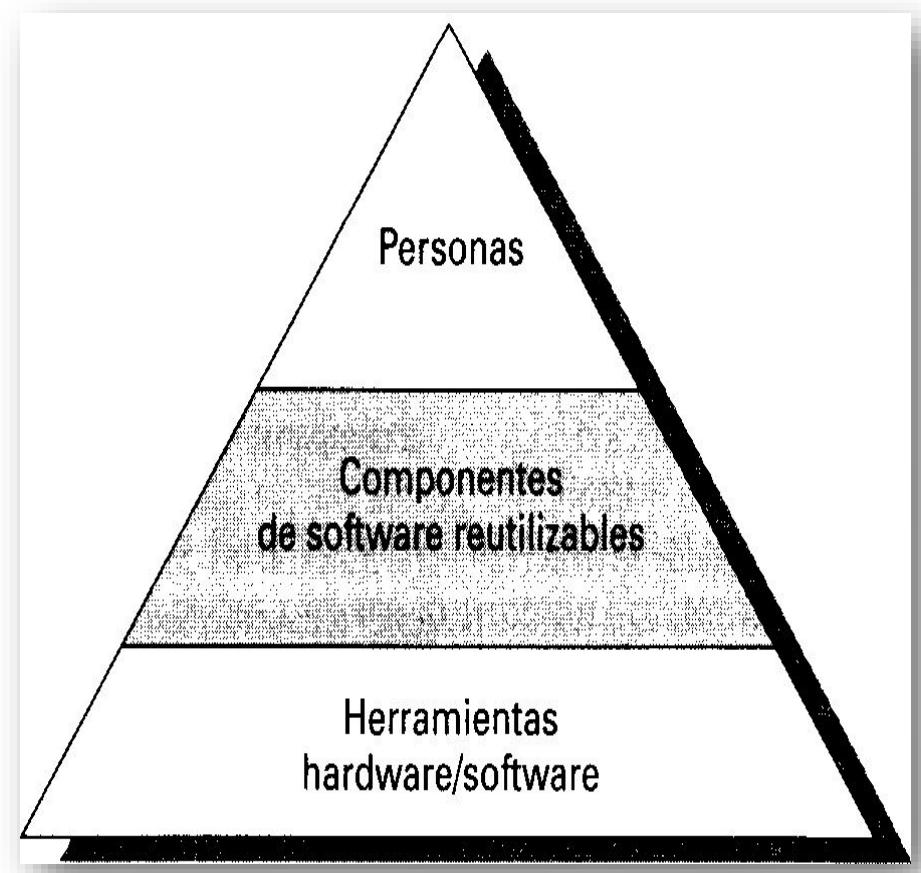
Cada recurso queda especificado mediante cuatro características:

Descripción del recurso, informe de disponibilidad, fecha cronológica en la que se requiere el recurso, tiempo durante el que será aplicado el recurso.

El encargado de la planificación comienza elevando el ámbito y seleccionando las habilidades que se requieren para llevar a cabo el desarrollo.

Hay que especificar tanto la posición dentro de la organización (por ejemplo: gestor, ingeniero de software experimentado, etc.) como la especialidad (por ejemplo: telecomunicaciones, bases de datos, cliente/servidor). Para proyectos relativamente pequeños una sola persona puede llevar a cabo todos los pasos de ingeniería del software, consultando con especialistas siempre que sea necesario.

El número de personas requerido para un proyecto de software sólo puede ser determinado después de hacer una estimación del esfuerzo de desarrollo.



Recursos de software reutilizables

La ingeniería del software basada en componentes (ISBC) destaca la reutilización, esto es, la creación y la reutilización de bloques de construcción de software. Dichos bloques de construcción, llamados componentes, deben establecerse en catálogos para una consulta más fácil, estandarizarse para una fácil aplicación y validarse para una fácil integración.

El autor Bennatan sugiere cuatro categorías de recursos de software que se deberían tener en cuenta a medida que se avanza con la planificación:

- Componentes ya desarrollados

Software existente se puede adquirir de una tercera parte o provenir de uno desarrollado internamente para un proyecto anterior, están listos para utilizarse en el proyecto actual y se han validado totalmente.

- Componentes ya experimentados

Especificaciones, diseños, código o datos de prueba desarrollados para proyectos anteriores que son similares al software que se va a construir para el proyecto actual, las modificaciones por tanto tendrán un riesgo relativamente bajo.

- Componentes con experiencia parcial

Especificaciones, diseños, código o datos de prueba desarrollados para proyectos anteriores que se relacionan con el software que se va a construir para el proyecto actual, pero que requerirán una modificación sustancial.

Componentes nuevos.

Los componentes de software que el equipo de software debe construir específicamente para las necesidades del proyecto actual.

Recursos de entorno

El entorno es donde se apoya el proyecto de software, llamado a menudo entorno de ingeniería del software, incorpora hardware y software.

El hardware proporciona una plataforma con las herramientas (software) requeridas para producir los productos que son el resultado de una buena práctica de la ingeniería del software.

Cuando se va a desarrollar un sistema basado en computadora (que incorpora hardware y software especializado), el equipo de software puede requerir acceso a los elementos en desarrollo por otros equipos de ingeniería.

Estimación

DE
PROYECTOS

Al principio, el coste del software constituía un pequeño porcentaje del coste total de los sistemas basados en computadora.

Un error considerable en las estimaciones del coste del software tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro de la mayoría de los sistemas informáticos.

Para sistemas complejos, personalizados, un gran error en la estimación del coste puede ser lo que marque la diferencia entre beneficios y pérdidas, sobrepasarse en el coste puede ser desastroso para el desarrollador.

La estimación del coste y del esfuerzo del software nunca será una ciencia exacta, son demasiadas las variables humanas, técnicas, de entorno, políticas que pueden afectar al coste final del software y al esfuerzo aplicado para desarrollarlo. Sin embargo, la estimación del proyecto de software puede dejar de ser un oscuro arte para convertirse en una serie de pasos sistemáticos que proporcionen estimaciones con un grado de riesgo aceptable como las siguientes:

Dejar la estimación para más adelante (obviamente, podemos realizar una estimación al cien por cien fiable tras haber terminado el proyecto).

Basar las estimaciones en proyectos similares ya terminados.

Utilizar técnicas de descomposición relativamente sencillas para generar las estimaciones de coste y de esfuerzo del proyecto.

Utilizar uno o más modelos empíricos para la estimación del coste y esfuerzo del software.

Las técnicas de descomposición utilizan un enfoque de divide y vencerás para la estimación del proyecto de software.

Mediante la descomposición del proyecto en sus funciones principales y en las tareas de ingeniería del software correspondiente, la estimación del coste y del esfuerzo puede realizarse de una forma escalonada idónea.

Se pueden utilizar los modelos empíricos de estimación como complemento de las técnicas de descomposición.

Métricas y modelos de estimación.

Estimación conforme al tamaño del software.

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

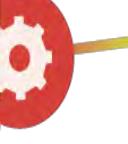
El grado en el que el planificador ha estimado el tamaño del software a construir, la habilidad para traducir la estimación del tamaño en esfuerzo humano, tiempo y dinero, el grado en el que el plan del proyecto refleja las habilidades del equipo de software, y la estabilidad de los requisitos del software y el entorno que soporta.

En el contexto de la planificación de proyectos, el tamaño se refiere a una producción cuantificable del proyecto de software. Si se toma un enfoque directo, el tamaño se puede medir en LDC (líneas de código), si se selecciona un enfoque indirecto, el tamaño se representa como PF (punto de función).

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

Tamaño en lógica difusa.

Se utilizan las técnicas aproximadas de razonamiento que son la piedra angular de la lógica difusa. Para aplicar este enfoque, el planificador debe identificar el tipo de aplicación, establecer su magnitud en una escala cuantitativa y refinar la magnitud dentro del rango original agregando la experiencia personal.



Tamaño en punto de función.

El planificador desarrolla estimaciones de características del dominio de información.

Tamaño de componentes estándar.

El software se compone de un número de componentes estándar que son genéricos para un área en particular de la aplicación. Por ejemplo, los componentes estándar para un sistema de información son: subsistemas, módulos, pantallas, informes, programas interactivos, programas por lotes, archivos, LDC e instrucciones para objetos.

El planificador de proyectos estima el número de incidencias de cada uno de los componentes estándar, y utiliza datos de proyectos históricos para determinar el tamaño de entrega por componente estándar.

Tamaño del cambio.

Este enfoque se utiliza cuando un proyecto comprende la utilización de software existente y que se debe modificar de alguna manera como parte de un proyecto. El planificador estima el número y tipo (por ejemplo: reutilización, añadir código, cambiar código, suprimir código) de modificaciones que se deben llevar a cabo.



Estimación basada en el problema

- Las líneas de código (LDC) y los puntos de función (PF) se describieron como medidas básicas a partir de las que se pueden calcular métricas de productividad.
- Los datos de LDC y PF se utilizan de dos formas durante la estimación del proyecto de software:
- Como una variable de estimación que se utiliza para dimensionar cada elemento del software

Como métricas de línea base recopiladas de proyectos anteriores y utilizadas junto con variables de estimación para desarrollar proyecciones de coste y de esfuerzo.

Las estimaciones de LDC y PF son técnicas de estimación distintas. A pesar de que ambas tienen varias características en común. El planificador del proyecto comienza con un enfoque limitado para el ámbito del software y desde este estado intenta descomponer el software en funciones que se pueden estimar individualmente.

Para cada función entonces se estiman las LDC y el PF (la variable de estimación). De forma alternativa, el planificador puede seleccionar otro componente para dimensionar clases u objetos, cambios o procesos de gestión en los que puede tener impacto.

Las métricas de productividad de línea base (por ejemplo: Líneas De Código/personas-mes o PF/pm, etc.) se aplican entonces para la variable de estimación adecuada y se extrae el coste o el esfuerzo de la función. Las estimaciones de función se combinan para producir una estimación global del proyecto entero.

Es decir, los proyectos se deberían agrupar por tamaño de equipo, área de aplicación, complejidad y otros parámetros relevantes.

Estimación basada en el proceso

La técnica más común para estimar un proyecto es basar la estimación en el proceso que se va a utilizar. Es decir, el proceso se descompone en un conjunto relativamente pequeño de actividades o tareas, y en el esfuerzo requerido para llevar a cabo la estimación de cada tarea.

Una vez que se mezclan las funciones del problema y las actividades del proceso, el planificador estima el esfuerzo (por ejemplo: personas-mes) que se requerirá para llevar a cabo cada una de las actividades del proceso de software en cada función.

El personal veterano se involucra de lleno con las primeras actividades y generalmente es mucho más caro que el personal junior, quienes están relacionados con las tareas de diseño posteriores, con la generación del código y con las pruebas iniciales.

Como último paso se calculan los costes y el esfuerzo de cada función, y la actividad del proceso de software.

Modelo COCOMO

Bany Boehm, introduce una jerarquía de modelos de estimación de software con el nombre de COCOMO, por Constructive Cost Model (Modelo Constructivo de Coste).

El modelo COCOMO original se ha convertido en uno de los modelos de estimación de coste del software más utilizados y estudiados en la industria y ha evolucionado a un modelo de estimación más completo llamado COCOMO 11 que trata las áreas siguientes:

Modelo de composición de aplicación

Utilizado durante las primeras etapas de la ingeniería del software, donde el prototipo de las interfaces de usuario, la interacción del sistema y del software, la evaluación del rendimiento, y la evaluación de la madurez de la tecnología son de suma importancia.

Modelo de fase de diseño previo

Utilizado una vez que se han estabilizado los requisitos y que se ha establecido la arquitectura básica del software.

Modelo de fase posterior a la arquitectura

Dentro de la jerarquía del modelo hay tres opciones de tamaño distintas: puntos objeto, puntos de función, y líneas de código fuente.

El modelo se clasifica en uno de los tres niveles de complejidad (eso es, básico, intermedio, o avanzado).

En esencia, la complejidad es una función del número y origen de las tablas de datos del cliente y servidor necesario para generar la pantalla o el informe y el número de vistas o secciones presentadas como parte de la pantalla o del informe.

Una vez que se ha determinado la complejidad, se valora el número de pantallas, informes, y componentes de acuerdo con la Tabla 5.1. La cuenta de punto objeto se determina multiplicando el número original de instancias del objeto por el factor de peso de la tabla 5.1 y se suman para obtener la cuenta total de punto objeto. Cuando se va a aplicar el desarrollo basado en componentes o la reutilización de software en general, se estima el porcentaje de reutilización (%reutilización) y se ajusta la cuenta del punto objeto:

$$PON = (\text{puntos objeto}) \times [(100 - \% \text{reutilización})/100]$$

donde PON significa puntos objeto nuevos.

PON calculado, se debe calcular la proporción de productividad.

La tabla presenta la proporción de productividad para los diferentes niveles de experiencia del desarrollador y de madurez del entorno de desarrollo.

Tipo de objeto	Peso de la Complejidad		
	Básico	Intermedio	Avanzado
Pantalla	1	2	3
informe	2	5	a
Componente L3G			10



Experiencia/capacidad del desarrollador	Muy baja	Baja	Normal	Alta	Muy alta
Madurez/capacidad del entorno	Muy baja	Baja	Normal	Alta	Muy alta
PROD	4	7	13	25	50

La ecuación del software

La ecuación del software es un modelo multivariable dinámico que asume una distribución específica del esfuerzo a lo largo de la vida de un proyecto de desarrollo de software. El modelo se ha obtenido a partir de los datos de productividad para unos 4.000 proyectos actuales de software, un modelo de estimación tiene esta forma:

$$E = [LDC \times B^{0,333}/P]^3 \times (1/t^4)$$

Una vez determinada la proporción de productividad, se puede obtener una estimación del esfuerzo del proyecto:

$$\text{Esfuerzo estimado} = PON / PROD$$

En modelos COCOMO 11 más avanzados", se requiere una variedad de factores de escala, conductores de coste y procedimientos de ajuste.

Donde:

E = esfuerzo en personas-mes o personas-año

t = duración del proyecto en meses o años

B = factor especial de destrezas

P = parámetro de productividad que refleja:

Madurez global del proceso y de las prácticas de gestión.

Métricas Orientadas a la Función

Las métricas del software orientadas a la función utilizan una medida de la funcionalidad entregada por la aplicación como un valor de normalización.

Ya que la funcionalidad no se puede medir directamente, se debe derivar indirectamente mediante otras medidas directas.

Las métricas orientadas a la función fueron propuestas por primera vez por Albretch, quien sugirió una medida llamada punto defunción.

Los puntos de función se derivan con una relación empírica según las medidas contables (directas) del dominio de información del software y las evaluaciones de la complejidad del software.

Los puntos de función se calculan completando la tabla de la Figura 4.5. Se determinan cinco características de dominios de información y se proporcionan las cuentas en la posición apropiada de la Tabla.

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

Número de entradas de usuario.

Se cuenta cada entrada de usuario que proporciona diferentes datos orientados a la aplicación. Las entradas se deberían diferenciar de las peticiones, las cuales se cuentan de forma separada.



Número de salidas de usuario.

Se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En este contexto la salida se refiere a informes, pantallas, mensajes de error, etc. Los elementos de datos particulares dentro de un informe no se cuentan de forma separada.

Número de peticiones de usuario.

Una petición se define como una entrada interactiva que produce la generación de alguna respuesta del software inmediata en forma de salida interactiva. Se cuenta cada petición por separado.

Número de archivos.

Se cuenta cada archivo maestro lógico (esto es, un grupo lógico de datos que puede ser una parte de una gran base de datos o un archivo independiente).

Número de interfaces externas.

Se cuentan todas las interfaces legibles por la máquina (por ejemplo: archivos de datos de cinta o disco) que se utilizan para transmitir información a otro sistema.

Una vez que se han recopilado los datos anteriores, a la cuenta se asocia un valor de complejidad, las organizaciones que utilizan métodos de puntos de función desarrollan criterios para determinar si una entrada en particular es simple, media o compleja siendo algo subjetivo.



Para calcular puntos de función (PF), se utiliza la relación siguiente:

$$PF = \text{cuenta-total} \times [0,65 + 0,01 \times 6(F_i)]$$

En donde cuenta-total es la suma de todas las entradas PF obtenidas de la figura 4.5.

F_i ($i = 1$ a 14) son valores de ajuste de la complejidad según las respuestas a las siguientes preguntas

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



Parámetros de medición	Cuenta	Factor de ponderación			=	
		Simple	Medio	Complejo		
Número de entradas de usuario		x 3	4	6	=	
Número de salidas de usuario		x 4	5	7	=	
Número de peticiones de usuario		x 3	4	6	=	
Número de archivos		x 7	10	15	=	
Número de interfaces externas		x 5	7	10	=	
Cuenta total		→				

Calculo de los puntos de función

Cada una de las preguntas anteriores es respondida usando una escala con rangos desde 0 (no importante o aplicable) hasta 5 (absolutamente esencial). Los valores constantes de la ecuación y los factores de peso que se aplican a las cuentas de los dominios de información se determinan empíricamente.

Una vez que se han calculado los puntos de función, se utilizan de forma análoga a las LDC como forma de normalizar las medidas de productividad, calidad y otros atributos del software.



Planificación del proyecto

El objetivo del gestor del proyecto es definir todas las tareas del proyecto, construir una red que describa sus interdependencias, identificar las tareas que son críticas dentro de la red y después hacerles un seguimiento para asegurarse de que el retraso se reconoce de inmediato. Para conseguirlo, el gestor debe tener una planificación temporal que se haya definido con un grado de resolución que le permita supervisar el progreso y controla el proyecto.

La planificación temporal de un proyecto de software es una actividad que distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto, asignando el esfuerzo a las tareas específicas de la ingeniería del software.

La planificación temporal identifica las principales actividades de la ingeniería de software y las funciones del producto a las que se aplican, se identifican y programan las tareas del software específicas (requeridas para realizar una actividad).

La planificación temporal para proyecto de desarrollo de software puede verse desde dos perspectivas bastante diferentes.

En la primera se ha establecido ya (irrevocablemente) una fecha final de entrega de un sistema basado en computadora. La organización del software esta limitada a distribuir el esfuerzo dentro del marco de tiempo previsto.

El segundo punto de vista de la planificación asume que se han estudiado unos límites cronológicos aproximados pero que la fecha final será establecida por la organización de la ingeniería del software.

El esfuerzo se distribuye para conseguir el mejor empleo de los recursos, y se define una fecha final después de un cuidadoso análisis del software, como todas las áreas de la ingeniería del software, la planificación temporal de proyectos del software se guía por unos principios básicos:

Compartimentación: El proyecto debe dividirse en un número de actividades y tareas manejables.

Interdependencia: Las actividades no pueden comenzar hasta que el resultado de otras este disponible y otras pueden ocurrir independientemente.

Asignación de tiempo: A cada tarea que se vaya a programar se le debe de asignar cierto numero de unidades de trabajo y se le debe asignar una fecha de inicio y otra de finalización.

Validación de esfuerzo.

Todos los proyectos tienen un número definido de miembros de la plantilla.

Responsabilidades definidas.

Cada tarea que se programe debe asignarse a un miembro del equipo específico.

Resultados definidos.

Cada tarea programada debe tener un producto.



Calendario de actividades

Designa la programación predeterminada de los trabajos para todos los recursos asignados al proyecto. Puede establecer el calendario del proyecto para indicar un periodo no laborable (como los días festivos de la organización).

Para la planificación es un instrumento que permite elegir las mejores opciones (la mejor forma de lograr objetivos), sobre todo ayudara a fijar correctamente el curso del proyecto, esto incluye actividades como:

Seleccionar el equipo de análisis.

Asignación de proyectos apropiados para cada miembro del equipo.

Estimación del tiempo que cada tarea requiere para su ejecución.

Programación del proyecto para que las tareas concluyan oportunamente.

Pasos para elaborar el programa:

Listar las actividades en columna.

Disponer el tiempo disponible para el proyecto e indicarlo.

Calcular el tiempo para cada actividad.

Indicar estos tiempos en forma de barras horizontales

Reordenar cronológicamente.

Ajustar tiempo o secuencia de actividades.

Simbología

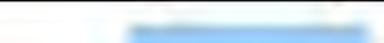
Nombre	Aspecto
Tarea	
División	
Progreso	
Hito	
Resumen	
Tarea resumida	
División resumida	
Hito resumido	
Progreso resumido	
Resumen del proyecto	



Diagrama de Gantt

Gráfica de Gantt o carta Gantt es una herramienta que permite modelar la planificación de las tareas necesarias para la realización de un proyecto, cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

Esta herramienta fue inventada por Henry L. Gantt en 1917.

Características.

El diagrama de Gantt consiste en una representación gráfica sobre dos ejes; en el vertical se disponen las tareas del proyecto y en el horizontal se representa el tiempo.

Cada actividad se representa mediante un bloque rectangular cuya longitud indica su duración. Los bloques correspondientes a tareas del camino crítico acostumbran a llenarse en otro color.

