INTRODUCCION

Software: los documentos asociados y la configuración de datos que se necesitan para hacer que los programas de computadora operen de manera correcta.

La ingeniería de software: aplica teorías, métodos y herramientas para el desarrollo de software profesional. Es una disciplina de la ingeniería (El uso de las teorías y los métodos adecuados para resolver los problemas teniendo en cuenta las limitaciones financieras y de organización) que se ocupa de todos los aspectos de la producción de software (No sólo el proceso técnico de desarrollo. También la gestión de proyectos y el desarrollo de herramientas, métodos, etc., para apoyar la producción de software.) desde las etapas iniciales de la especificación del sistema hasta el mantenimiento del sistema después de que haya entrado en uso.

Costos: El costo del software en una PC suele ser mayores que el costo del hardware. El mantenimiento del software cuesta más que el costo del desarrollo del mismo. La ingeniería de software tiene que ver con el desarrollo de software rentable.

Productos de software

Productos genéricos: Sistemas independientes que se comercializan y venden a cualquier cliente que desee comprar. Ejemplos - Software para PC tales como programas de gráficos, herramientas de gestión de proyectos; Software CAD; software para mercados específicos, tales como los sistemas de citas para los dentistas. La especificación de lo que el software debe hacer es propiedad del desarrollador del software y las decisiones sobre los cambios en el software son hechas por el desarrollador.

Productos personalizados: Software que esté encargado por un cliente específico para satisfacer sus propias necesidades. Ejemplos - incorporado sistemas de control, software de control del tráfico aéreo, sistemas de monitorización de tráfico. La especificación de lo que el software debe hacer es propiedad del cliente del software y él es el que toma decisiones sobre los cambios de software necesarios.

Software de calidad: tiene que ver con el comportamiento del software mientras se ejecuta, y la estructura y organización de los programas del sistema y la documentación asociada (ej: tiempo de respuesta del software ante la duda de un usuario y la comprensibilidad del código del programa). El conjunto específico de atributos que se espera de un sistema de software depende evidentemente de su aplicación.

Atributos esenciales de un buen software

Mantenimiento: El software debe escribirse de tal forma que pueda evolucionar para satisfacer las necesidades cambiantes de los clientes. Éste es un atributo crítico porque el cambio del software es un requerimiento inevitable de un entorno empresarial variable.
Confiabilidad y seguridad: La confiabilidad del software incluye una variedad de características incluyendo confiabilidad, protección y seguridad. El software fiable no debe causar daño físico o económico en caso de fallo del sistema. Los usuarios malintencionados no deben poder acceder o dañar el sistema.

Eficiencia: El software no debe desperdiciar el uso de los recursos del sistema, como los ciclos de memoria y procesador. Por lo tanto, la eficiencia incluye la capacidad de respuesta, el tiempo de procesamiento, la utilización de la memoria, etc.
Aceptabilidad: El software debe ser aceptable para el tipo de usuario para el que está diseñado. Esto significa que debe ser comprensible, utilizable y compatible con otros sistemas que utilizan.

PREGUNTAS

¿Qué es software? Programas de cómputo y documentación asociada. Los productos de software se desarrollan para un cliente en particular o para un mercado en general.

¿Cuáles son los atributos del buen software? El buen software debe entregar al usuario la funcionalidad y el desempeño requeridos, y debe ser sustentable, confiable y utilizable.

¿**Qué es ingeniería de software?** La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software.

¿**Cuáles son las actividades fundamentales de la ingeniería de software**? Especificación, desarrollo, validación y evolución del software.

¿**Cuál es la diferencia entre ingeniería de software y ciencias de la computación**? Las ciencias de la computación se enfocan en teoría y fundamentos; mientras la ingeniería de software se enfoca en el sentido práctico del desarrollo y en la distribución de software.

¿**Cuál es la diferencia entre ingeniería de software e ingeniería de sistemas**? La ingeniería de sistemas se interesa por todos los aspectos del desarrollo de sistemas basados en computadoras, incluidos hardware, software e ingeniería de procesos. La ingeniería de software es parte de este proceso más general.

¿**Cuáles son los principales retos que enfrenta la ingeniería de software**? Se enfrentan con una diversidad creciente, demandas por tiempos de distribución limitados y desarrollo de software confiable.

¿**Cuáles son los costos de la ingeniería de software?** Aproximadamente 60% de los costos del software son de desarrollo, y 40% de prueba. Para el software elaborado específicamente, los costos de evolución superan con frecuencia los costos de desarrollo.

¿**Cuáles son los mejores métodos y técnicas de la ingeniería de software?** Aun cuando todos los proyectos de software deben gestionarse y desarrollarse de manera profesional, existen diferentes técnicas que son adecuadas para distintos tipos de sistema. Por ejemplo, los juegos siempre deben diseñarse usando una serie de prototipos, mientras que los sistemas críticos de control de seguridad requieren de una especificación completa y analizable para su desarrollo. Por lo tanto, no puede decirse que un método sea mejor que otro.

Detalles generales que afectan la mayoría del software

Heterogeneidad: Cada vez con mayor frecuencia se requieren sistemas que operen como sistemas distribuidos a través de redes que incluyan diferentes tipos de computadoras y dispositivos móviles.

Cambio empresarial y social: Los negocios y la sociedad cambian de manera rápida, conforme se desarrollan las economías emergentes y nuevas tecnologías están a la disposición. Ambos necesitan tener la posibilidad de cambiar su software existente y desarrollar rápidamente uno nuevo.
Seguridad y confianza: Dado que el software está vinculado con todos los aspectos de la vida, es esencial confiar en dicho software.