El Gráfico de Gantt no ofrece condiciones para el análisis de opciones, ni toma en cuenta factores como el costo.

Para la planificación de actividades relativamente simples, el gráfico de Gantt representa un instrumento de bajo costo y extrema simplicidad en su utilización.

Ejemplo

ETAPA	ACTIVIDAD	TIEMPO EN SEMANAS				
		1	2	3	4	5
1	Recopilación de la información.					
2	Análisis de la información.					
3	Elaboración del proyecto preliminar.					
4	Revisión y corrección					
5	Presentación del proyecto final y validación.					
6	Instrumentación.					



Diagrama de Pert

La Técnica de Revisión y Evaluación de Proyectos, comúnmente abreviada como PERT, es un modelo para la administración y gestión de proyectos inventado en 1958 por la Oficina de Proyectos Especiales de la Marina de Guerra del Departamento de Defensa de los EUA.

PERT es básicamente un método para analizar las tareas involucradas en completar un proyecto dado, especialmente el tiempo para completar cada tarea, e identificar el tiempo mínimo necesario para completar el proyecto total.

La parte más famosa de PERT son las Redes PERT, diagramas de líneas de tiempo que se interconectan. PERT está diseñado para proyectos de gran escala, que se ejecutan de una vez, complejos y no rutinarios.

Una malla PERT permite planificar y controlar el desarrollo de un proyecto. Las redes PERT trabajan con tiempos probabilísticos. Normalmente para desarrollar un proyecto específico lo primero que se hace es determinar, en una reunión multidisciplinaria, cuáles son las actividades que se deberá ejecutar para llevar a feliz término el proyecto, cuál es la precedencia entre ellas y cuál será la duración esperada de cada una.

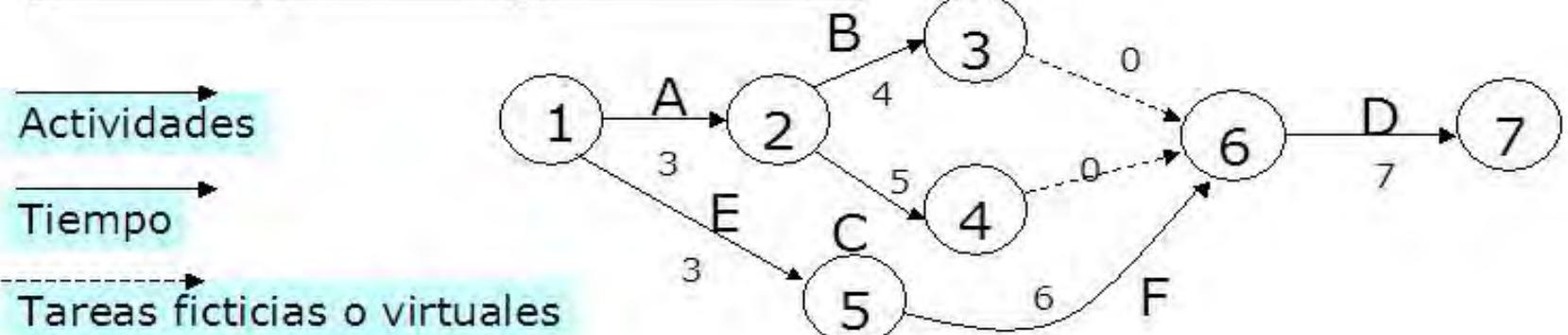
Para definir la precedencia entre actividades se requiere de una cierta cuota de experiencia profesional en el área, en proyectos afines.



Ejemplo

Antecesor	Predecesor	Estimación días
A		3
B	A	4
C	A	5
D	B,C,F	7
E		3
F	E	6

Los nodos son hitos, sucesos



Supervisión y control del plan del proyecto

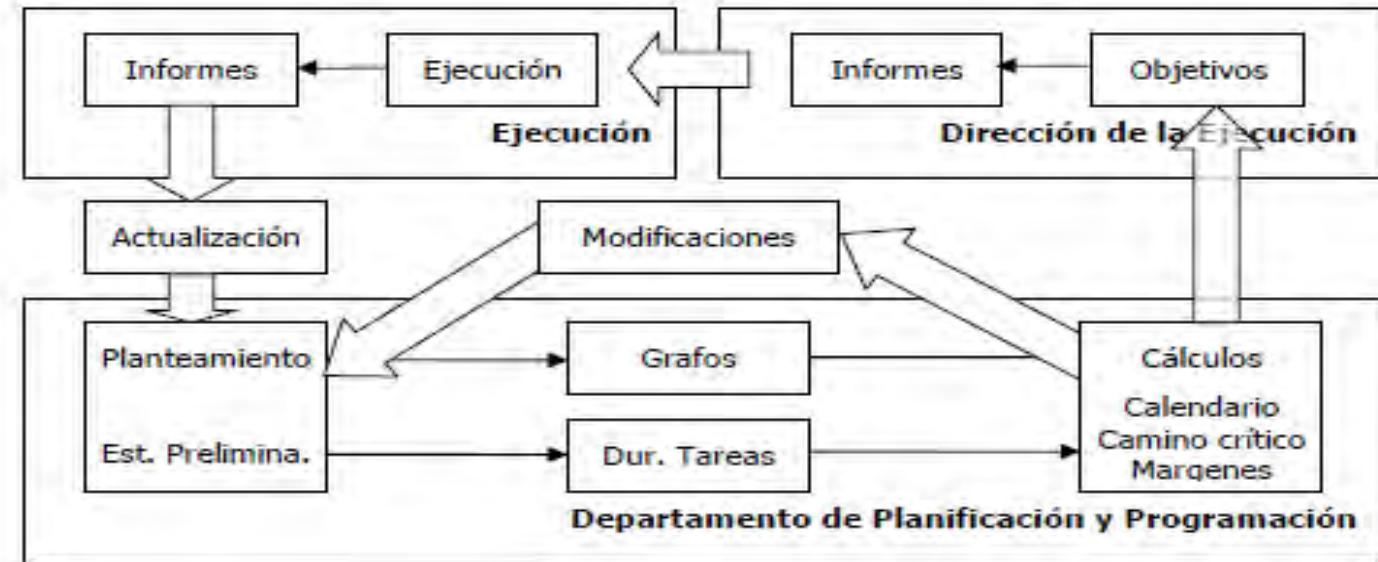
Una vez aprobada una planificación de un proyecto, el paso siguiente necesario es el de hacer cumplir esta planificación, y proceder a la ejecución y desarrollo del mismo.

Para ello, como hemos adelantado previamente, es necesaria la actuación de un equipo de personas que se encarguen de la puesta en funcionamiento y supervisión de todas las actividades del proyecto, así como el control de la correcta implementación de los planes realizados, entras palabras, medir el progreso del proyecto.

Al frente de este equipo se situará un director o jefe del proyecto, que será el último responsable de la coordinación del equipo de gestión, y de todos las partes involucradas en el proyecto, así como del control de las actividades, tareas, costes, uso de los recursos... Para poder llevar a cabo estas tareas, es necesario disponer de una correcta programación, así como de las herramientas adecuadas de control.

Control y Seguimiento del Proyecto

Una de las responsabilidades esenciales del jefe de proyecto y de las instancias jerárquicas involucradas en la operación es el seguimiento y control de los resultados, así como la adopción de las medidas correctoras que sean necesarias para rencausar la situación cuando se requiera. El organigrama funcional expresa el circuito de interrelaciones entre la ejecución, la programación y la dirección para efectuar el control del proyecto.



El control sólo es posible si previamente los objetivos del proyecto han sido definidos con la suficiente claridad y precisión, consiste en comparar lo que acontece en la realidad con lo que anteriormente se había previsto, y tomar las decisiones que permitan corregir las desviaciones que se hayan producido con el objeto de alcanzar las metas fijadas.

Incluye:

- Medición de las actividades en curso del proyecto
- Las variables de seguimiento del proyecto (costes, esfuerzo, alcance, etc.)
- Identificar las acciones correctivas para abordar las cuestiones y los riesgos
- Influenciar en los factores que podrían eludir el control

UNIDAD III. METODOLOGÍAS



3.1 Metodologías estructuradas



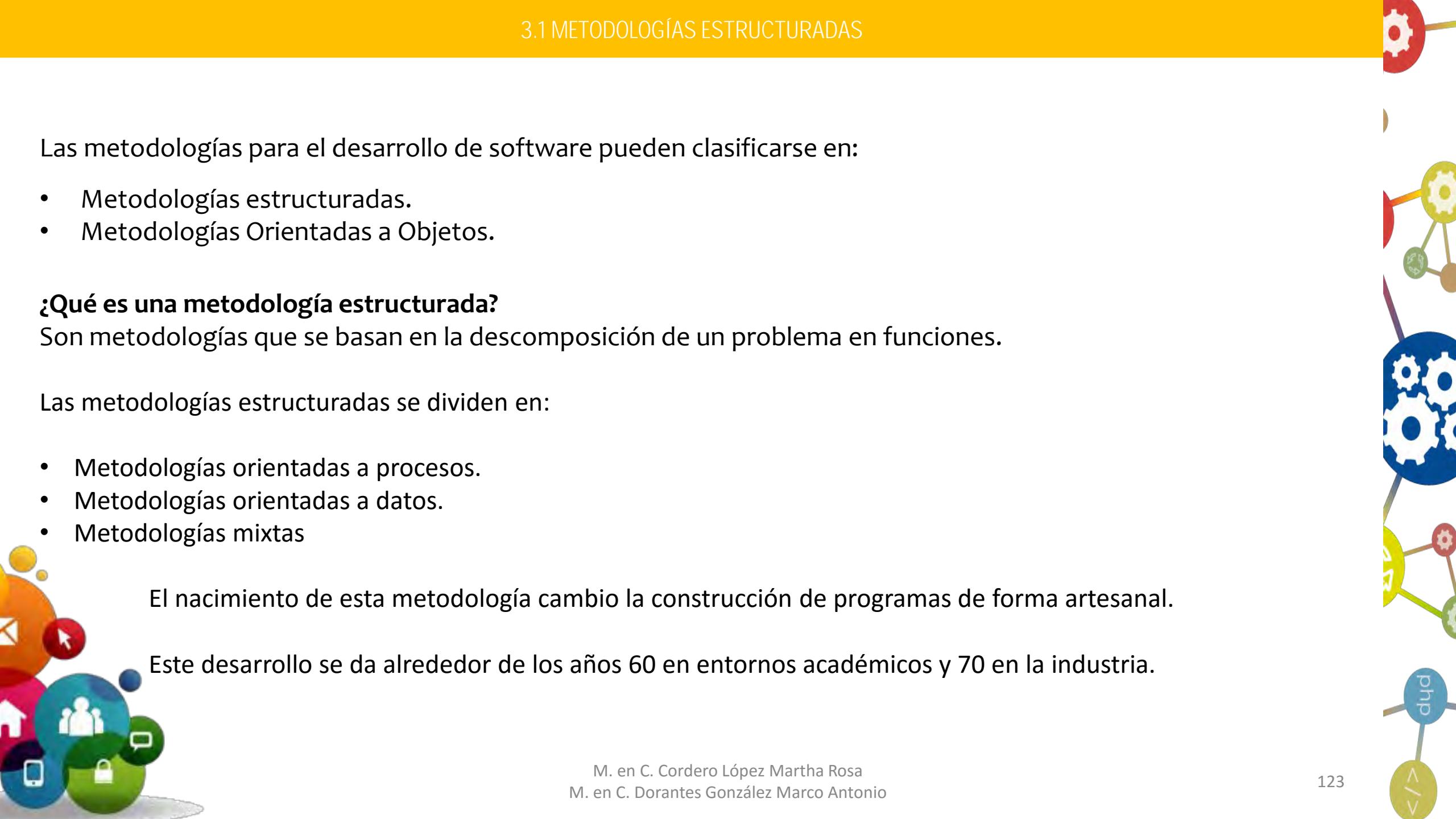
3.2 Metodologías Orientadas a Objetos



3.3 Proceso Unificado de Rational



Metodologías estructuradas



Las metodologías para el desarrollo de software pueden clasificarse en:

- Metodologías estructuradas.
- Metodologías Orientadas a Objetos.

¿Qué es una metodología estructurada?

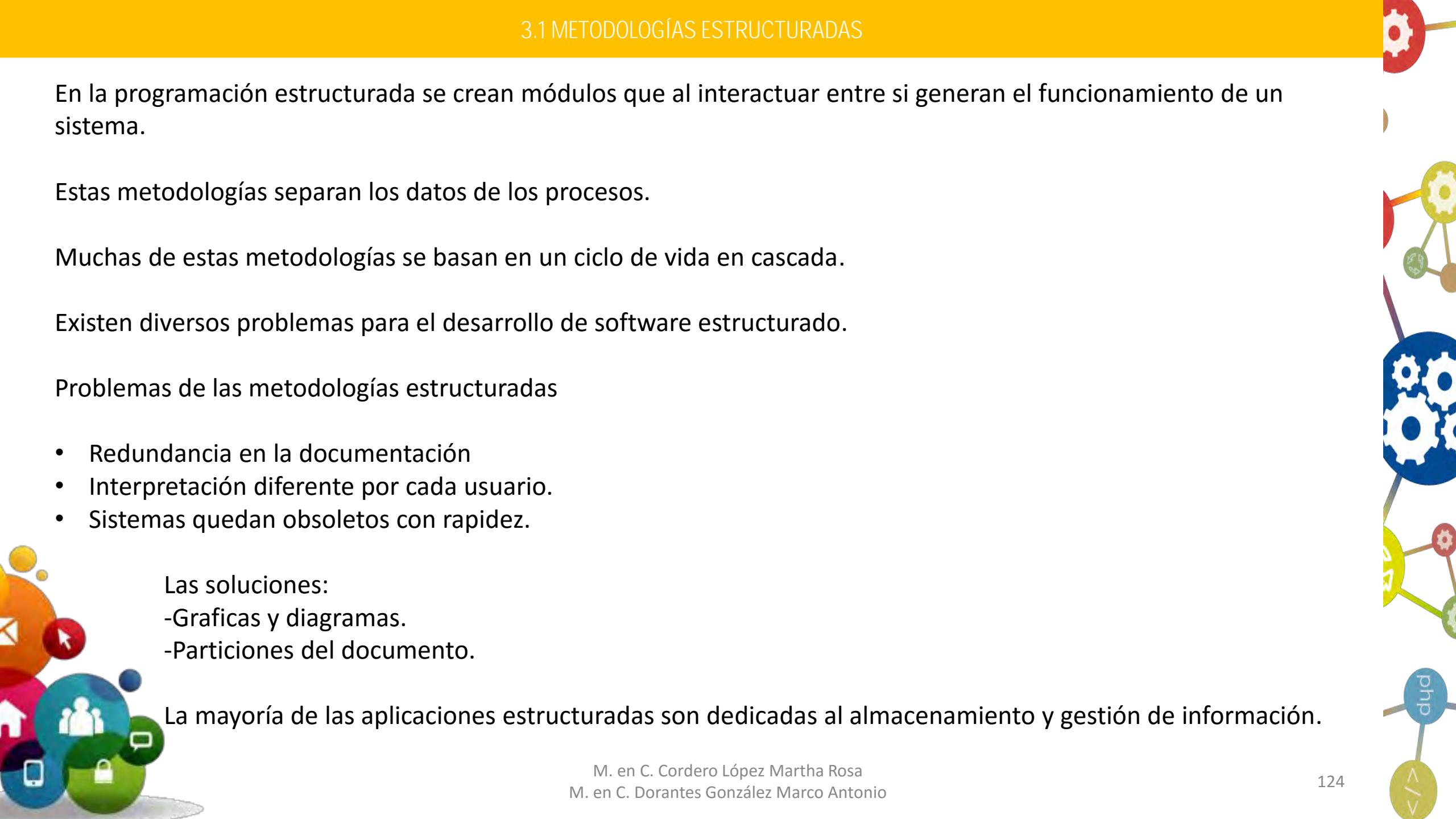
Son metodologías que se basan en la descomposición de un problema en funciones.

Las metodologías estructuradas se dividen en:

- Metodologías orientadas a procesos.
- Metodologías orientadas a datos.
- Metodologías mixtas

El nacimiento de esta metodología cambio la construcción de programas de forma artesanal.

Este desarrollo se da alrededor de los años 60 en entornos académicos y 70 en la industria.



En la programación estructurada se crean módulos que al interactuar entre si generan el funcionamiento de un sistema.

Estas metodologías separan los datos de los procesos.

Muchas de estas metodologías se basan en un ciclo de vida en cascada.

Existen diversos problemas para el desarrollo de software estructurado.

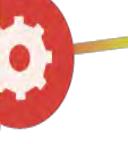
Problemas de las metodologías estructuradas

- Redundancia en la documentación
- Interpretación diferente por cada usuario.
- Sistemas quedan obsoletos con rapidez.

Las soluciones:

- Graficas y diagramas.
- Particiones del documento.

La mayoría de las aplicaciones estructuradas son dedicadas al almacenamiento y gestión de información.



Metodología Merise

La metodología Merise fue desarrollada en 1977 por el ministerio de industria francés.

La base de Merise comenzó en 1972 en la universidad de Aix en Provence.

Esta metodología esta integrada por: Análisis, Concepción y Gestión de proyectos.

La metodología esta conformada por 4 fases:

1.- Estudio preliminar.

Análisis de la situación

Propuesta de solución global (gestión, organización, decisiones del comité y directivo)

2.- Estudio detallado.

Análisis del sistema realizar estudios técnicos y presupuestos.

3.- Implementación.

Solución en un lenguaje de programación.

Evaluación de hardware y software

Pruebas

4.- Realización y puesta en marcha.
Instalación del sistema desarrollado.
Organización del personal por áreas.

Los ciclos de la metodología Merise están compuestos por:

Ciclo de abstracción.
Utiliza formalismos conceptuales, organizacionales y operacionales.

Ciclo de vida.
Utiliza periodos de concepción, realización y mantenimiento.

Ciclo de decisión.

- Elección de procedimientos de trabajo y tareas.
- Modificación de pantallas.



Metodología Yourdon

Esta metodología involucra análisis, desarrollo del diseño y mejora en la medición de la calidad del diseño de software.

Dentro de esta metodología en la sección del análisis y desarrollo del diseño se generan:

- Diagrama de flujo de datos.
- Diagrama de estructura. (diagrama jerárquico de procesos y su relación, es la herramienta mas importante de Yourdon)

En la medición y mejora del diseño de software:

- Se utilizan dos técnicas: acoplamiento y cohesión.

El acoplamiento mide el grado de independencia de cada modulo.

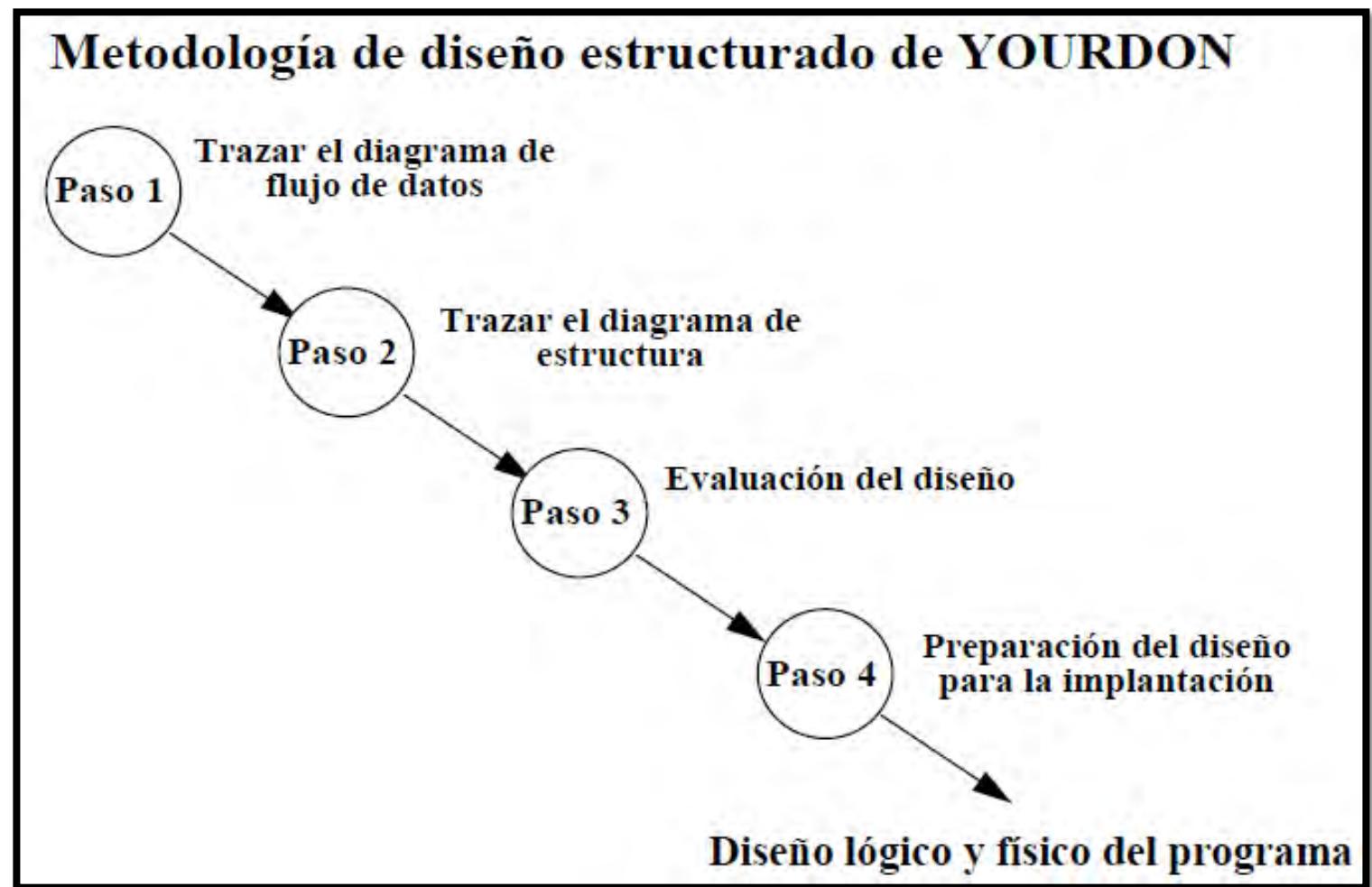
La cohesión mide la fuerza de la relación entre los elementos de cada modulo.