¿Cuáles son los retos fundamentales que afronta la ingeniería del software?
En el siglo XXI, la ingeniería del software afronta tres retos fundamentales:
1. **El reto de la heterogeneidad**. ~~Cada vez más, se requiere que los sistemas operen como sistemas distribuidos en redes que incluyen diferentes tipos de computadoras y con diferentes clases de sistemas de soporte. A menudo es necesario integrar software nuevo con sistemas heredados más viejos escritos en diferentes lenguajes de programación.~~ El reto de la heterogeneidad es desarrollar técnicas para construir software confiable que sea lo suficientemente flexible ~~para adecuarse a esta heterogeneidad.~~
2**. El reto de la entrega**. ~~Muchas técnicas tradicionales de ingeniería del software consumen tiempo. El tiempo que éstas consumen es para producir un software de calidad. Sin embargo, los negocios de hoy en día deben tener una gran capacidad de respuesta y cambiar con mucha rapidez. Su software de soporte también debe cambiar con la misma rapidez.~~ El reto de la entrega es reducir los tiempos de entrega para sistemas grandes y complejos sin comprometerla calidad del sistema.
3. **El reto de la confianza**~~. Puesto que el software tiene relación con todos los aspectos de nuestra vida, es esencial que podamos confiar en él. Esto es especialmente importante en sistemas remotos de software a los que se accede a través de páginas web o de interfaces de servicios web,~~ El reto de la confianza es desarrollar técnicas que demuestren que los usuarios pueden confiaren el software.

SISTEMAS - APLICACIONES

Hay muchos tipos diferentes de sistemas de software y no existe un conjunto universal de las técnicas de software que es aplicable a todas ellas.
Los métodos de ingeniería de software y las herramientas que se utilizan dependen del tipo de aplicación que se está desarrollando, los requisitos del cliente y los antecedentes del equipo de desarrollo.

Tipos de aplicaciones

Aplicaciones autónomas: Estos son los sistemas de aplicación que se ejecutan en un equipo local, como un PC. Incluyen toda la funcionalidad necesaria y no es necesario estar conectado a una red. Ejemplos de tales aplicaciones son las de oficina en una PC, programas CAD, software de manipulación de fotografías, etcétera.

Aplicaciones basadas en transacciones interactivas: Las aplicaciones que se ejecutan en un equipo remoto y se puede acceder por los usuarios desde sus propios ordenadores o terminales. Esto incluye aplicaciones web como aplicaciones de comercio electrónico.

Sistemas de procesamiento por lotes: Estos son sistemas de negocios que están diseñados para procesar los datos en grandes lotes. Procesan un gran número de entradas individuales para crear salidas correspondientes. Los ejemplos de sistemas batch incluyen sistemas de facturación periódica, como los sistemas de facturación telefónica y los sistemas de pago de salario.

Sistemas de entretenimiento: Se trata de sistemas que son principalmente para su uso personal y que están destinados a entretener al usuario.

Sistemas de control incrustados: Se trata de sistemas de control de software que controlan y gestionan los dispositivos de hardware. Numéricamente, hay probablemente más sistemas integrados que cualquier otro tipo de sistema. Ejemplos de sistemas embebidos incluyen el software en un teléfono móvil (celular), el software que controla los frenos antibloqueo de un automóvil y el software en un horno de microondas para controlar el proceso de cocinado.

Sistemas de recopilación de datos: Se trata de sistemas que recopilan datos de su entorno utilizando un conjunto de sensores y envían los datos a otros sistemas para el procesamiento. El software tiene que interactuar con los sensores y se instala regularmente en un ambiente hostil, como en el interior de un motor o en una ubicación remota.

Sistemas para el modelado y simulación: Éstos son sistemas que desarrollan científicos e ingenieros para modelar procesos o situaciones físicas, que incluyen muchos objetos separados interactuantes. Son computacionalmente intensivos y para su ejecución requieren sistemas paralelos de alto desempeño.

Sistemas de sistemas: Estos son sistemas que están compuestos de un número de otros sistemas de software. Por ejemplo: producto del software genérico, como un programa de hoja de cálculo.

Fundamentos de la ingeniería de software

Algunos principios fundamentales se aplican a todos los tipos de sistema de software, con independencia de las técnicas de desarrollo utilizados:

-Los sistemas deben ser desarrollados mediante un proceso de desarrollo dirigido y entendido. Por supuesto, diferentes procesos se utilizan para diferentes tipos de software.
-La fiabilidad y el rendimiento son importantes para todos los tipos de sistema.
-La comprensión y la gestión de la especificación de requisitos de software y (lo que el software debe hacer) son importantes.
-En su caso, debe volver a utilizar el software que ya ha sido desarrollado en lugar de escribir un nuevo software.

La ingeniería de software y la web

La Web es ahora una plataforma para ejecutar aplicaciones y las organizaciones están desarrollando cada vez más los sistemas basados en la web en lugar de los sistemas locales. Los servicios Web permiten la funcionalidad de la aplicación para acceder a través de Internet. La computación en nube es un enfoque para la prestación de servicios de informática donde las aplicaciones se ejecutan de forma remota en la "nube". Los usuarios no compran software de pago de compra en función del uso.
Reutilización de software es el enfoque dominante para la construcción de sistemas basados en la web. Durante la construcción de estos sistemas, se piensa en cómo puede ser construido a partir de componentes y sistemas de software preexistentes. Los sistemas basados en la Web deben ser desarrollados y entregados de forma incremental. En la actualidad se reconoce en general que no es práctico para especificar todos los requisitos para este tipo de sistemas de anticipación. Las interfaces de usuario están limitadas por las capacidades de los navegadores web. Las tecnologías como AJAX permiten interfaces enriquecidas que se crean dentro de un navegador web, pero siguen siendo difíciles de usar. Formularios Web con scripts locales son más comúnmente utilizados.

Los sistemas basados en la Web son sistemas distribuidos complejos, pero los principios fundamentales de la ingeniería de software previamente discutidos son tan aplicables a ellos como lo son para cualquier otro tipo de sistema. Las ideas fundamentales de la ingeniería de software, que se analizan en la sección anterior, se aplican al software basado en la web de la misma manera que se aplican a otros tipos de sistemas de software.

Por supuesto, éstos no son independientes. Por ejemplo, es necesario hacer cambios rápidos a los sistemas heredados para proveerlos de una interfaz de servicio web. Para tratar estos retos, necesitaremos nuevas herramientas y técnicas, así como formas innovadoras de combinación y uso de métodos de ingeniería del software existentes.

Puntos clave – RESUMEN

-La ingeniería de software es una disciplina de ingeniería que se interesa por todos los aspectos de la producción de software.
-Los atributos esenciales de los productos de software son mantenimiento, confiabilidad, seguridad, eficiencia y aceptabilidad.
-Las actividades de alto nivel de especificación, desarrollo, validación y evolución son parte de todos los procesos de software.
-Existen muchos tipos diferentes de sistemas y cada uno requiere para su desarrollo de herramientas y técnicas adecuadas de ingeniería de software.
-Las ideas fundamentales de la ingeniería de software son aplicables a todos los tipos de sistemas de software.
-Las nociones fundamentales de la ingeniería de software son universalmente aplicables a todos los tipos de desarrollo de sistema

---

Actividades del proceso de software

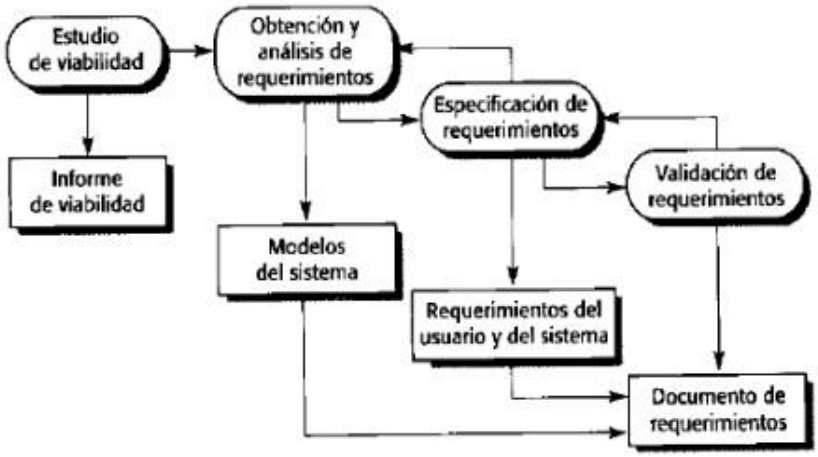
¿Qué es un proceso de software?
~~• Un Proceso es Un conjunto de actividades interrelacionadas que transforman entradas en salidas. (ISO 12207/UNE 77104)~~
~~• Un Proceso Software (PS) es un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software. (Fugetta, 2000)~~
Un proceso del software es un conjunto de actividades y resultados asociados que producen un producto de software. Estas actividades son llevadas a cabo por los ingenieros de software.

Existen **cuatro actividades fundamentales** de procesos que son comunes para todos los procesos del software. Estas actividades son:
1. **Especificación** del software donde los clientes e ingenieros definen el software a producir y las restricciones sobre su operación.
2. **Desarrollo** del software donde el software se diseña y programa.
3. **Validación** del software donde el software se valida para asegurar que es lo que el cliente requiere.
4. **Evolución** del software donde el software se modifica para adaptarlo a los cambios requeridos por el cliente y el mercado.

Diferentes tipos de sistemas necesitan diferentes procesos de desarrollo.
~~Por ejemplo, el software de tiempo real en un avión tiene que ser completamente especificado antes de que empiece el desarrollo, mientras que en un sistema de comercio electrónico, la especificación y el programa normalmente son desarrollados juntos.~~
~~Por lo tanto~~, estas actividades genéricas pueden organizarse de diferentes formas y describirse en diferentes niveles de detalle para diferentes tipos de software. ~~Sin embargo,~~ el uso de un proceso inadecuado del software puede reducir la calidad o la utilidad del producto de software que se va a desarrollar y/o incrementarlos costes de desarrollo.

Especificación de software

•El proceso de establecer qué servicios son requeridos y las limitaciones en el funcionamiento del sistema y el desarrollo.
•Proceso de ingeniería de requerimientos:
-Estudio de viabilidad;
-Obtención y análisis de requerimientos;
-Especificación de los requerimientos;
-Validación de requerimientos.