Lo ideal es que se tenga un bajo acoplamiento y un alto grado de cohesión.

En la implantación se hace:

- Diseño físico del sistema.

Diagrama





Metodología Gane Searson

La métrica Gane Sarson se comienza a utilizar en 1977.

Esta métrica es el resultado de varios años de práctica en la consultoría de análisis y diseño estructurado.

Es creada por la empresa MCAUTO/IST bajo el nombre de STRADIS SDM.

Para el uso y desempeño de esta metodología se utilizan los siguientes 5 pasos:

1.- Construir un modelo lógico en curso.

2.- Construir un modelo lógico del nuevo sistema lo cual involucra:

Diagramas de flujo de datos, Diccionario de datos, y especificaciones de los procesos.

También construir un modelo de datos que exprese en 3era forma normal (3FN) los datos almacenados.

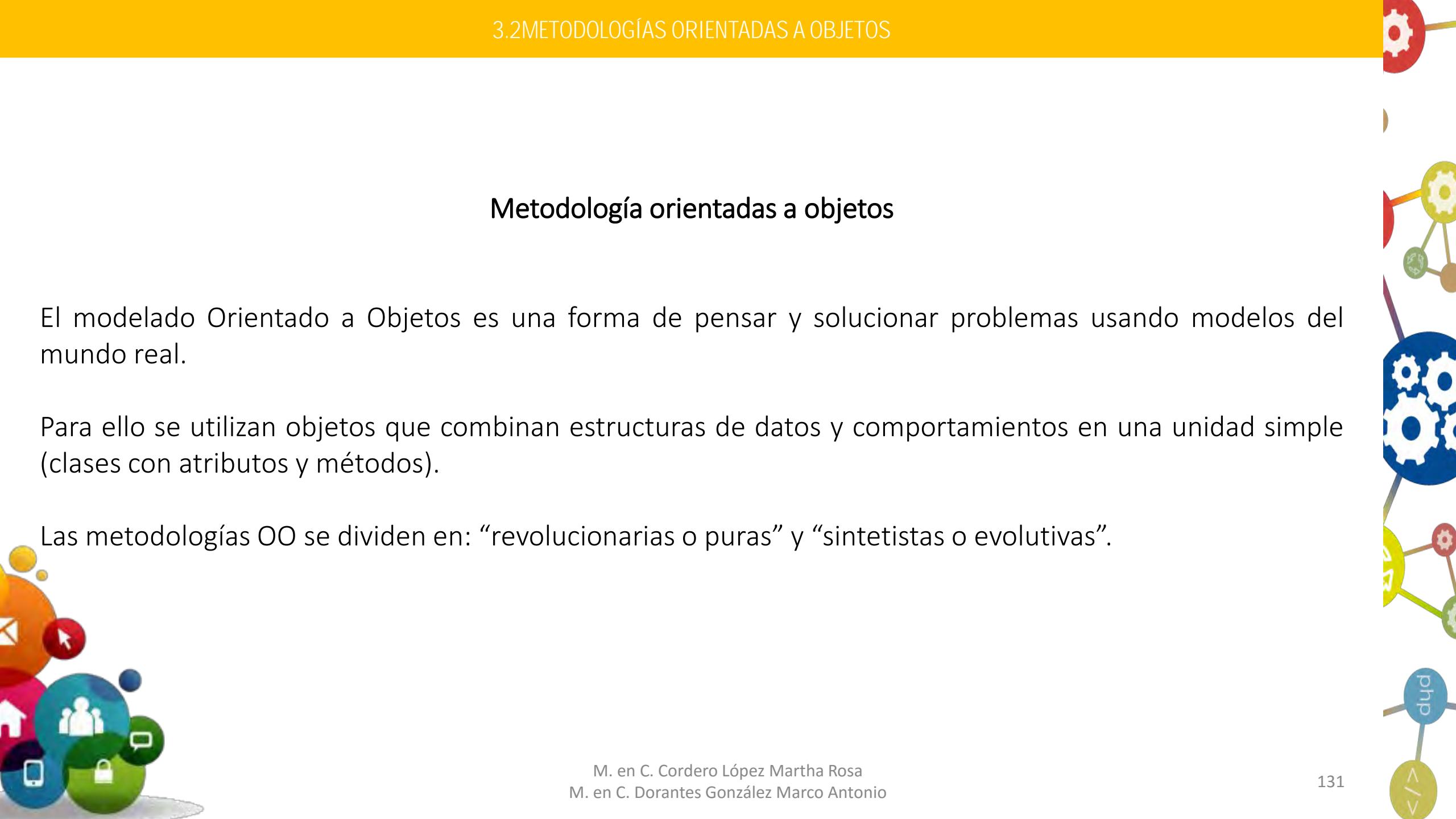
3.- Diseñar físicamente la BD.

4.- Crear un nuevo modelo físico del sistema.

5.- Empaquetar la especificación en subsistemas.



Metodologías Orientadas a Objetos



Metodología orientadas a objetos

El modelado Orientado a Objetos es una forma de pensar y solucionar problemas usando modelos del mundo real.

Para ello se utilizan objetos que combinan estructuras de datos y comportamientos en una unidad simple (clases con atributos y métodos).

Las metodologías OO se dividen en: “revolucionarias o puras” y “sintetistas o evolutivas”.

OMT

La metodología OMT (Técnica de Modelado de Objetos) es desarrollada por James Rumbaugh y sus colaboradores de General Electric en 1991.

Es una metodología que en su momento gozo de mucho prestigio para el desarrollo de sistemas dentro de la industria.

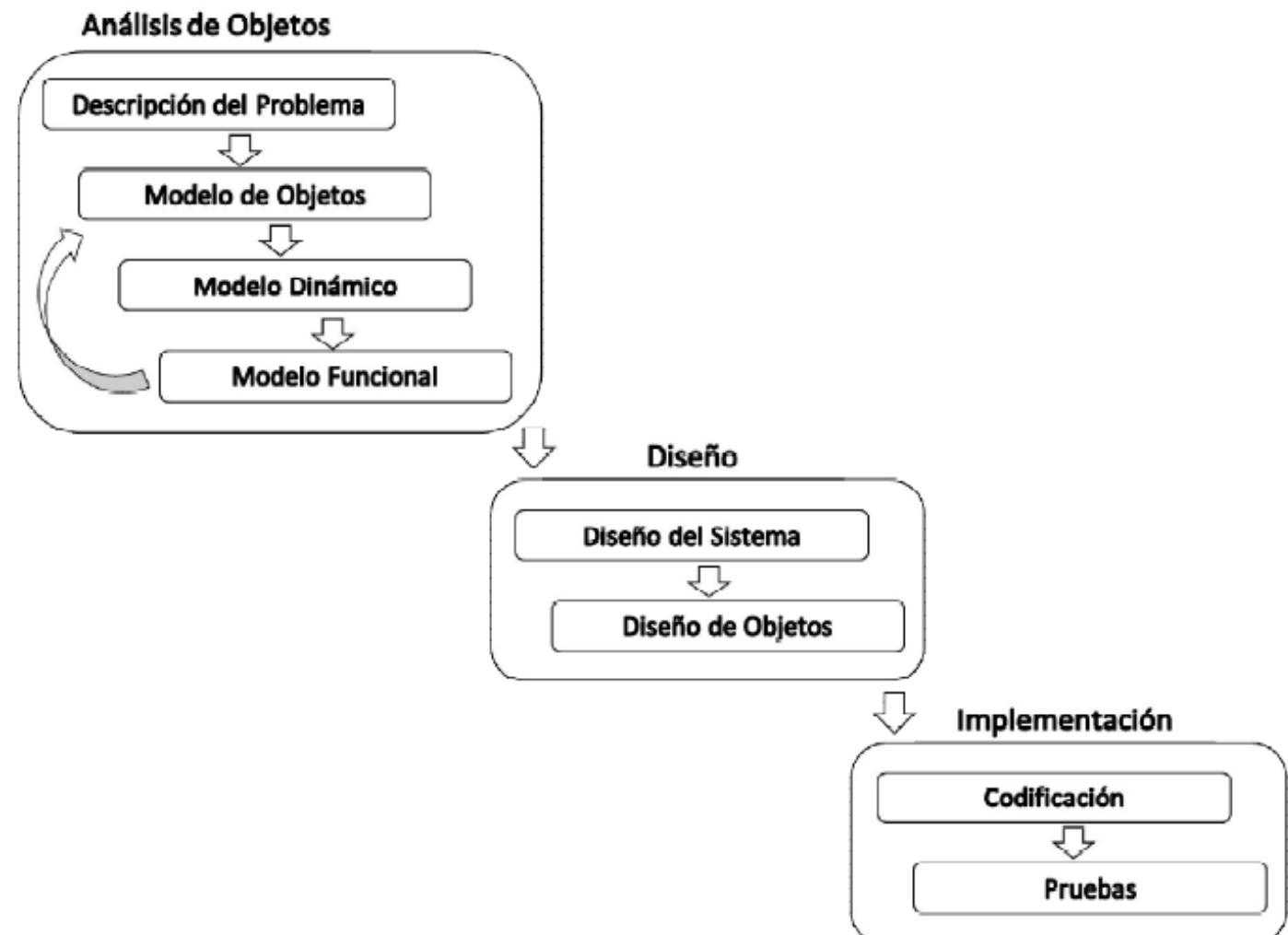
Esta metodología se apoya de 3 modelos para describir un sistema:

EL MODELO DE OBJETOS

EL MODELO DINÁMICO

EL MODELO FUNCIONAL

- DESCRIBE “QUE” ESTÁ CAMBIANDO
- CONTIENE DIAGRAMAS DE OBJETOS
- DESCRIBE “CUANDO” CAMBIA
- CONTIENE DIAGRAMAS DE ESTADO
- DESCRIBE “COMO” CAMBIA
- CONTIENE DIAGRAMAS DE FLUJOS DE DATOS





Ingeniería de software Orientado a Objetos OOSE.

Esta metodología es creada por Ivar Jacobson en 1992, la ingeniería de software orientado a objetos por su abreviatura se conoce como OOSE.

Consta de 5 etapas o modelos para la correcta realización y obtención de un sistema OO.

Primera Etapa: Análisis de Requerimientos o Modelo de requisitos.

Utiliza:

Diagrama de casos de uso.

Modelo de objeto de dominio: vista lógica del sistema que puede usarse para hacer los casos del uso.

Descripción de interfaces: participación del usuario en la creación de la interfaz, lógica del usuario con respecto al sistema.

Segunda Etapa: Análisis De Estructura o Modelo Ideal

Se define la estructura lógica del sistema independiente de la aplicación.

Y comienza la construcción del Sistema, se establece su robustez.

Tercera Etapa: Modelo de Plan o Modelo Real

Se estudia a fondo el lenguaje de programación en el cual se construirá el sistema y la forma de almacenar los datos que utilizará.

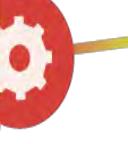
Se elaboran los diagramas de la interacción y gráficos de transición de estado.

Cuarta Etapa: Implementación o Modelo de Aplicación

Quinta Etapa: Modelo de Pruebas o Comprobación



Proceso Unificado de Rational



Proceso unificado y Proceso unificado de Rational RUP.

El Proceso Unificado es un proceso de desarrollo de software definido como:

“Conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema software”.
Contesta las siguientes preguntas:

¿Quién está haciendo? Trabajadores

¿Qué es lo que está haciendo? Componentes de software

¿Cuándo debe hacerlo? Fases del proceso

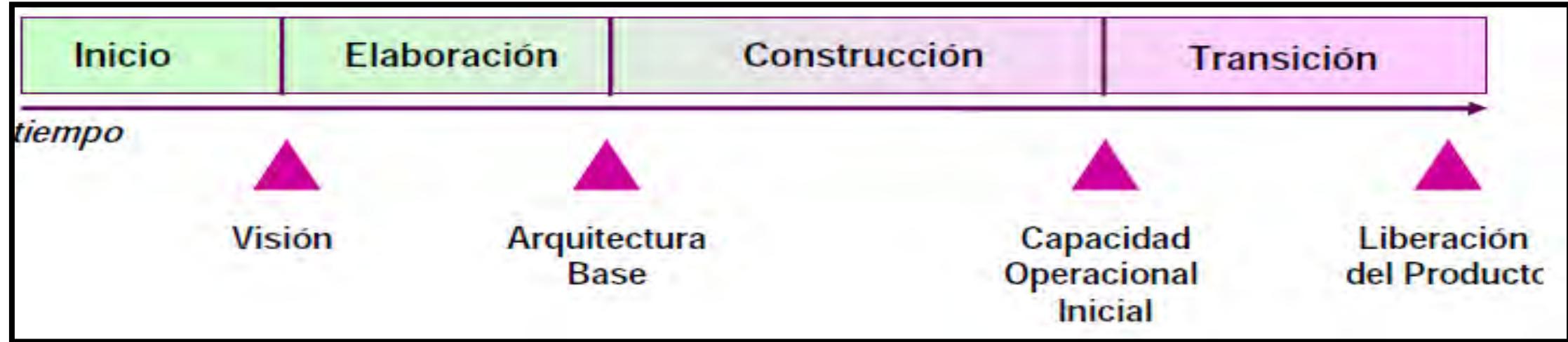
¿Cómo obtener un cierto objetivo? Encadenamiento de actividades



Sus características son:

- Iterativo e Incremental
- Basado en casos de uso
- Gerencia de requerimientos
- Centrado en la arquitectura
- Proceso configurable
 - Modelación Visual usando UML
 - Basado en componentes

Ciclo de vida



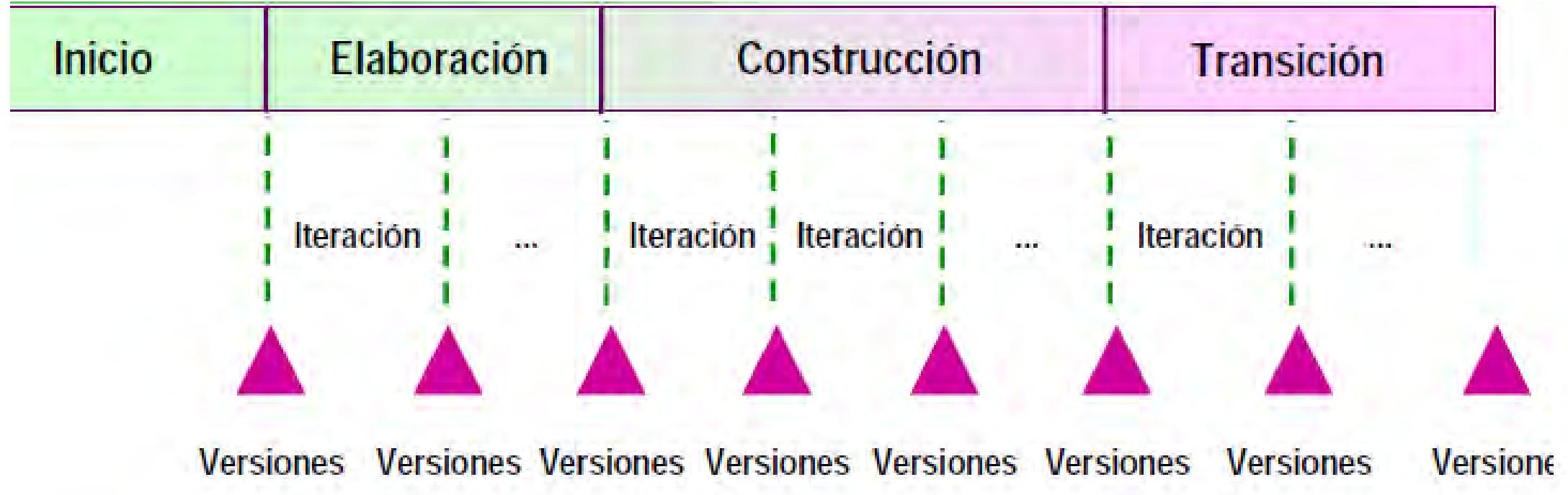
Inicio = Requerimientos, define alcance y factibilidad del proyecto.

Elaboración = Diseño, planifica el proyecto, especifica las características y la arquitectura base.

Construcción = implementación: Elaboración del proyecto.

Transición: Entrega del producto a los usuarios.

Ciclo de vida



Producción de diferentes versiones del sistema.

Para el proceso unificado de Rational o RUP tenemos que incluimos lo anteriormente visto para el proceso unificado pero se complementa lo siguiente:

El ciclo de vida de esta metodología es la misma que el proceso unificado pero sus iteraciones se realizan bajo algo conocido como disciplinas.

Las disciplinas pueden ser de dos tipos:

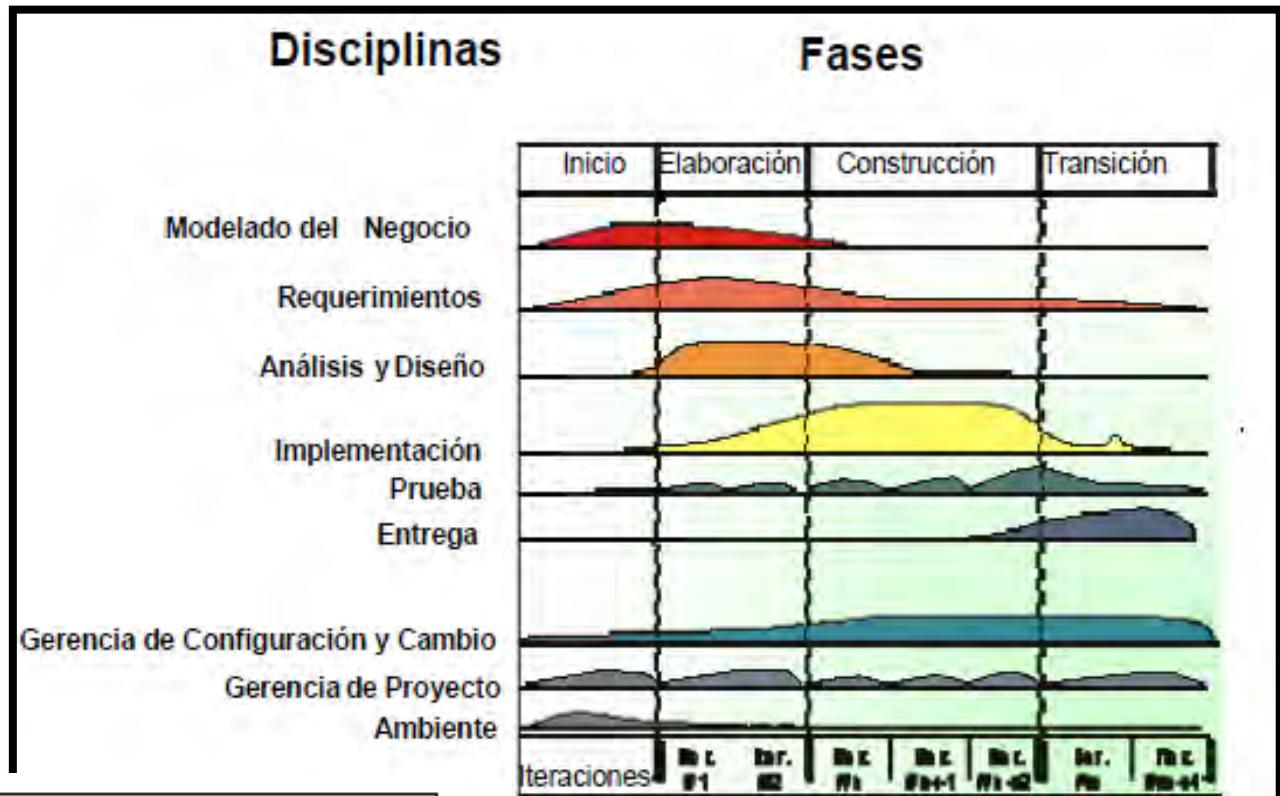
Disciplinas de desarrollo que están formadas por:

- Ingeniería de negocios.
- Requerimientos.
- Análisis y diseño.
 - Implementación.
 - Pruebas.