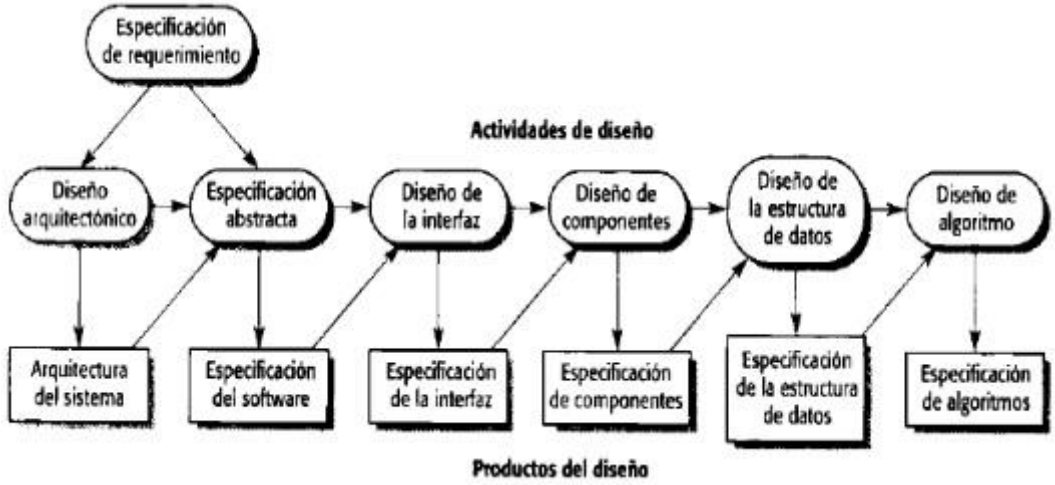


Diseño e implementación de software

•El proceso de convertir la especificación del sistema en un sistema ejecutable.
•Diseño de software: Diseñar una estructura de software que dé constancia de la especificación;
•Implementación: Traducir esta estructura en un programa ejecutable;
•Las actividades de diseño e implementación están estrechamente relacionadas y pueden ser interpoladas.

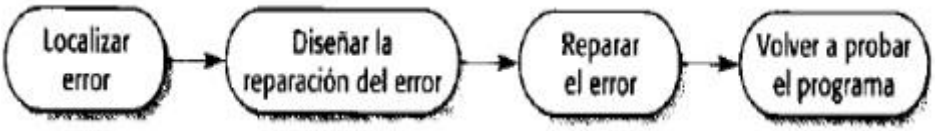
Actividades del proceso de Diseño
•Diseño arquitectónico
•Especificación abstracta
•Diseño de la interfaz
•Diseño de componentes
•Diseño de estructura de datos
•Diseño de algoritmos

El proceso de diseño de software



Métodos estructurados
•Enfoques sistemáticos para el desarrollo de un diseño de software.
•El diseño es por lo general documentado como un conjunto de modelos gráficos.
•Posibles modelos:
-Modelo de objetos;
-Modelo de secuencia;
-Modelo de transición de estados;
-Modelo estructural;
-Modelo de flujo de datos.

Programación y depuración
•Traducción de un diseño en un programa y la eliminación de errores de ese programa.
•La programación es una actividad personal - no hay ningún proceso de programación genérico.
•Los programadores deben acarrear alguna prueba del programa para descubrir defectos en el programa y eliminar esas fallas en el proceso de depuración.



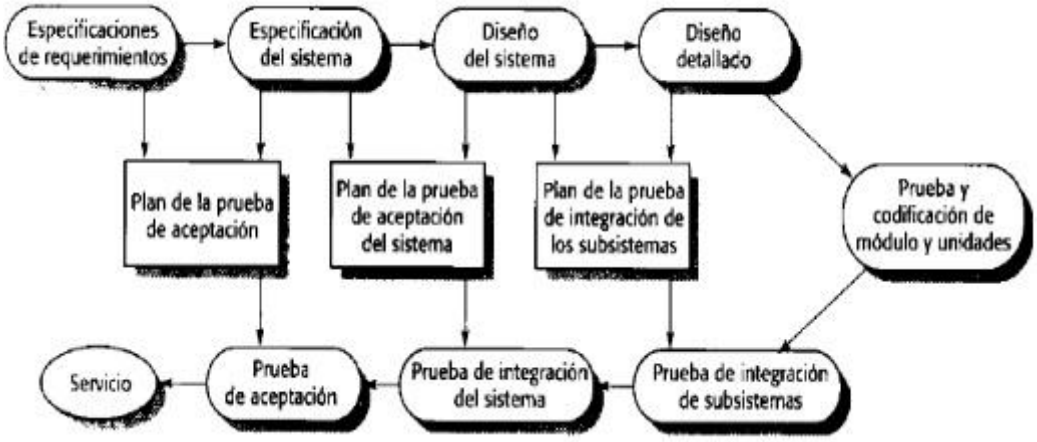
Validación de software

•Verificación y validación (V & V) es para poner de manifiesto que un sistema cumple con su especificación y cumple los requerimientos del cliente.
•Implica el control y revisión de los procesos y pruebas del sistema.
•La prueba del sistema implica la ejecución del sistema con casos de prueba que se derivan de la especificación de los datos reales para ser procesados por el sistema.

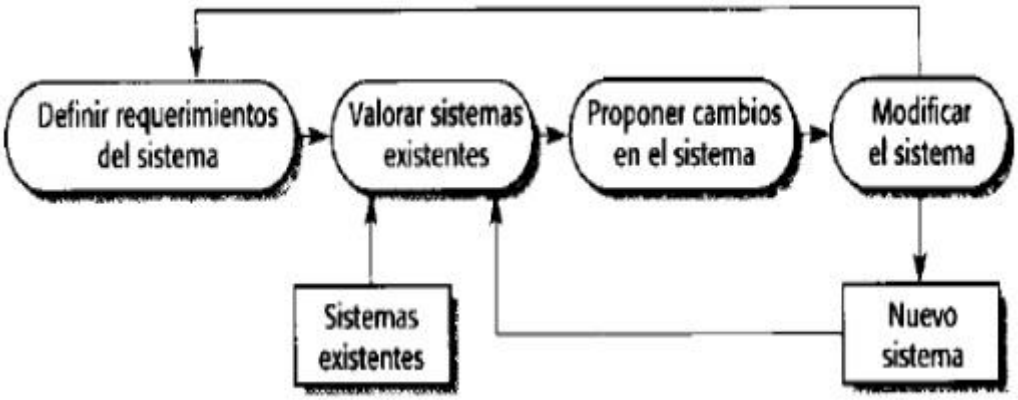
El proceso de pruebas

Etapas del proceso de pruebas
•Prueba de componentes o unidades:
-Cada uno de los componentes son probados por separado;
-Componentes pueden ser funciones u objetos o agrupaciones coherentes de estas entidades.
•Pruebas del sistema:
-Prueba del sistema en su conjunto. Prueba de propiedades emergentes es particularmente importante.
•Pruebas de aceptación:
-Pruebas con los datos de los clientes para comprobar que el sistema cumple con los requerimientos del cliente.

Fases de prueba



Evolución del software
•El software es inherentemente flexible y puede cambiar.
•Como los requerimientos cambian a través del cambio de las circunstancias empresariales, el software que soporta la empresa también debe evolucionar y cambiar.



¿Qué es un modelo de procesos del software?
Un modelo de procesos del software es una descripción simplificada de un proceso del software que presenta **una** visión de ese proceso. Estos modelos pueden incluir actividades que son parte de los procesos y productos de software y el papel de las personas involucradas en la ingeniería del software.

Algunos ejemplos de estos tipos de modelos que se pueden producir son:
• **Un modelo de flujo de trabajo**. Muestra la secuencia de actividades en el proceso junto con sus entradas, salidas y dependencias. Las actividades en este modelo representan acciones humanas.
• **Un modelo de flujo de datos o de actividad**. Representa el proceso como un conjunto de actividades, cada una de las cuales realiza alguna transformación en los datos. Muestra cómo la entrada en el proceso, tal como una especificación, se transforma en una salida, tal como un diseño. Pueden representar transformaciones llevadas a cabo por las personas o por las computadoras.
• **Un modelo de rol/acción**. Representa los roles de las personas involucrada en el proceso del software y las actividades de las que son responsables.

La mayor parte de los modelos de procesos del software se basan en uno de los ***tres modelos generales*** o paradigmas de desarrollo de software:
• **El enfoque en cascada**. Considera las actividades anteriores y las representa como fases de procesos separados, tales como la especificación de requerimientos, el diseño del software, la implementación, las pruebas, etcétera. Después de que cada etapa queda definida «se firma» y el desarrollo continúa con la siguiente etapa.
• **Desarrollo iterativo**. Este enfoque entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones muy abstractas. Este se refina basándose en las peticiones del cliente para producir un sistema que satisfaga las necesidades de dicho cliente. El sistema puede entonces ser entregado. De forma alternativa, se puede reimplementar utilizando un enfoque más estructurado para producir un sistema más sólido y mantenible.
• **Ingeniería del software basada en componentes (CBSE).** Esta técnica supone que las partes del sistema existen. El proceso de desarrollo del sistema se enfoca en la integración de estas partes más que desarrollarlas desde el principio.

CASE

CASE (Ingeniería del Software Asistida por Computadora) comprende un amplio abanico de diferentes tipos de programas que se utilizan para ayudar a las actividades del proceso del software, como el análisis de requerimientos, el modelado de sistemas, la depuración y las pruebas.
En la actualidad, todos los métodos vienen con tecnología CASE asociada, como los editores para las notaciones utilizadas en el método, módulos de análisis que verifican el modelo del sistema según las reglas del método y generadores de informes que ayudan a crear la documentación del sistema. Las herramientas CASE también incluyen un generador de código que automáticamente genera código fuente a partir del modelo del sistema y de algunas guías de procesos para los ingenieros de software.

Las herramientas de desarrollo del software (llamadas en ocasiones herramientas de Ingeniería de Software Asistido por Computadora o CASE, por las siglas de Computer-Aided Software Engineering) son programas usados para apoyar las actividades del proceso de la ingeniería de software. En consecuencia, estas herramientas incluyen editores de diseño, diccionarios de datos, compiladores, depuradores (debuggers), herramientas de construcción de sistema, etcétera. Las herramientas de software ofrecen apoyo de proceso al automatizar algunas actividades del proceso y brindar información sobre el software que se desarrolla.

Los ejemplos de actividades susceptibles de automatizarse son:
■ Desarrollo de modelos de sistemas gráficos, como parte de la especificación de requerimientos o del diseño del software.
■ Generación de código a partir de dichos modelos de sistemas gráficos.
■ Producción de interfaces de usuario a partir de una descripción de interfaz gráfica, creada por el usuario de manera interactiva.
■ Depuración del programa mediante el suministro de información sobre un programa que se ejecuta.
■ Traducción automatizada de programas escritos, usando una versión anterior de un lenguaje de programación para tener una versión más reciente.