Disciplinas de soporte.

- Configuración y administración del cambio (versiones del proyecto).
- Administrando horarios y recursos del proyecto.
- Administrando ambiente de desarrollo.
- Distribución del proyecto.

Diagrama de Iteraciones y disciplinas o también conocidas como flujos de trabajo



Disciplina	Modelos
Requisitos	Modelo de Casos de Uso
Análisis	Modelo de Análisis
Diseño	Modelo de Diseño - Modelo de Despliegue
Implementación	Modelo de Implementación
Prueba	Modelo de Prueba

Metodologías agiles

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término ágil aplicado al desarrollo de software.

En esta reunión participan 17 expertos de la industria del software.

Su objetivo fue esbozar los principios que deberían permitir desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía ágil del desarrollo de software.

El manifiesto ágil consta de 4 puntos:

- Elegir primero el equipo de desarrolladores que el entorno en el que se va a desarrollar.
- Desarrollar software más funcional que una buena documentación.
- El cliente forma parte del equipo de desarrollo.
- La planificación del software es flexible y abierta para su fácil modificación



Tabla comparativa

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Programación extrema

Creada por Kent Beck famoso ingeniero de software estadounidense y participante del manifiesto ágil.

Esta metodología es conocida como Extreme Programming o XP.

Es la práctica más utilizada para la creación de software cuando se dispone de un proyecto y equipos pequeños y el plazo de entrega es demasiado limitado.

El equipo del desarrollo de software debe de contar con:

- Programador.
- Cliente.
- Encargado de pruebas (tester).
- Encargado de seguimiento (tracker, verifica si las estimaciones realizadas fueron un acierto).
- Entrenador (Coach, proporciona guías para el desarrollo del software).
- Consultor.
- Gestor (coordinación).

Esta metodología se basa en lo siguiente:

- La planificación:

Se da por pláticas frecuentes con el cliente, se definen un conjunto de enunciados especificando lo que se desea solucionar y obtener del sistema.

- Se desarrolla gradualmente (Entregas parciales del software).

- Su diseño debe de ser lo mas simple posible.

- Se generan pruebas para verificar el sistema.

- Programación en parejas: esto genera menos cantidad de errores en el sistema.

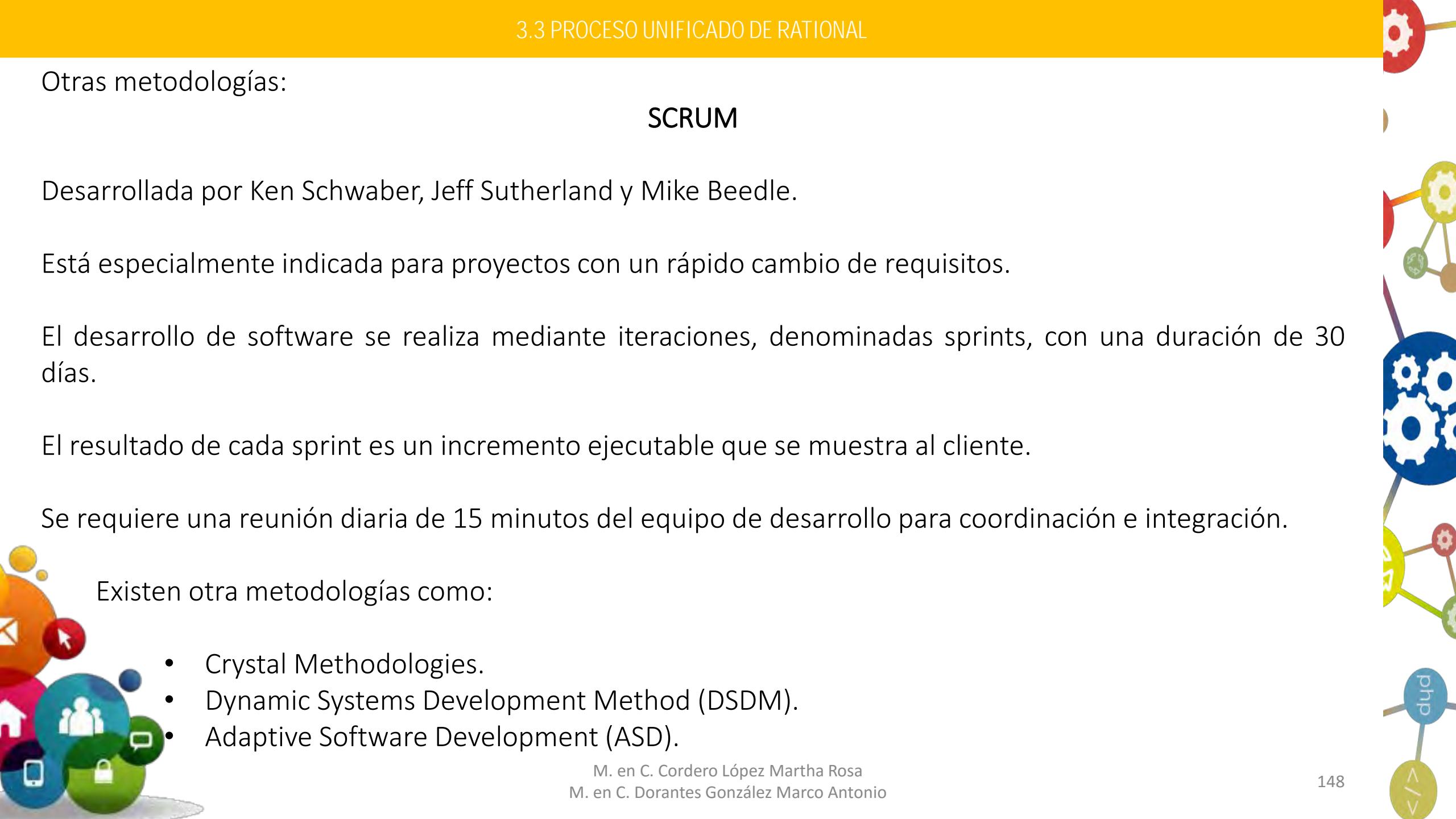
- Propiedad colectiva del código: cualquier programador puede cambiar cualquier parte en el código del sistema.

- Integrar el código de manera continua.

- Trabajar 40 horas por semana, no mas ni menos el trabajo extra desmotiva al equipo.

- Cliente siempre involucrado en el desarrollo del sistema.

- Programación en parejas: esto genera menos cantidad de errores en el sistema.
- Propiedad colectiva del código: cualquier programador puede cambiar cualquier parte en el código del sistema.
- Integrar el código de manera continua.
- Trabajar 40 horas por semana, no mas ni menos el trabajo extra desmotiva al equipo.
- Cliente siempre involucrado en el desarrollo del sistema.



Otras metodologías:

SCRUM

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle.

Está especialmente indicada para proyectos con un rápido cambio de requisitos.

El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días.

El resultado de cada sprint es un incremento ejecutable que se muestra al cliente.

Se requiere una reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Existen otras metodologías como:

- Crystal Methodologies.
- Dynamic Systems Development Method (DSDM).
- Adaptive Software Development (ASD).

UNIDAD IV. CALIDAD Y NORMAS DE CALIDAD



4.1 Conceptos de la calidad



4.2 Calidad de sistemas de información



4.3 Calidad del producto de software



4.4 Modelos y normas de calidad

Conceptos de la calidad

Conceptos de calidad

Calidad

La calidad se refiere a las características mensurables, cosas que se pueden comparar con estándares conocidos.

Los aspectos de calidad evaluados dentro del software son:

- Complejidad dicromático.
- Cohesión, número de puntos de función.
- Líneas de código.
- Etc.

Existen dos tipos de calidad en el software: calidad del diseño y calidad de concordancia.

Calidad de diseño.

Se refiere a las características que especifican los ingenieros para la elaboración del software (especificar forma de construirlo).

La calidad de diseño comprende los requisitos, especificaciones y el diseño del sistema.

Calidad de concordancia.

Es el grado de cumplimiento de las especificaciones de diseño durante su realización.

Cuanto mayor sea el grado de cumplimiento, más será el nivel de calidad de concordancia.

Control de calidad.

El control de calidad es una serie de inspecciones, revisiones y pruebas utilizadas a lo largo del proceso del software para asegurar que un producto cumple con los requisitos que dice tener.

Garantía de calidad.

Consiste en la auditoría de información.

La garantía de calidad busca proporcionar los datos necesarios sobre la calidad del producto, para ver si el producto está cumpliendo sus objetivos.

Coste de calidad

Incluye todos los costes generados en la búsqueda de la calidad.

Los costes de calidad se pueden dividir en:

Costes de prevención:

- planificación de la calidad.
- revisiones técnicas formales.
- equipo de pruebas.
- formación.

Costes de evaluación:

- inspección entre procesos.
- calibrado y mantenimiento de equipo.
- pruebas.

Los costes de fallos incluyen:

- revisión.
- reparación.
- análisis de fallos.

Calidad de los sistemas de información

Los requisitos del software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.

La ingeniería de software marca las pautas que marcan la calidad en el software.

En la calidad los requisitos de un software son: explícitos e implícitos.

Los requisitos explícitos son los requerimientos que el cliente pide en la elaboración del software.

Los requisitos implícitos son los requerimientos que no se nombran como el buen y adecuado mantenimiento del sistema.

Según la norma IEEE 1601 la calidad de un software se define como el grado que posee de rendimiento, fiabilidad o seguridad un software.

Calidad del producto software

La calidad en el desarrollo y el mantenimiento del software se ha convertido en uno de los principales objetivos estratégicos de las organizaciones.

Cada vez más, los procesos principales de las organizaciones y su supervivencia dependen de los sistemas informáticos.

Según estadísticas el Standish Group, CHAOS report en el 2004 afirma que solo el 51% de los proyectos de software tienen éxito (ver cuadro comparativo).

1995	2004
Se gastaron US\$250 billones* al año en tecnología de información	Se gastaron US\$ 255 billones al año en tecnología de información
31% de los proyectos se cancelaron antes de su terminación	15% de los proyectos se cancelaron antes de su terminación
53% de los proyectos costaron 189% más de sus estimados originales	34% de los proyectos costaron 43% más de los estimados originales
Se gastaron \$81 billones por los proyectos cancelados de software	Se gastaron \$55 billones por los proyectos cancelados de software
El 16% de los proyectos se terminaron en tiempo y en presupuesto	El 51% de los proyectos se terminaron en tiempo y en presupuesto

Modelos y normas de calidad

Modelos y normas de calidad.

Los sistemas de tecnologías de la información desempeñan un papel crítico en la práctica totalidad de las empresas.

En el desarrollo de software como otros servicios se necesita supervisión constante por parte de profesionales para mantener actualizado y en condiciones de funcionamiento optimas dicho servicio.

Para la supervisión del software esta pasa por un control de calidad, que es un conjunto de estándares denominados modelos de referencia.

Para la evaluación del software se contemplan:

- Los procesos.
- El producto final.

Los modelos de referencia para evaluar la calidad son los siguientes:

Modelos de referencia por ámbitos.

Ámbito = Procesos: CMMI, ISO/IEC 15504.

Ámbito = Producto: ISO/IEC 9126, 14598, 25000.

Modelos de referencia de calidad por producto.

Norma ISO/IEC 9126: Modelo de calidad del producto de software.

Norma ISO/IEC 14598: Calidad de la evaluación del producto de software.

Norma ISO/IEC 25000: denominada SQUARE.

ISO/IEC 9126.

Desarrolladas entre 1991 y 2001, estas normalizan la ingeniería de software conforme a la calidad del producto de software y definen lo siguiente:

- Características de calidad del software
- Métricas internas y externas.
- La calidad del producto está sujeta a las condiciones particulares de uso.



ISO 25000.

La familia de normas 25000 establece un modelo de calidad para el producto software además de definir la evaluación de la calidad del producto.

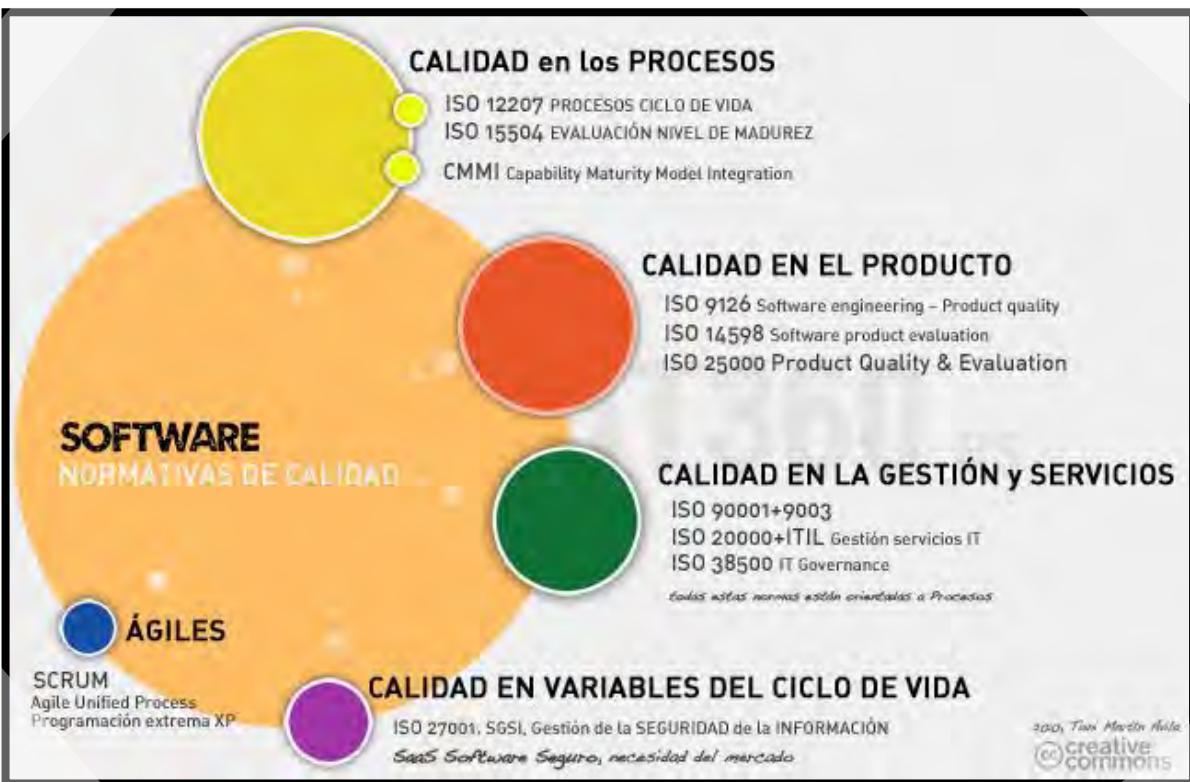
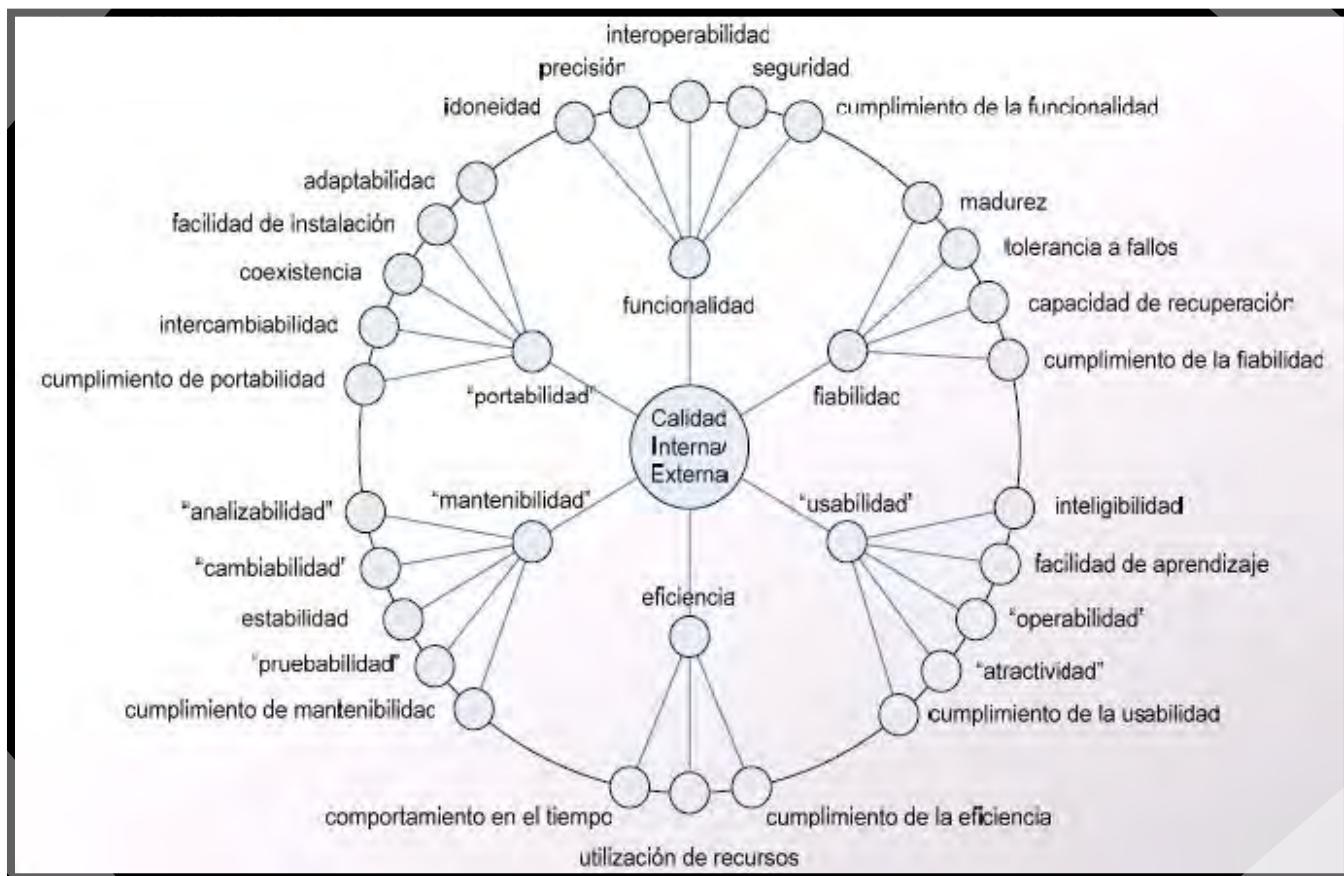


Diagrama de la calidad de un software.



ISO 9000

La familia de normas ISO 9000 evalúa Sistemas de Gestión de la calidad eficaz por sus siglas SGCE.

El objetivo de los SGCE es dirigir y controlar una organización con respecto a la calidad, logrando la realización de todas las actividades y obtener resultados planificados.



Los 8 principios de gestión de la calidad considerados en ISO 9000 son:

- 1.- Enfoque al cliente.
- 2.- Liderazgo.
- 3.- Participación del personal.
- 4.- Enfoque basado en procesos.
- 5.- Enfoque de sistema para la gestión.
- 6.- Mejora continua.
- 7.- Enfoque basado en hechos para tomar decisiones.
- 8.- Relaciones mutuamente beneficiosas.

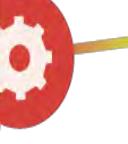
Fundamentos según ISO 9000 para tener un SGC.

ISO 9000	Fundamento
2.1	Base racional para los SGC.
2.2	Requisitos para los SGC y para los productos.
2.3	Enfoque de SGC.
2.4	Enfoque basado en procesos.
2.5	Política de calidad y objetivos de calidad.
2.6	Papel de la alta dirección dentro de un SGC.

ISO 9000	Fundamento
2.7	Documentación.
2.8	Evaluación de los SGC.
2.9	Mejora Continua.
2.10	Papel de las técnicas estadísticas.
2.11	SGC y otros Sistemas de Gestión.
2.12	Relación entre los SGC y los Modelos de Excelencia.