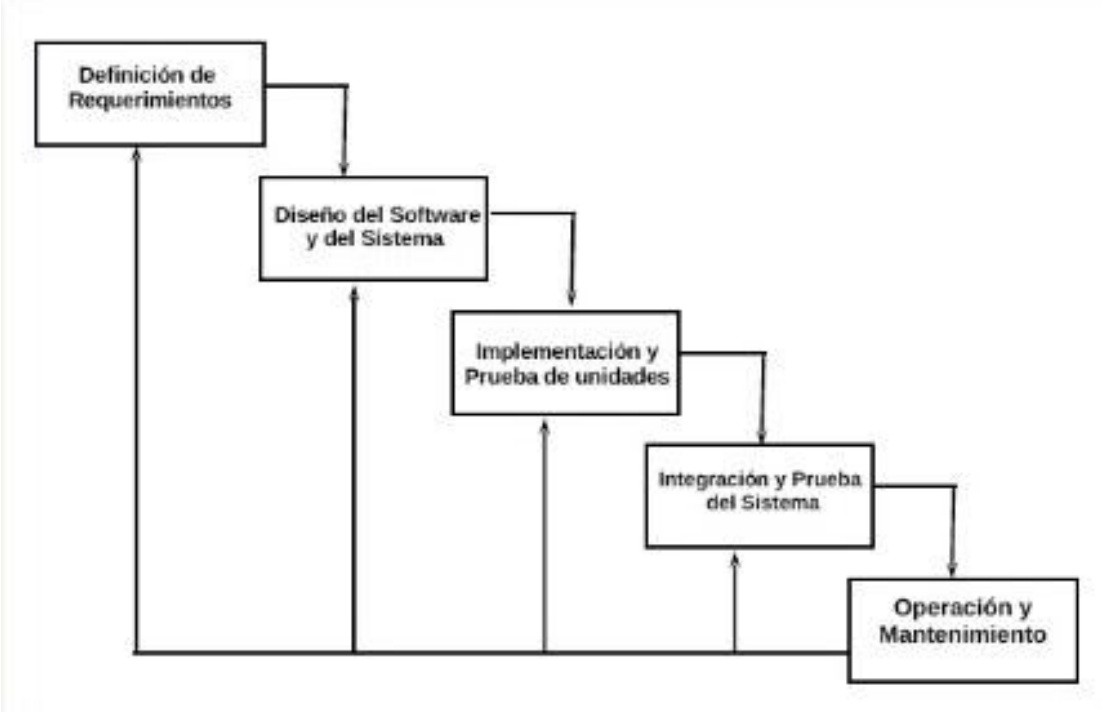
Procesos de software

•Un modelo de proceso del software es una representación abstracta de un proceso. Presenta una descripción de un proceso desde una perspectiva particular.

Modelos genéricos de proceso de software
•El modelo de cascada: Separadas y distintas fases de la especificación y el desarrollo.
•Desarrollo evolutivo: Especificación, desarrollo y validación están intercalados.
•Ingeniería de Software Basada en Componentes: El sistema está montado a partir de los componentes existentes.

Hay muchas variantes de estos modelos, por ejemplo, el desarrollo formal donde se toma un proceso similar al de cascada, pero la especificación es una especificación formal que se perfecciona a través de varias etapas hacia un diseño implementable.

Modelo de cascada – fases



•Análisis de requerimientos y definición
•Diseño del sistema y del software
•Ejecución y unidad de pruebas
•Implementación y pruebas del sistema
•Operación y mantenimiento

Problemas

•El principal inconveniente del modelo de cascada es la dificultad de acomodar el cambio después de que el proceso está en marcha. Una fase tiene que ser completada antes de pasar a la siguiente fase.

•Rígida división del proyecto en fases distintas que hace difícil responder a la evolución de las necesidades del cliente.

Por lo tanto, este modelo sólo es apropiado cuando los requisitos son bien entendidos y los cambios serán bastante limitados durante el proceso de diseño.

Pocos sistemas de negocio tienen requerimientos estables.

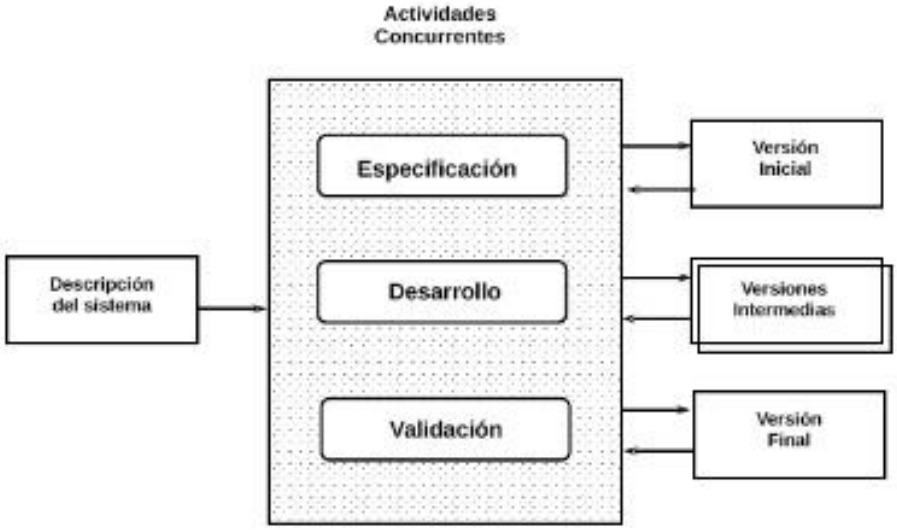
El modelo de cascada se utiliza para grandes proyectos de ingeniería de sistemas donde el desarrollo de un sistema se efectúe en varios sitios.

Desarrollo evolutivo

2 tipos:

•**Desarrollo exploratorio**: El objetivo es trabajar con los clientes y evolucionar a un sistema final a partir de un esbozo inicial de especificación. Debería empezar con buen entendimiento de los requerimientos y añadir nuevas funciones en la forma propuesta por el cliente.

•**Prototipado desechable**: El objetivo es comprender los requisitos del sistema. Se puede empezar con especificaciones poco entendidas.



Problemas

•Poca visibilidad en el proceso;

•Los sistemas son a menudo mal estructurado;

•Habilidades especiales (por ejemplo, lenguajes para prototipado rápido) pueden ser requeridas.

Aplicabilidad

•Para sistemas interactivos pequeños o medianos;

•Para partes de grandes sistemas (por ejemplo, la interfaz de usuario);

•Para sistemas de corta vida.

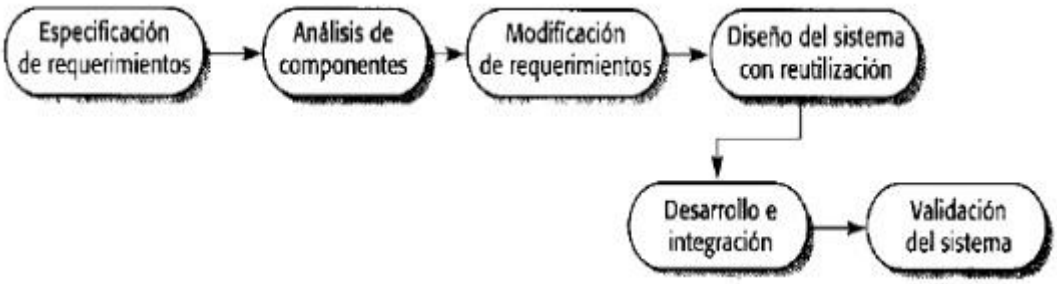
Para sistemas pequeños y de tamaño medio (hasta 500.000 líneas de código), el enfoque evolutivo de desarrollo es el mejor. Los problemas del desarrollo evolutivo se hacen particularmente agudos para sistemas grandes y complejos con un periodo de vida largo, donde diferentes equipos desarrollan distintas partes del sistema. Es difícil establecer una arquitectura del sistema estable usando este enfoque, el cual hace difícil integrar las contribuciones de los equipos.

Para sistemas grandes, se recomienda un proceso mixto que incorpore las mejores características del modelo en cascada y del desarrollo evolutivo. Esto puede implicar desarrollar un prototipo desechable utilizando un enfoque evolutivo para resolver incertidumbres en la especificación del sistema. Puede entonces reimplementarse utilizando un enfoque más estructurado. Las partes del sistema bien comprendidas se pueden especificar y desarrollar utilizando un proceso basado en el modelo en cascada. Las otras partes del sistema, como la interfaz de usuario, que son difíciles de especificar por adelantado, se deben desarrollar siempre utilizando un enfoque de programación exploratoria.

Ingeniería de Software Basada en Componentes

•Sobre la base de la reutilización sistemática donde se integran los sistemas de los componentes existentes o COTS (Comercial-off-the-shelf) de sistemas.
•Las fases del proceso
-Análisis de componentes;
-Modificación de requerimientos;
-Diseño del sistema con reutilización;
-Desarrollo e integración.

•Este enfoque está siendo cada vez más utilizado a medida que los componentes estándar van surgiendo.



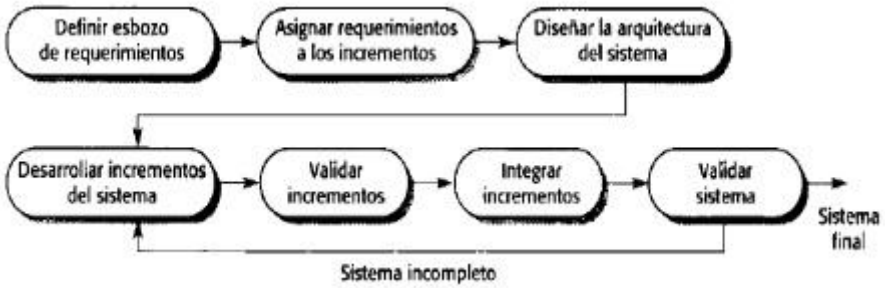
Iteración de Procesos

•Los requerimientos del sistema SIEMPRE evolucionan en el transcurso de un proyecto, así que el proceso de iteración donde las fases son revisadas, siempre son parte del proceso de grandes sistemas.
•Iteración se puede aplicar a cualquiera de los modelos de procesos genéricos.

•Dos enfoques (relacionados)
-Entrega incremental;
-Desarrollo en Espiral.

Entrega Incremental

•En lugar de entregar el sistema en una sola entrega, el desarrollo y la prestación se divide en incrementos, y con cada incremento se entrega parte de la funcionalidad requerida.
•Son priorizados los requerimientos de los usuarios y los requerimientos de más alta prioridad son incluidos en los primeros incrementos.
•Una vez que el desarrollo de un incremento se ha iniciado, los requisitos están congelados, a pesar de que los requerimientos de los incrementos posteriores pueden seguir evolucionando.