La definición de procesos en ISO 900 se da de la siguiente forma:





Los requisitos según ISO 9000 para la calidad de un software se agrupan bajo 20 títulos:

1. Responsabilidad de la gestión.
2. Inspección, medición y equipo de pruebas.
3. Sistema de calidad.
4. Inspección y estado de pruebas.
5. Revisión de contrato.
6. Acción correctiva.
7. Control de diseño.
8. Control de producto no aceptado.
9. Control de documento.
10. Tratamiento, almacenamiento, empaquetamiento y entrega.
11. Compras.
12. Producto proporcionado al comprador.
13. Registros de calidad.
14. Identificación y posibilidad de seguimiento del producto,
15. Auditorías internas de calidad.
16. Formación
17. Control del proceso
18. Servicios.
19. Inspección y estado de prueba.
20. Técnicas estadísticas.



ISO 25000

Es una familia de normas de ISO encargada de evaluar y dar los requerimientos para la calidad de un producto de software.

Es conocida como Modelo SQUARE y se basa en las siguientes normas para determinar la calidad y evaluar requisitos de un software.

ISO/IEC 2500n División de gestión de calidad.

ISO/IEC 2501n División de modelo de calidad.

ISO/IEC 2502n División de mediciones de calidad.

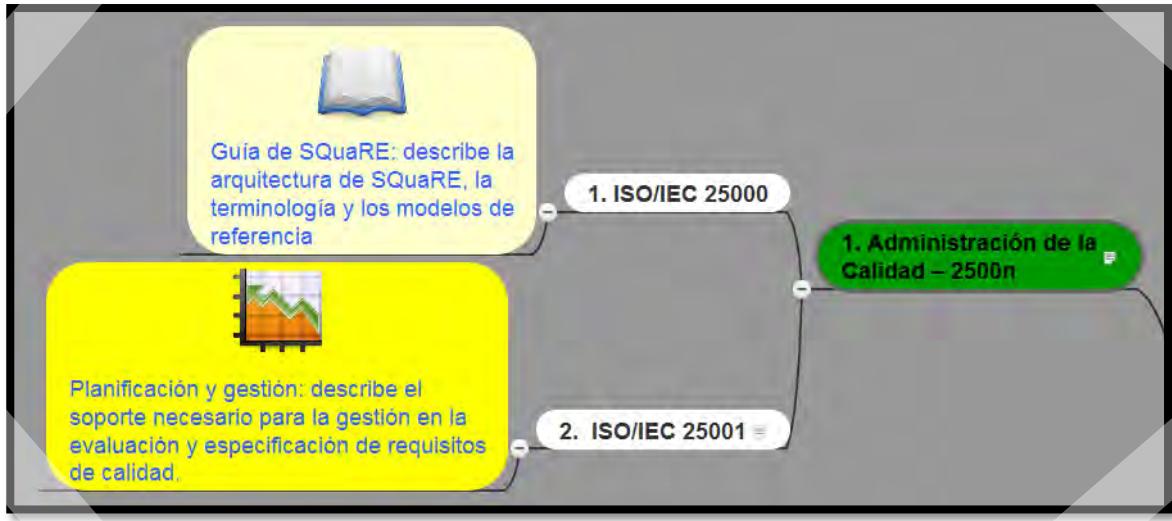
ISO/IEC 2503n División de requisitos de calidad.

ISO/IEC 2504n División de evaluación de calidad.

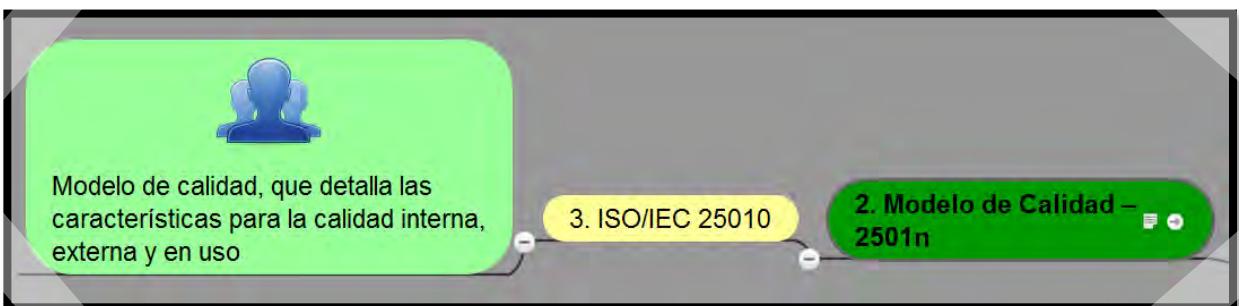
ISO/IEC 25050-25099n Estándares de extensión.



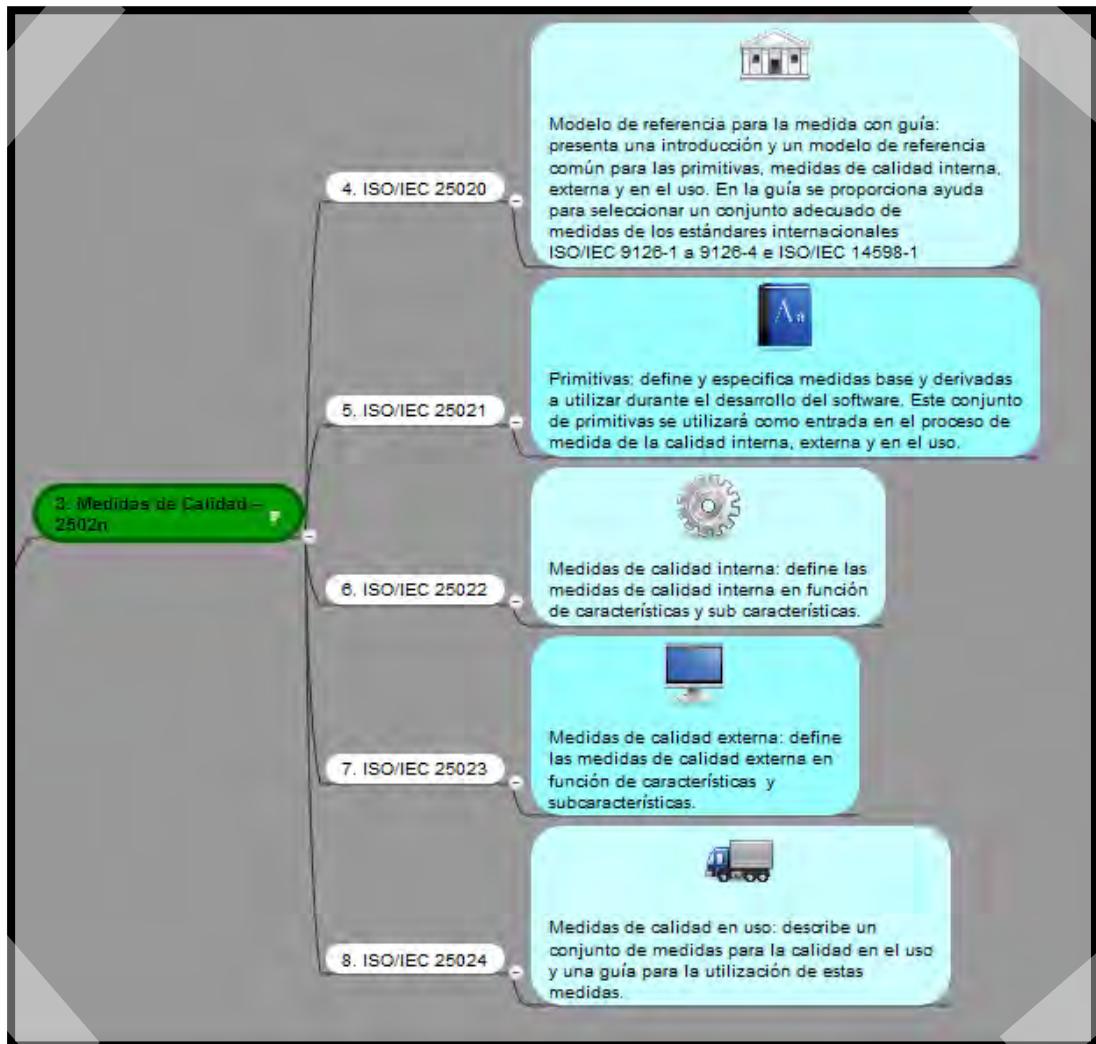
ISO/IEC 2500n División de gestión de calidad.

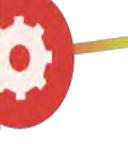


ISO/IEC 2501n División de modelo de calidad.



ISO/IEC 2502n División de medicaciones de calidad.





ISO/IEC 2503n División de requisitos de calidad.

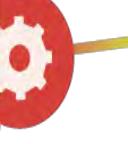


Requisitos de calidad:
el único estándar de
esta serie.

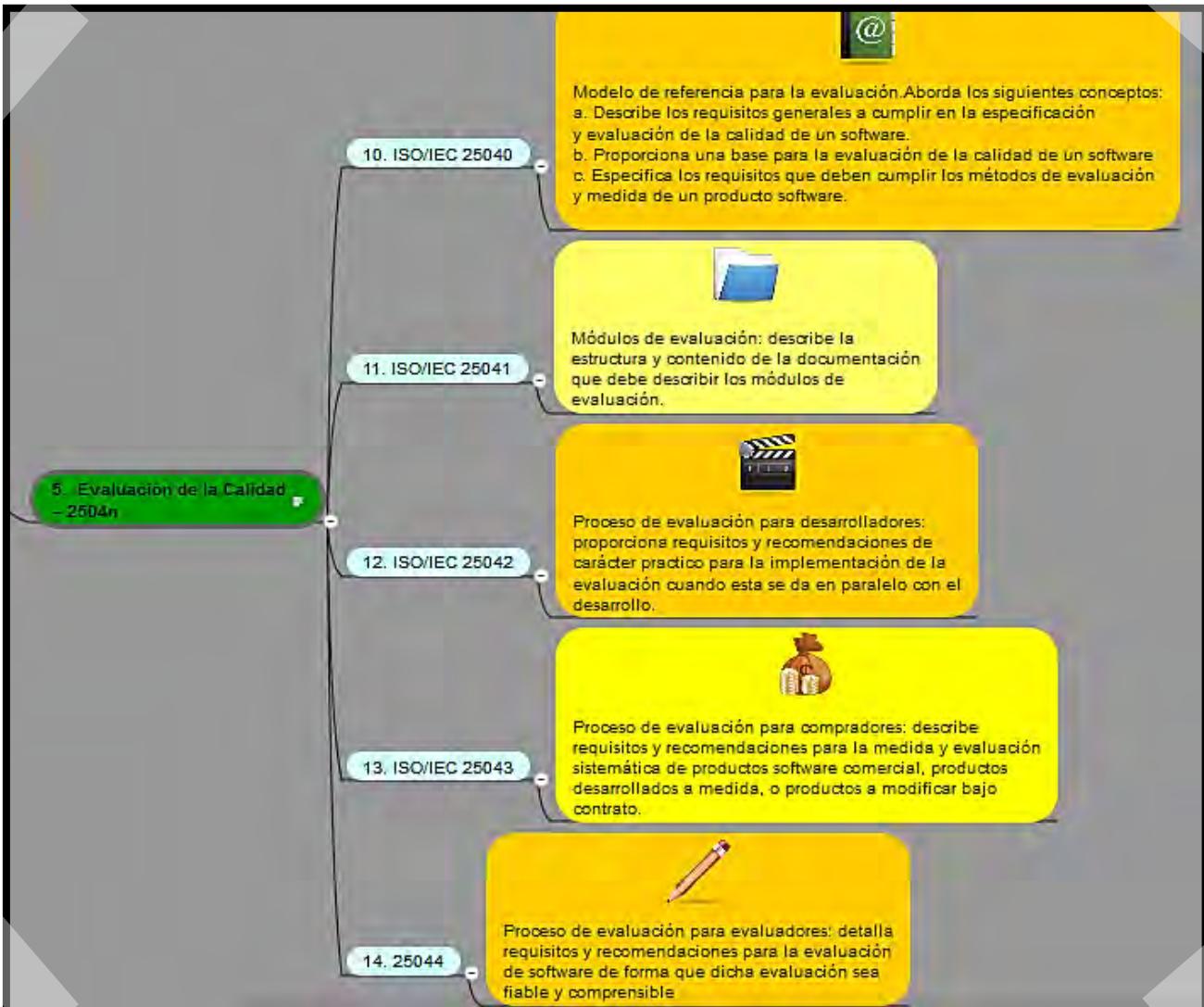
9. ISO/IEC 25030

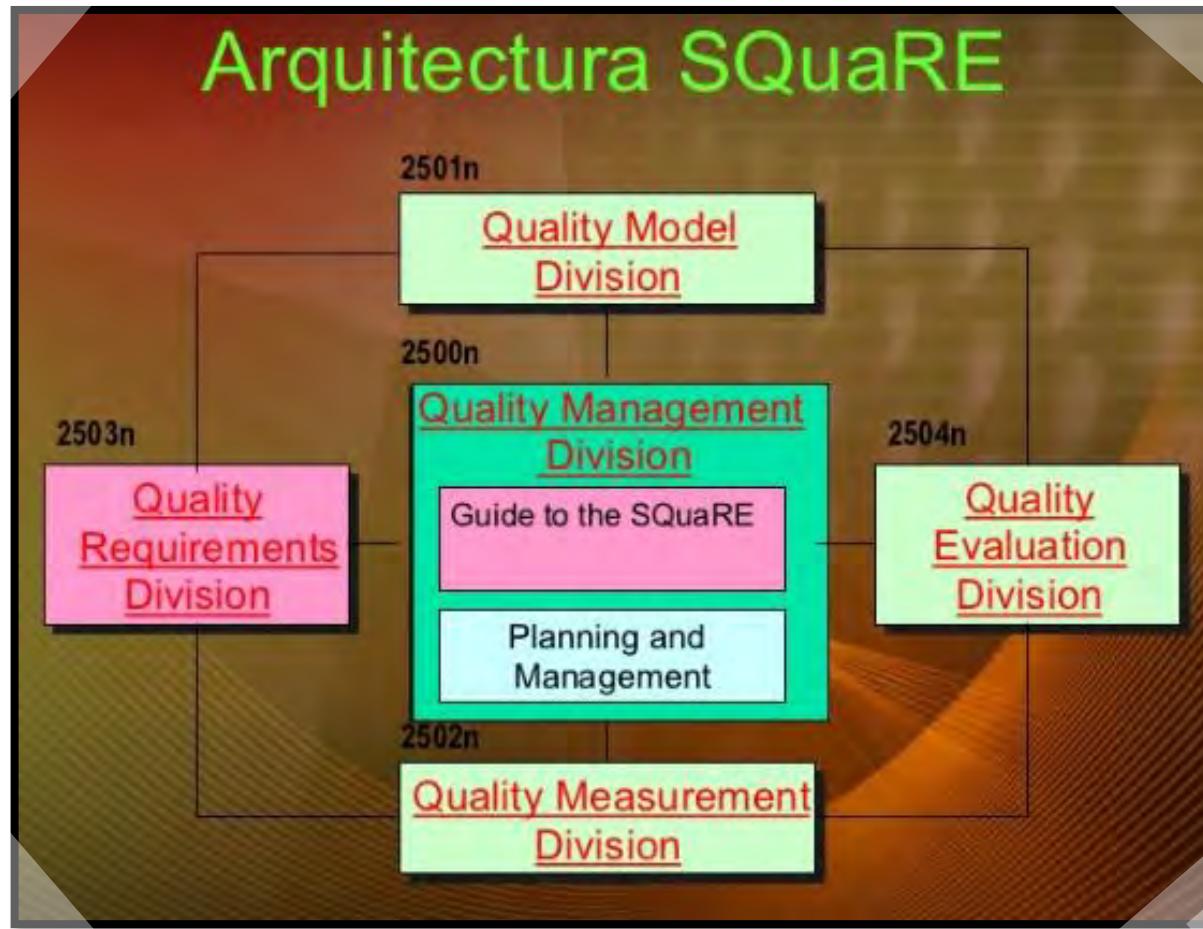
**4. Requerimientos de
Calidad – 2503n**





ISO/IEC 2504n División de evaluación de calidad.





IEE Std. 1061-1998

El IEEE Std. 1061-1998 es un estándar para una metodología de métricas de calidad del software.

Proporciona una metodología para establecer requisitos de calidad e identificar, implementar y validar medidas de calidad del proceso y del producto software.

Esta metodología es aplicable a todo software.

Este estándar cubre conceptos de los siguientes tres grupos:

- Medidas
- Procesos
- Objetivos – metas

La siguiente tabla contiene los principales estándares IEEE que están orientados al aseguramiento de la calidad de un proyecto de software:

IEEE 730-2002	Planes de aseguramiento de la calidad del software
IEEE 829-1998	Documentación de pruebas del software
IEEE 982.1, 982.2	Diccionario estándar de medidas para producir software fiable
IEEE 1008-1987	Pruebas de unidad del software
IEEE 1012-1998	Verificación y validación del software
IEEE 1028-1997	Revisiones del software
IEEE 1044-1993	Clasificación estándar para anomalías del software
IEEE 1061-1998	Estándar para una metodología de métricas de calidad del software
IEEE 1228-1994	Planes de seguridad del software

ISO/IEC 15939 (2002)

El estándar ISO/IEC 15939 identifica las actividades y las tareas necesarias para aplicar con éxito la medición del software dentro de un proyecto o una estructura de medición organizacional.

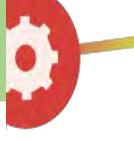
Los dos componentes clave incluidos en este estándar son:

- Proceso de medición del software (software measurement process)
-
- Modelo de información de la medición (measurement information model)

El proceso de medición del software está dirigido por las necesidades de información de la organización, por cada necesidad de información se produce un producto de información que intenta satisfacerlo.

La medición describe cómo los atributos relevantes se cuantifican y se convierten en indicadores que proporcionan una base para la toma de decisión en cuanto a un proyecto de software.

Para medidas del software se apoya básicamente en los conceptos de ISO/IEC 14598 y de ISO/IEC 9126.



Para finalizar y entender como se aplica la métrica ISO/IEC15939 nos podemos guiar en la siguiente tabla:

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



UNIDAD V. MODELOS DE MADUREZ



5.1 Introducción



5.2 Proceso de Software Personal (PSP)



5.3 Proceso de Software de Equipo (TSP)



5.4 Modelo de Capacidad de Madurez (CMM)



5.5 Modelo de Capacidad de Madurez Integrado (CMMI)



5.6 MoProSoft



Introducción

Se abordarán los procesos de gestión más importantes orientados al trabajo de los desarrolladores de software.

Así como los lineamientos que deben seguir para llegar a un correcto término del software con la calidad esperada por el cliente.





Proceso de Software Personal (PSP)

Proceso de Software de Personal (PSP)

El PSP fue definido en 1995 por Watts S. Humphrey del Software Engineering Institute (SEI) en la Carnegie Mellon University.

PSP es una metodología que se concentra en las prácticas de trabajo de los ingenieros del proyecto de manera individual.

La calidad de un software aumenta ya que PSP obliga a que cada uno de los desarrolladores del sistema haga un análisis individual de las tareas y código que le toca realizar en torno a la calidad.

El fin de PSP es incrementar la calidad del trabajo individual de cada ingeniero desarrollador del sistema.



El PSP sigue 3 diferentes principios:

1. Cada ingeniero es diferente; para ser eficiente, debe planificar su trabajo basándose en datos tomados de su trayectoria profesional.
2. Para mejorar su trabajo, los ingenieros deben usar procesos personales bien definidos y cuantificados.
3. El ingeniero debe asumir la responsabilidad personal de la calidad de sus productos como consecuencia de un esfuerzo positivo.

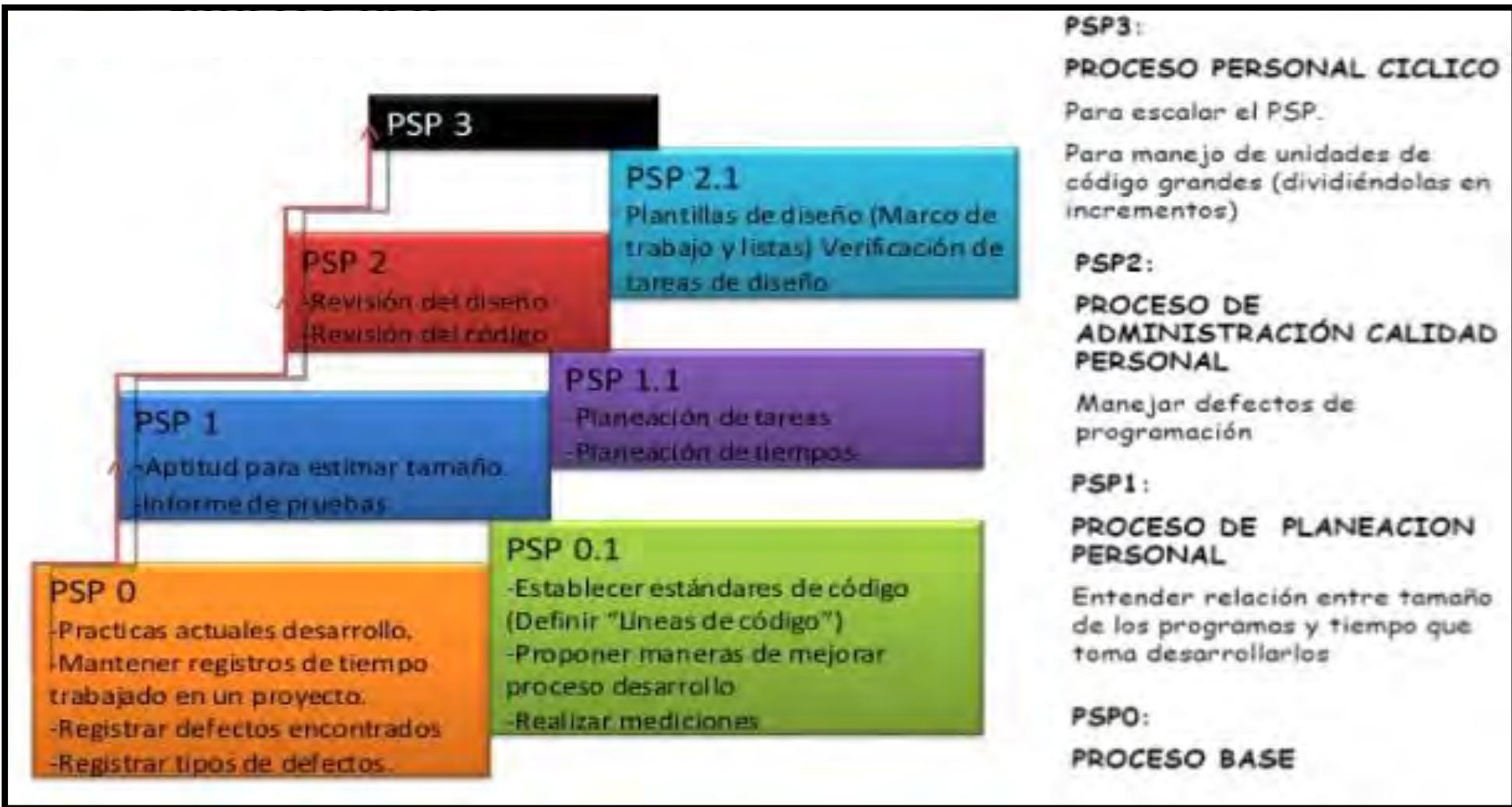
El PSP tiene como objetivos:

- El principal objetivo del PSP es elaborar software de calidad desde el principio ahorrando tiempo, aprox. de un 20% a 40% en comparación con algunos otros métodos de desarrollo de software.
- Reducción de costos.





El PSP se basa en los siguientes niveles de desarrollo:



Para finalizar debemos llevar en tiempo y forma un registro de las actividades del proyecto, necesitamos una hoja de tiempos como la siguiente:

Fecha	Hora		Tiempo de Interrupción	Tiempo	Descripción Actividad	Comentario	C	U
	Inicio	Fin						



Proceso de Software de Equipo (TSP)

Proceso de Software de equipo (TSP)

Es una metodología que permite crear software de calidad cuando se tiene un equipo desarrollador generando un entorno donde el trabajo de equipo sea efectivo, normal y natural.

Se basa en procesos estructurados que indican que debe de hacer el equipo en cada fase de desarrollo del proyecto.

Tiene como objetivos:

Minimizar costos del desarrollo de software.

Incrementar la calidad de los productos del equipo.

Integrar equipos para tener un mayor rendimiento y alcanzar metas definidas.

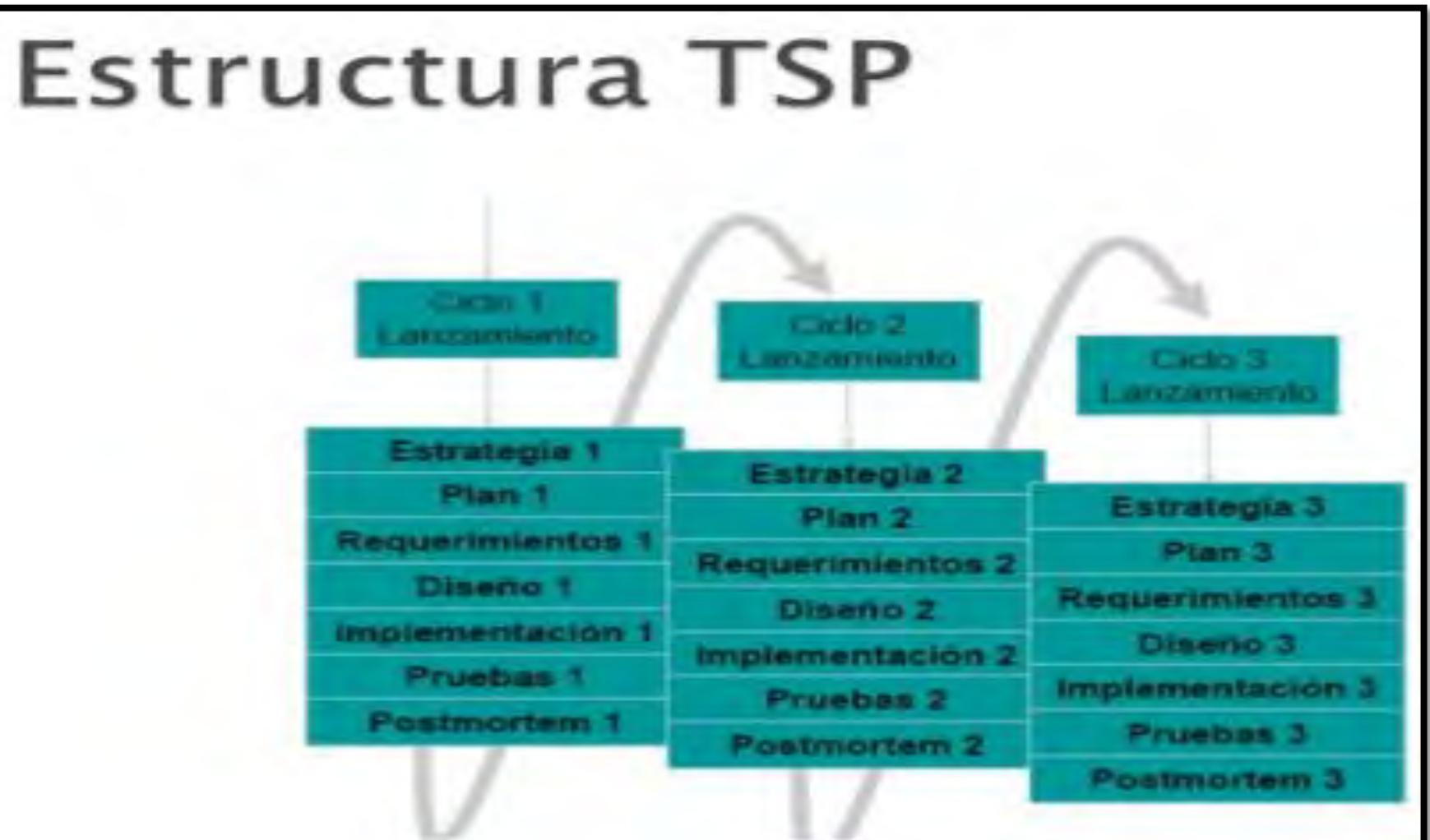
Designar gerentes de equipo e indicarles como dirigir y motivar a su equipo de manera correcta.

El TSP tiene las siguientes características :

- Usa equipos auto-dirigidos junto con un coach (gerente) que ayuda a desarrollar las habilidades de trabajo en equipo.
- Tiene procesos operacionales flexibles que permiten a los equipos adaptar los procesos en pro del desempeño de la calidad.
- Usa planes detallados con actividades no mayores a 10 hrs en periodos de 3-6 meses y establece juntas de cierre llamadas postmortem.



Estructura y fases del TSP



En las etapas se hace lo siguiente:

Lanzamiento: necesidades del cliente, revisión de objetivos y metas, asignación de equipos y roles del personal.

Estrategia: se crea un diseño conceptual del producto, se decide que será producido en cada ciclo, estimación del esfuerzo y tamaño del proyecto.

Plan o planeación: se determinan las tareas a realizar y se realiza una bitácora de las mismas.

Requerimientos: se hace un plan de pruebas que demuestre que el sistema satisface las necesidades del cliente.

Diseño: se elabora un diseño de alto nivel y se desarrolla un plan de pruebas de integración.

Pruebas: se construye y diseña el sistema, se llevan a cabo todos los planes realizados anteriormente para las pruebas y se genera la documentación del sistema.



Postmorten: reunión en donde se analizan los resultados del ciclo.





Modelo de Capacidad de Madurez (CMM)

Modelo de capacidad de madurez (CMM)

Este modelo establece un conjunto de prácticas o procesos clave agrupados en Áreas Clave de Proceso (KPA - Key Process Area).

A su vez estas Áreas de Proceso se agrupan en cinco "niveles de madurez" orientados a organizaciones como forma de normalización en la calidad del software.

Los niveles son:

Nivel 1: Inicial. Las organizaciones no disponen de un ambiente estable para el desarrollo y mantenimiento de software, el resultado de los proyectos es impredecible.

Nivel 2: Repetible. Se repiten éxitos anteriores en proyectos con aplicaciones similares se aplica la disciplina necesaria para el proceso.

Nivel 3: Definido. El proceso del software de las actividades de gestión y de ing. se documenta, se estandariza y se integra dentro de un proceso de software.

Nivel 4: Gestionado. Se recopilan medidas detalladas del proceso del software y de la calidad del producto.

Nivel 5: Optimización. Mediante una retroalimentación cuantitativa del proceso, ideas y tecnologías innovadoras se posibilita una mejora del proceso.

Las KPAs que se deben lograr en cada nivel de madurez de proceso son:

Nivel 2 de Madurez del Proceso

- Gestión de configuración del software
- Garantía de calidad del software
- Gestión de subcontratación del software
- Seguimiento y supervisión del proyecto del software
- Planificación del proyecto del software
- Gestión de requisitos

Nivel 3 de Madurez del Proceso

- Revisiones periódicas
- Coordinación entre grupos
- Ingeniería de productos de software
- Gestión de integración del software
- Programa de formación
- Definición del proceso de la organización
- Enfoque del proceso de la organización

Nivel 4 de Madurez del Proceso

- Gestión de calidad del software
- Gestión cuantitativa del proceso

Nivel 5 de Madurez del Proceso

- Gestión de cambios del proceso
- Gestión de cambios de tecnología



Modelo de Capacidad de Madurez Integrado (CMMI)



Modelo de capacidad de madurez integrado

CMMI es una metodología formada por:

- Systems engineering (SE)
- Software engineering (SW)
- Integrated product and process development (IPPD)
- Supplier sourcing (SS)

Los 6 niveles de madurez definidos en CMMI para medir la capacidad de los procesos son:

- 0.- Incompleto: El proceso no se realiza, o no se consiguen sus objetivos.
- 1.- Ejecutado: El proceso se ejecuta y se logra su objetivo.
- 2.- Gestionado: Además de ejecutarse, el proceso se planifica, se revisa y se evalúa para comprobar que cumple los requisitos.

3.- Definido: Además de ser un proceso gestionado se ajusta a la política de procesos que existe en la organización.

4.- Cuantitativamente gestionado: Además de ser un proceso definido se controla utilizando técnicas cuantitativas.

5.- Optimizante: Además de ser un proceso cuantitativamente gestionado, de forma sistemática se revisa y modifica para adaptarlo a los objetivos del negocio. Mejora continua





MoProsoft



MoProSoft

El Modelo de Procesos de Software fue desarrollado a solicitud de la Secretaría de Economía para servir de base a la Norma Mexicana para la Industria de Desarrollo y Mantenimiento de Software.

Tomó referencias como: ISO9000:2000, Nivel 2 y 3 de CMM® V.1.1, PMBOK, Y SWEBOK.

Moprosot se estructura en 3 categorías:

- Categoría de Alta Dirección (DIR)

Establecen los lineamientos para los procesos de la Gerencia y se retroalimenta con la información generada por ellos en apoyo a la estrategia de la organización.

- Categoría de Gerencia (GER)

Definen los elementos para el funcionamiento de los procesos de la Operación en función de la estrategia de Dirección y comunica los resultados a la Alta Dirección.

- Categoría de Operación (OPE)

Se realizan las actividades de acuerdo a los elementos proporcionados por la Gerencia y entrega a ésta la información y productos generados.

Cada categoría se compone de lo siguiente:

- Categoría Dirección.

- ✓ Gestión de Negocios

- Categoría Gerencia

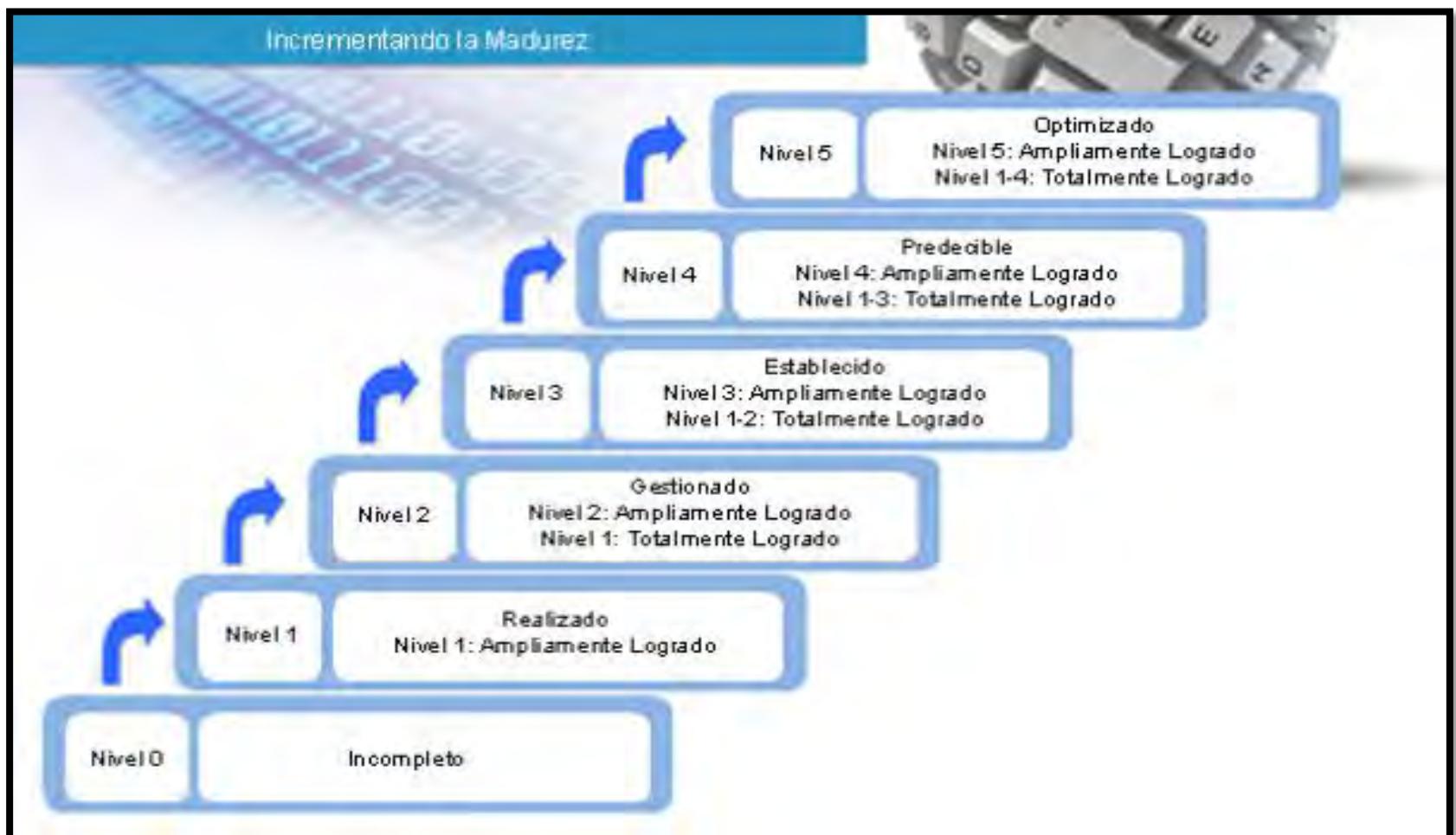
- ✓ Gestión de Proyectos
 - ✓ Gestión de Recursos
 - ✓ Recursos Humanos y Ambiente de Trabajo
 - ✓ Bienes, Servicios e Infraestructura
 - ✓ Conocimiento de la Organización

- Categoría Operación

- ✓ Administración Específica de Proyectos
 - ✓ Desarrollo y Mantenimiento de Software



Niveles de MoProSoft



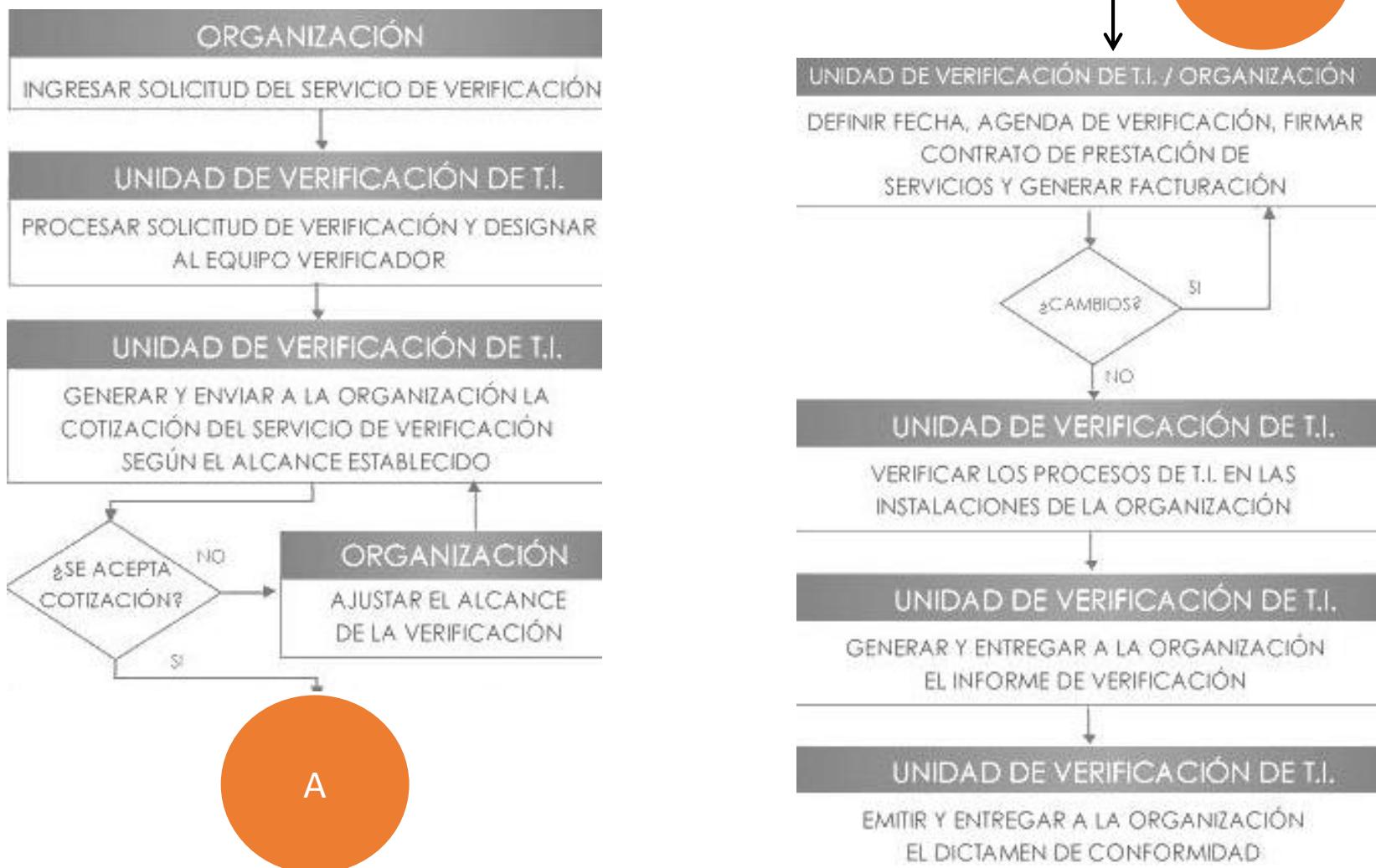


Niveles de MoProSoft

Nivel	Nivel de Capacidad	Descripción	Color
1	Realizado	El proceso se implementa y alcanza su propósito	Amarillo
2	Gestionado	El proceso realizado se administra. Sus productos de trabajo están establecidos, controlados y mantenidos	Azul
3	Establecido	El proceso realizado y gestionado se implementa por medio de un proceso definido	Verde
4	Predecible	El proceso establecido opera bajo límites definidos y conocidos	Rosa
5	Optimizado	El proceso predecible se mejora continuamente	N.A.



Como adquirir la norma MoProSoft





Como ultimo punto de la unidad se muestran los precios de una normalización MoProSoft para una organización.

Nivel Objetivo	Precios sin I.V.A.
Nivel 1	\$ 23, 500.00
Nivel 2	\$ 33, 500.00
Nivel 3	Solicitar Cotización
Nivel 4	Solicitar Cotización
Nivel 5	Solicitar Cotización



UNIDAD VI. TEMAS SELECTOS



6.1 Herramientas CASE



6.2 Ingeniería WEB



6.3 Reingeniería

Herramientas CASE

HERRAMIENTAS CASE

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software. Como es sabido, los estados en el Ciclo de Vida de desarrollo de un Software son: Investigación Preliminar, Análisis, Diseño, Implementación e Instalación.

CASE se define también como:

Conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

La sigla genérica para una serie de programas y una filosofía de desarrollo de software que ayuda a automatizar el ciclo de vida de desarrollo de los sistemas.

Una innovación en la organización, un concepto avanzado en la evolución de tecnología con un potencial efecto profundo en la organización. Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

Estas herramientas pueden proveer muchos beneficios en todas las etapas del proceso de desarrollo de software, algunas de ellas son:

- ◆ Verificar el uso de todos los elementos en el sistema diseñado.
- ◆ Automatizar el dibujo de diagramas.
- ◆ Ayudar en la documentación del sistema.
- ◆ Ayudar en la creación de relaciones en la Base de Datos.
- ◆ Generar estructuras de código.

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo, además de la propia herramienta.

Algunos de los diagramas y modelos utilizados con mayor frecuencia son:

- Diagrama de casos de uso
- Diagrama de clase
- Diagrama de secuencia
- Diagrama de colaboración.
- Diagrama de estados
- Diagrama de actividad.
- Diagrama de componentes
- Diagrama de despliegue.
- Diagrama de composición estructural

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo, además de la propia herramienta.

Algunos de los diagramas y modelos utilizados con mayor frecuencia son:

- Diagrama de casos de uso
- Diagrama de clase
- Diagrama de secuencia
- Diagrama de colaboración.
- Diagrama de estados
- Diagrama de actividad.
- Diagrama de componentes
- Diagrama de despliegue.
- Diagrama de composición estructural



Fuente: www.traidnt.net



Fuente: modeling-languages.com



Fuente: [en.Wikipedia.org](https://en.wikipedia.org)



Fuente: curso_sin2.blogia.com

Ingeniería WEB

Patrones de diseño

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y describe también el núcleo de la solución al problema, de forma que puede utilizarse un millón de veces sin tener que hacer dos veces lo mismo.(Alexander(arquitecto/urbanista))

Un patrón de diseño es:

- una solución estándar para un problema común de programación
- una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios
- un proyecto o estructura de implementación que logra una finalidad determinada
- un lenguaje de programación de alto nivel
- una manera más práctica de describir ciertos aspectos de la organización de un programa conexiones entre componentes de programas la forma de un diagrama de objeto o de un modelo de objeto.

Ingeniería web

La ingeniería web estudia el desarrollo y la creación de WebApps o en su traducción al español aplicaciones web.

Este tipo de aplicaciones tienen atributos diferentes al software general a considerar como:

Intensivas de Red.

Reside en una red y debe dar servicio a las necesidades de una comunidad diversa de clientes.

Controlada por el contenido.

En muchos casos, la función primaria de una WebApp es utilizar hipermedia para presentar al usuario el contenido de textos, gráficos, sonido y vídeo.

Evolución continua.

Las aplicaciones Web están en constante evolución. No es inusual que algunas WebApps (específicamente, su contenido) se actualicen cada hora.

Inmediatez

El tiempo que se tarda en comercializar un sitio Web completo puede ser cuestión de días o semanas.

Seguridad

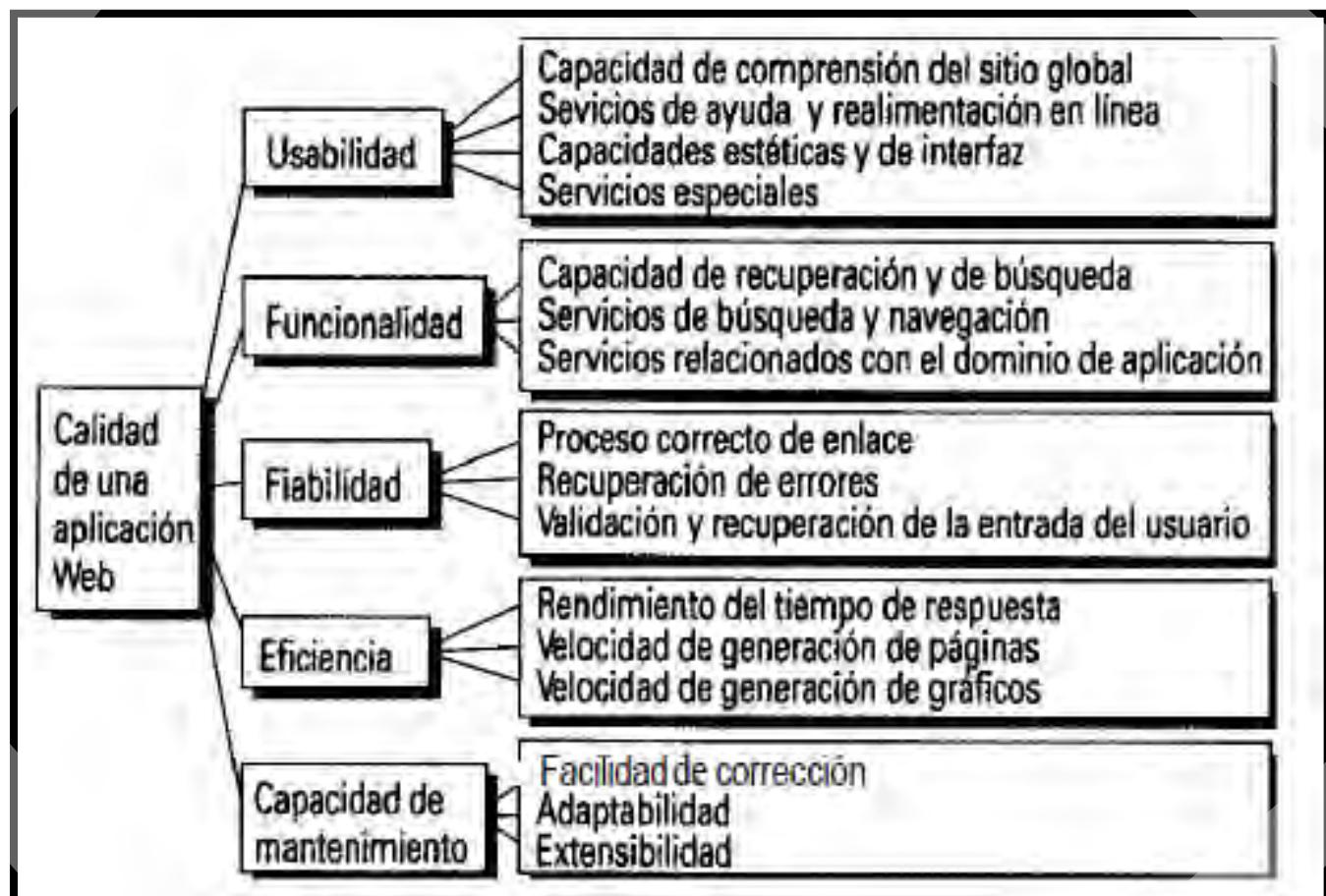
Las WebApps están disponibles a través de acceso por red, es imposible limitar la población de usuarios que pueden entrar a la aplicación. Con objeto de proteger el contenido confidencial, deberán implementarse fuertes medidas de seguridad en toda la infraestructura de la aplicación.

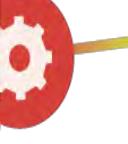
Estética

Una parte innegable del atractivo de una WebApp es su apariencia e interacción. Cuando se ha diseñado una aplicación con el fin de comercializarse o vender productos o ideas, la estética puede tener mucho que ver con el éxito del diseño técnico.

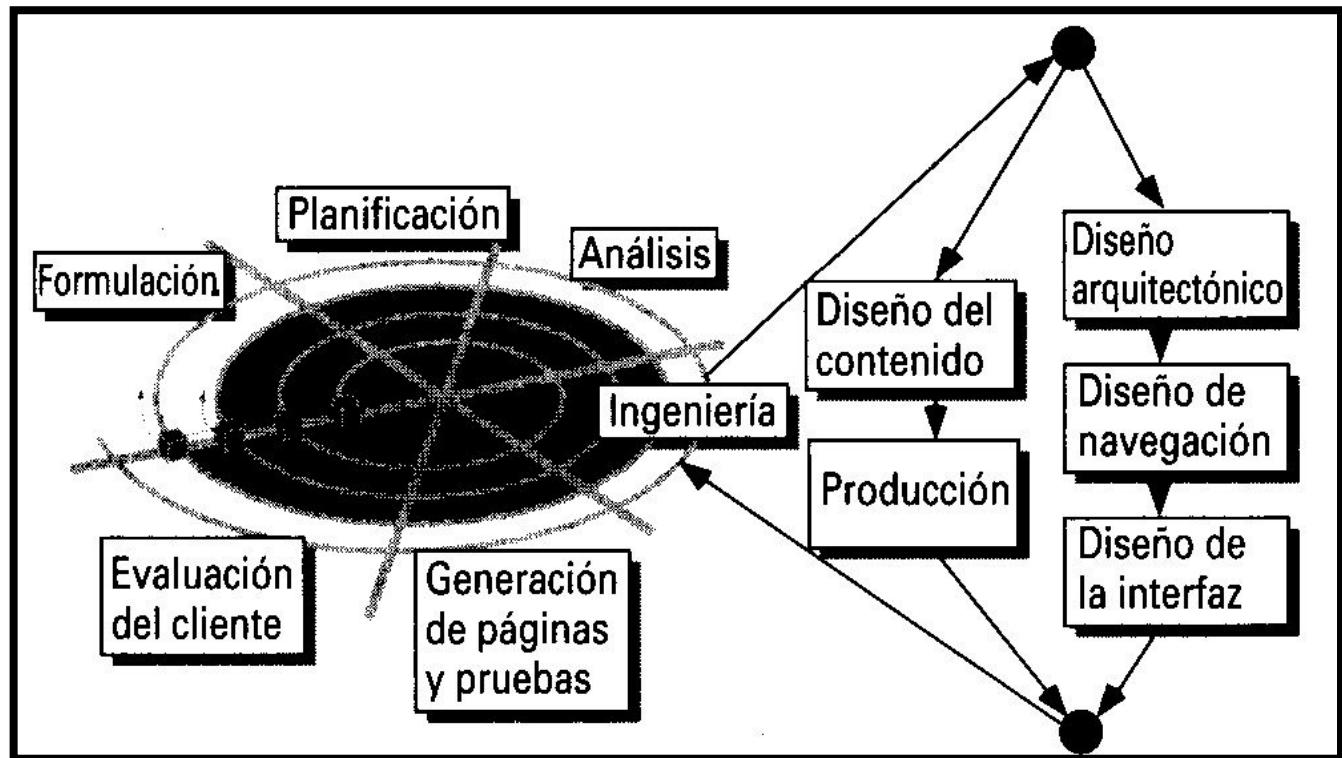


Árbol de calidad para una aplicación web.





Para la elaboración de aplicaciones Web se puede manejar el modelo IWeb el cual nos permite identificar los aspectos importantes a considerar de una aplicación web.



Como se aprecia en el esquema podemos completar la etapa de ingeniería elaborando de manera paralela los dos siguientes caminos:

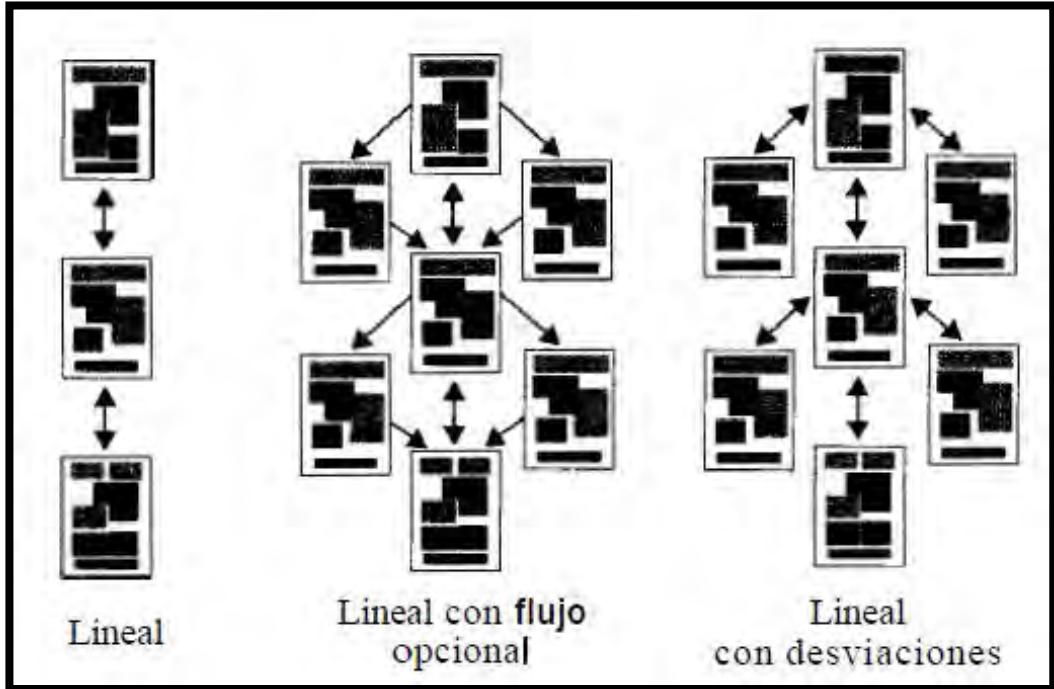
- Etapa de diseño del contenido seguida de la producción.
- Etapa de diseño arquitectónico, diseño de navegación y diseño de la interfaz.

Para la elaboración de una aplicación web se pueden utilizar plantillas las cuales proporcionan un marco de trabajo esquemático y estructura para el desarrollo web.

Las siguientes son las estructuras o plantillas que puede seguir una WepApp:



Estructura lineal



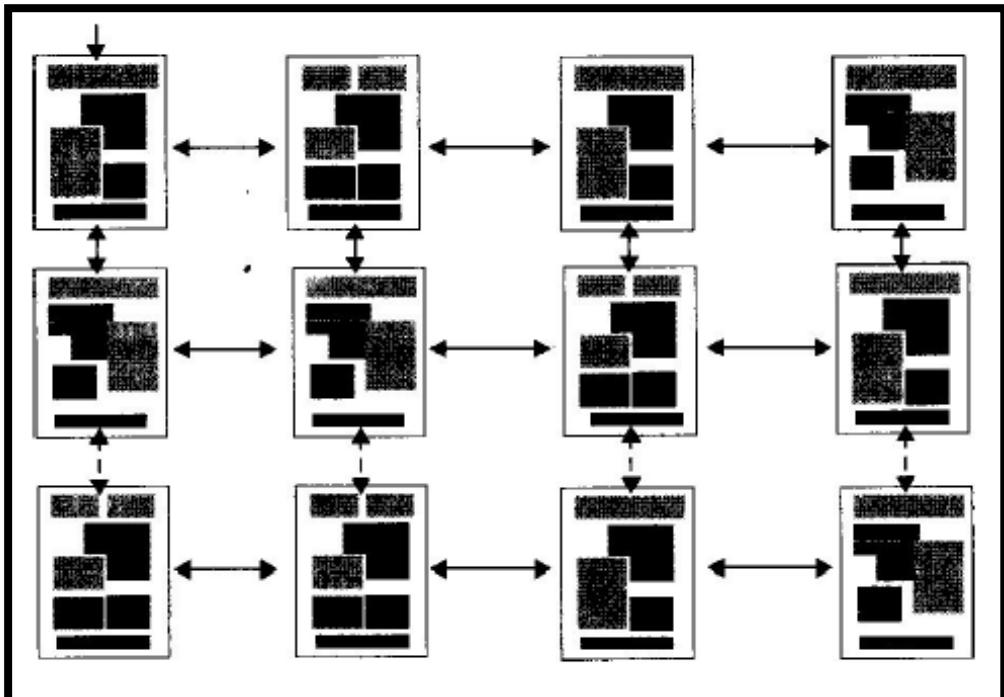


Estructura reticular: contenido es organizado categóricamente dos dimensiones (o más).

Por ejemplo, consideremos un sitio de comercio electrónico que vende palos de golf, la dimensión horizontal representa el tipo de palo en venta (por ejemplo, maderas, hierros, wedges, putters), la dimensión vertical representa la oferta proporcionada por los fabricantes.

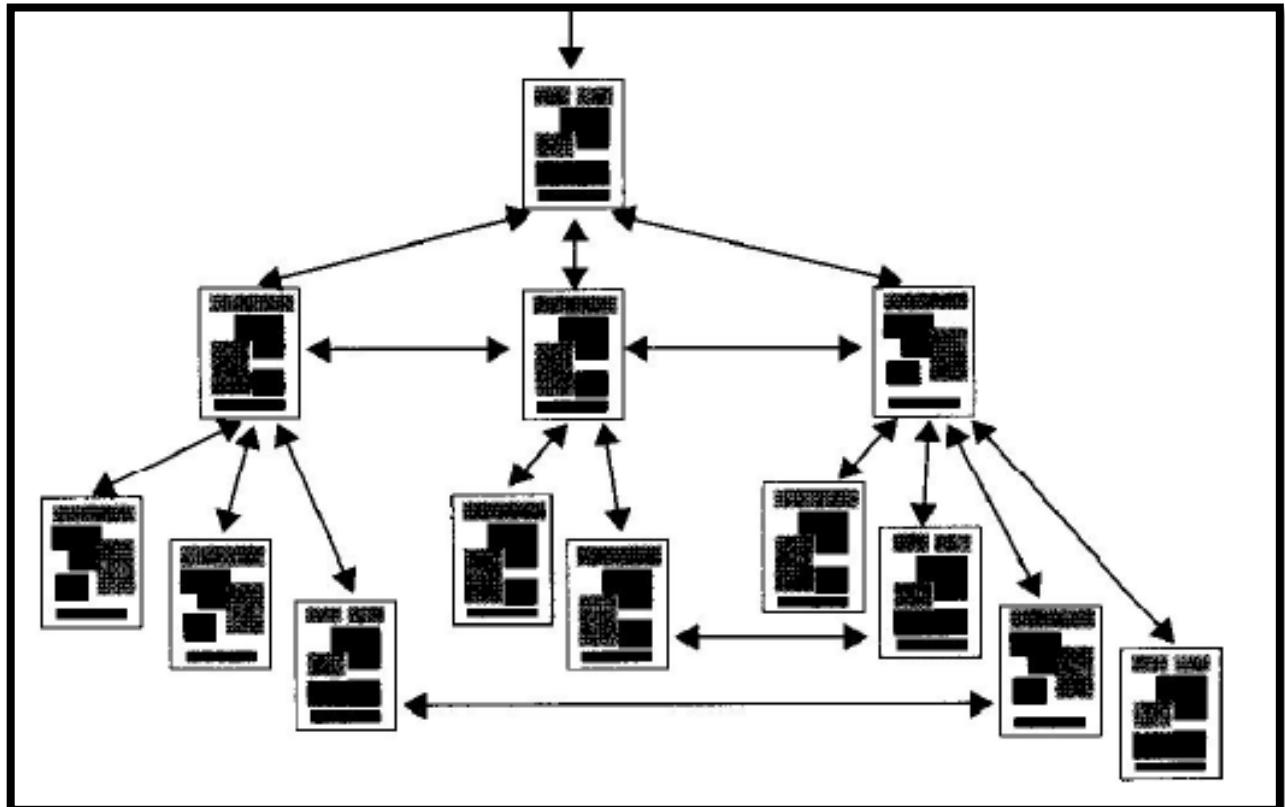
Así un usuario pueda navegar horizontalmente para encontrar los putters, y recorrer la columna para examinar las ofertas.

Estructura reticular:



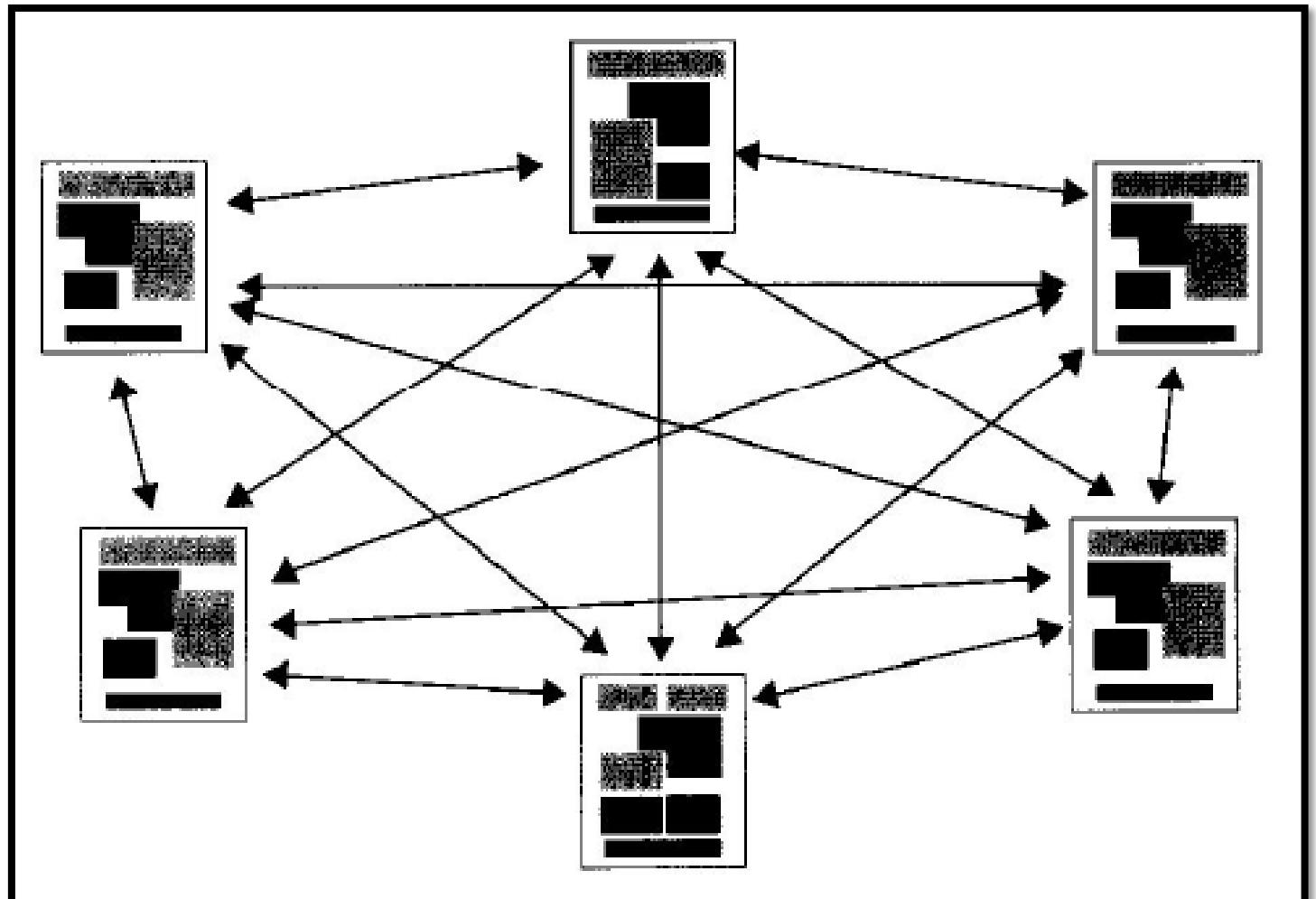


Estructuras jerárquicas: estructuras donde el contenido es desglosado en forma de árbol.





Estructura en red: estructura que se basa en hipertexto para enlazar los diferentes contenidos de la aplicación.



Además de los aspectos anteriores, para el diseño de la interfaz de una WebApp se debe de tomar en cuenta los siguientes aspectos:

- La lectura en una aplicación web es más lenta que si se lee un texto aproximadamente en un 25%, por tanto no hay que obligar al usuario a leer grandes cantidades de texto.
- La estética nunca deberá sustituir la funcionalidad.
- El diseño no deberá depender de las funciones del navegador.
- Las opciones de navegación deberán ser obvias, incluso para el usuario casual.

Las pruebas realizadas a las WebApps para la verificación de errores y fallas son:

1. El modelo de contenido de la WebApp es revisado para descubrir errores.
2. El modelo de diseño para la WebApp es revisado para descubrir errores de navegación.
3. Se aplican pruebas de unidad a los componentes de proceso seleccionados y las páginas Web (pruebas a formularios, guiones, applets).
4. Se construye la arquitectura, se realizan las pruebas de integración.
5. La WebApp ensamblada se prueba para conseguir una funcionalidad global y un contenido.

Reingeniería

Reingeniería

La reingeniería se explicara comenzando por una visión general de la reingeniería de procesos de negocio pasando entonces a una discusión más detallada de las actividades técnicas que se producen cuando se efectúa una reingeniería del software.

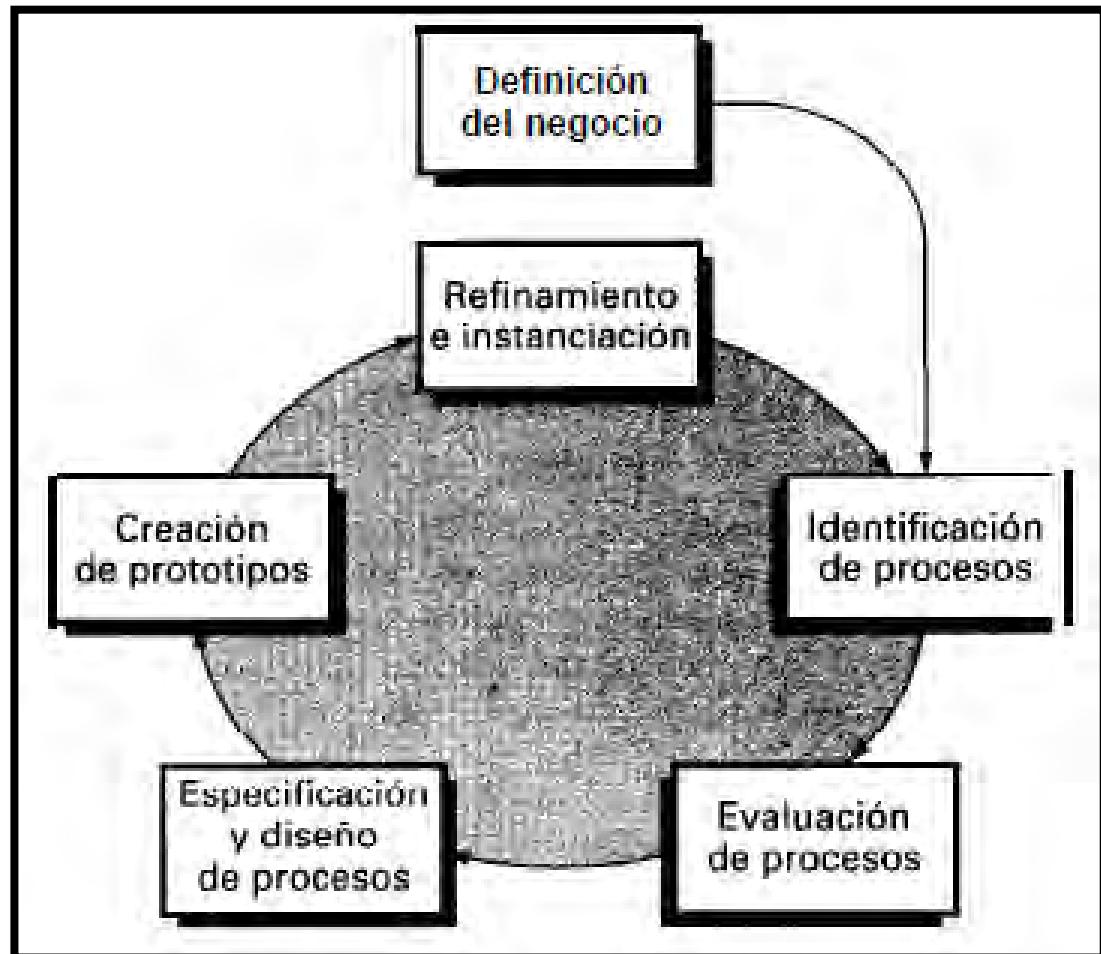
Reingeniería de procesos.

La Reingeniería de procesos es la búsqueda e implementación de cambios radicales en el proceso de negocios para lograr un avance significativo.

Un *proceso de negocio* es un conjunto de tareas lógicamente relacionadas que se llevan a cabo para obtener un determinado resultado de negocio, combinando las personas, los equipos, los recursos materiales y los procedimientos de negocio.

Para llevar a cabo una reingeniería de procesos podemos seguir el modelo RPN el cual se basa en el siguiente diagrama.

Modelo RPN.





Definición del negocio

Los objetivos de negocios se identifican en un contexto de cuatro controladores principales: reducción de costes, reducción de tiempos, mejora de calidad y desarrollo y potenciación del personal.



Identificación de procesos

En esta actividad se identifican los procesos críticos para alcanzar los objetivos definidos para el negocio, se crean prioridades en función de su importancia, necesidad de cambio, o cualquier otra forma que resulte adecuada para la actividad de reingeniería.



Evaluación de procesos

Los procesos existentes deberán analizarse y medirse exhaustivamente., se identifican los costes y los tiempos consumidos por las tareas de proceso, se anotarán cuidadosamente y se aislarán los problemas de calidad y rendimiento.



Especificación y diseño de procesos

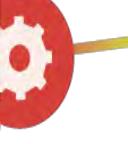
Se prepararán casos prácticos para cada uno de los procesos que se tengan que rediseñar, estos identifican un escenario que proporciona resultados a un cliente diseñándose un nuevo conjunto de tareas.

Creación de prototipos

Es preciso construir un prototipo del proceso de negocios rediseñado antes de integrarlo por completo en el negocio.

Refinamiento e instancia

Basándose en la realimentación procedente del prototipo, se refina el proceso de negocio y después se instancia en el sistema del negocio.



Reingeniería del software, Reestructuración, Ingeniería inversa y directa.

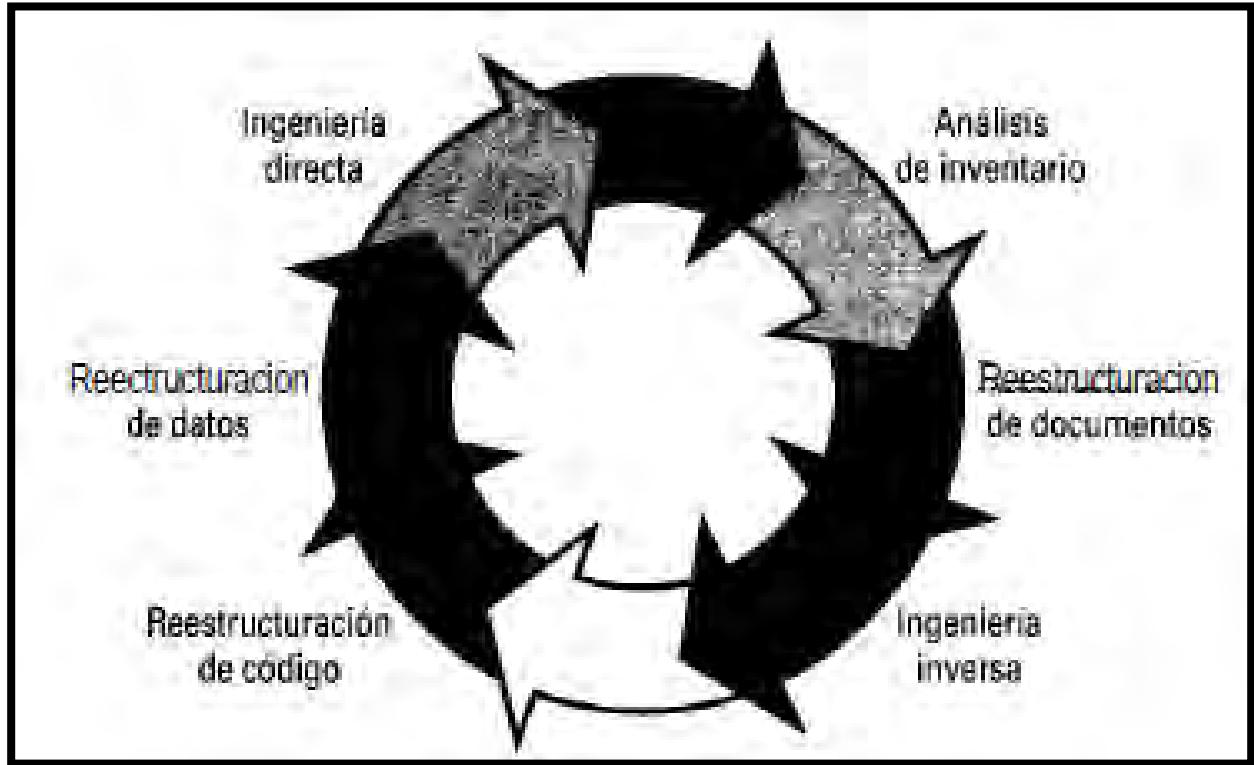
La reingeniería de sistemas de información es una actividad que absorberá recursos de las tecnologías de la información en una compañía durante muchos años.

Esta es la razón por la cual toda organización necesita una estrategia para la reingeniería del software.

Para esto tenemos un modelo de procesos de reingeniería del software el cual se basa en el siguiente diagrama:



Modelo de procesos de reingeniería.



Modelo de procesos de reingeniería

Análisis de inventario

Todas las organizaciones de software deberán disponer de un inventario de todas sus aplicaciones. El inventario puede que no sea más que una hoja de cálculo con la información que proporciona una descripción detallada (por ejemplo: tamaño, edad, importancia para el negocio) de todas las aplicaciones activas.

Es importante destacar que el inventario deberá revisarse con regularidad.

Reestructuración de documentos.

Una documentación escasa es la marca de muchos sistemas heredados.

¿Qué se puede hacer al respecto?

Opción 1: La creación de documentación consume muchísimo tiempo. No es óptimo volver a crear la documentación de un programa si este es relativamente estático, está llegando al final de vida útil, y no es probable que experimente muchos cambios.

Opción 2: Es preciso actualizar la documentación, pero se dispone de recursos limitados, quizá no se necesario volver a documentar por completo la aplicación. Más bien se documentarán por completo aquellas partes del sistema que estén experimentando cambios en ese momento.

Opción 3: El sistema es fundamental para el negocio, y es preciso volver a documentarlo por completo.

Ingeniería inversa.

La ingeniería inversa del software es el proceso de análisis de un programa con el fin de crear una representación de programa con un nivel de abstracción más elevado que el código fuente.

La ingeniería inversa es un proceso de recuperación de diseño. Con las herramientas de la ingeniería inversa se extraerá del programa existente información del diseño arquitectónico y de proceso, e información de los datos.

Reestructuración del código

Algunos sistemas tienen una arquitectura de programa relativamente sólida, pero los módulos han sido codificados de forma que es difícil comprenderlos, comprobarlos y mantenerlos. En estos casos, se reestructura el código analizándolo mediante herramientas se indican las violaciones de las estructuras de programación.

Reestructuración de datos

Un programa que posea una estructura de datos débil será difícil de adaptar y de mejorar, se analiza minuciosamente la arquitectura del programa y se definen los modelos de datos necesarios. Se identifican objetos de datos, atributos y se revisan las estructuras de datos a efectos de calidad.

Ingeniería directa (*forward engineering*).

La ingeniería directa, que se denomina como renovación o reclamación, no solamente recupera la información de diseño de un software ya existente, sino que, además, utiliza esta información para alterar o reconstruir el sistema existente en un esfuerzo por mejorar su calidad global.

En la mayoría de los casos, el software procedente de una reingeniería vuelve a implementar la funcionalidad del sistema existente, y añade además nuevas funciones y/o mejora el rendimiento global.

Referencias

- Pressman, Roger. Adaptado por Ince, Darrel . Ingeniería de Software un enfoque práctico. McGrawHill. España. Quinta edición.
- Philippe Kruchten. Planos Arquitectónicos: El Modelo de “4+1” Vistas de la arquitectura del Software. Recuperado el 13 de diciembre del 2015 de: http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:modelo4_1.pdf
- Somerville, Ian. Ingeniería de Software. Pearson. España: Séptima edición 2005.
- **Ingeniería de Software “Un enfoque práctico” – Roger Pressman, 5ta edición, Ed. McGrawHill**
- <http://www.tamps.cinvestav.mx/~ertello/swe/sesion01.pdf>
- <http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema03.pdf>
- <http://softwarearequipa.blogspot.mx/2010/05/importancia-del-software-en-las.html>
- <http://redalyc.uaemex.mx/redalyc/pdf/1942/194214476008.pdf>
- <http://www.slideshare.net/toryneutral/metodos-formales>
- Ingeniería de Software “Un enfoque práctico” – Roger Pressman, 5ta edición, Ed. McGrawHill
- <http://www.alumnos.unican.es/~uc17923/Tema3.pdf>
- <http://www.slideshare.net/vichodmx/cronograma-de-actividades-2555329>
 - <http://www.habilidadesdegestion.com/Gestiondeproyectos/fasesdelagestiondeunproyecto/seguimiento y control.htm>
 - http://es.wikipedia.org/wiki/T%C3%A9cnica_de_revisi%C3%B3n_y_evaluaci%C3%B3n_de_programas

- Calidad de software. Systemata. Recuperado el 12 de diciembre del 2015 de: http://www.systemata.cl/SYSTEMATA/images/calidad_del_software.png
- Philippe Kruchten. Planos Arquitectónicos: El Modelo de “4+1” Vistas de la arquitectura del Software. Recuperado el 13 de diciembre del 2015 de: http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:modelo4_1.pdf
- Ernesto Díaz. Sistemas de información UCV, Ciencias I-2012. Publicado el 16 abril del 2012. Recuperado el 13 de diciembre del 2015 de: http://1.bp.blogspot.com/-QUKrC1Q_5jk/T5SnAtCeifI/AAAAAAAEEjM/dluKTIBOYo8/s1600/bpmn.png
- Aduanas.gub.uy .Diagrama de actividades. Recuperado el 13 de diciembre del 2015 de: servicios.aduanas.gub.uy
- Microsoft Development Network. Diagrama de secuencias. Recuperado el 13 de diciembre del 2015 de: <https://i-msdn.sec.s-msft.com/dynimg/IC378044.jpeg>
- Gabriel Ortiz. Modelado de datos de orientado a objetos. Recuperado el 13 de diciembre del 2015 de: http://www.gabrielortiz.com/imagenes/012_7.gif
 - Diagrama de paquetes. Feedsreader-intranet. Proyecto creado para la asignatura ISG3 de la Escuela Técnica de Ingeniería Informática de la Universidad de Sevilla. Recuperado el 13 de diciembre del 2015 de: <https://code.google.com/p/feedsreader-intranet/downloads/detail?name=DiagramaPaquetes%20GRUPO9.jpg>

- Imagen del título. Recuperado el 12 de diciembre del 2015 de: http://3.bp.blogspot.com/-MsqiyZAnf1M/VfNJpXXnNSI/AAAAAAAABAw/C8vSrpdhAXU/s1600/thumnail_tic.jpg
- Imagen barra derecha. Recuperado el 12 de diciembre del 2015 de: <http://dhd.mx/desarrollo-de-software-a-la-medida.html>
- Imagen esquina inferior izquierda. Recuperado el 12 de diciembre del 2015 de: <http://binarykode.com/>
- UNIDAD I. INTRODUCCIÓN AL ENFOQUE ORIENTADO A OBJETOS. Recuperado el 12 de diciembre del 2015 de: <http://www.codigogeek.com/wp-content/uploads/2015/10/desarrollo-software.jpg>
- UNIDAD II. LA CALIDAD Y LAS MÉTRICAS. Recuperado el 12 de diciembre del 2015 de: www.qbo-track.com
- UNIDAD III. ANÁLISIS. Recuperado el 12 de diciembre del 2015 de: teoriaredactada.blogspot.com
- UNIDAD IV. DISEÑO DE OBJETOS. Recuperado el 12 de diciembre del 2015 de: <http://www.v-mann.com/files/files/imagenes/desarrollo.jpg>
- UNIDAD V. DISEÑO Y DESARROLLO DE SISTEMAS. Recuperado el 12 de diciembre del 2015 de: <http://www.camara-alemana.org.pe/downloads/dise%C3%B1oweb.png>
- UNIDAD VI. TÓPICOS SELECTOS. Recuperado el 12 de diciembre del 2015 de: <http://www.arqphys.com/construccion/fotos/construccion/Patron-de-dise%C3%B1o.jpg>