~~Ventajas del Desarrollo incremental~~

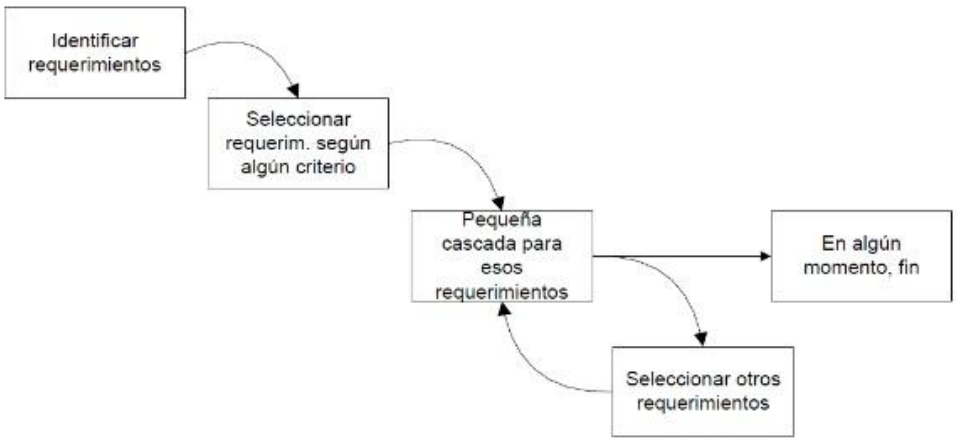
~~•Entregas con valor para el cliente pueden ser hechas con cada incremento, de manera que la funcionalidad del sistema está disponible tempranamente.~~

~~•Los primeros incrementos actúan como un prototipo para ayudar a obtener requerimientos para incrementos posteriores.~~

~~•Menor riesgo de fracaso del proyecto general.~~

~~•La más alta prioridad del sistema de servicios tiende a recibir más pruebas.~~

Modelos iterativos e incrementales (o evolutivos)



Principales ventajas

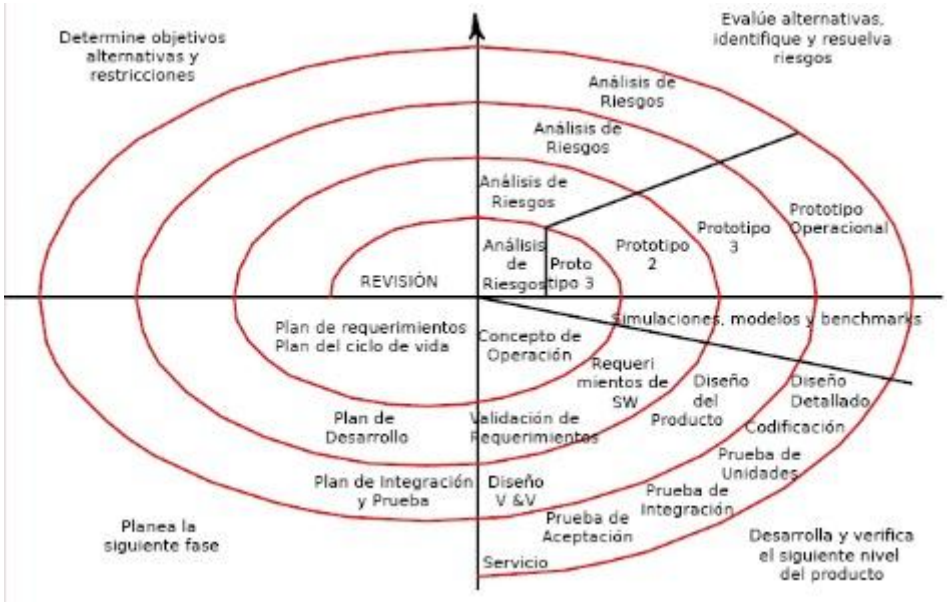
-El usuario ve algo rápidamente. La funcionalidad del sistema está disponible tempranamente.
-Se admite que lo que se está construyendo es el sistema, y por lo tanto se piensa en su calidad desde el principio. La más alta prioridad del sistema de servicios tiende a recibir más pruebas.
-Se pueden atacar los principales riesgos (disminuye posibilidad de fracaso del proyecto)
-Los ciclos van mejorando con las experiencias de los anteriores

Principales variaciones

-Duración de las iteraciones
-Existencia o no de tipos de iteraciones
-Cantidad de esfuerzo dedicado a identificación inicial de requerimientos
-Actitud "defensiva" ante los cambios

Desarrollo en Espiral

•El proceso se representa como una espiral y no como una secuencia de actividades con marcha atrás.
•Cada bucle en la espiral representa una fase en el proceso.
•Fases no fijas, tales como las fases de diseño o especificación - bucles de la espiral se eligen en función de lo que se necesita.
•Riesgos se evalúan de forma explícita y se resuelven a través todo el proceso.



Sectores del modelo en espiral

•La fijación de objetivos: -Los objetivos específicos de cada fase son identificados.

•Evaluación de riesgos y su reducción: -Se evalúan los riesgos y las actividades puestas en marcha para reducir los principales riesgos.

•Desarrollo y validación: -Un modelo de desarrollo para el sistema elegido es el que puede ser cualquiera de los modelos genéricos.

•Planificación: -El proyecto es revisado y la próxima fase de la espiral es planeada.

Modelo de proceso	Funciona con requisitos y arquitectura no predefinidos	Produce software altamente fiable	Gestión de riesgos	Permite correcciones sobre la marcha	Visión del progreso por el Cliente y el Jefe del proyecto
Cascada	Bajo	Alto	Bajo	Bajo	Bajo
Evolutivo exploratorio	Medio o Alto	Medio o Alto	Medio	Medio o Alto	Medio o Alto
Evolutivo prototipado	Alto	Medio	Medio	Alto	Alto
Incremental	Bajo	Alto	Medio	Bajo	Bajo
Espiral	Alto	Alto	Alto	Medio	Medio

RUP

METODOLOGÍA PURA

•Es una metodología cuyo fin es entregar un producto de software.

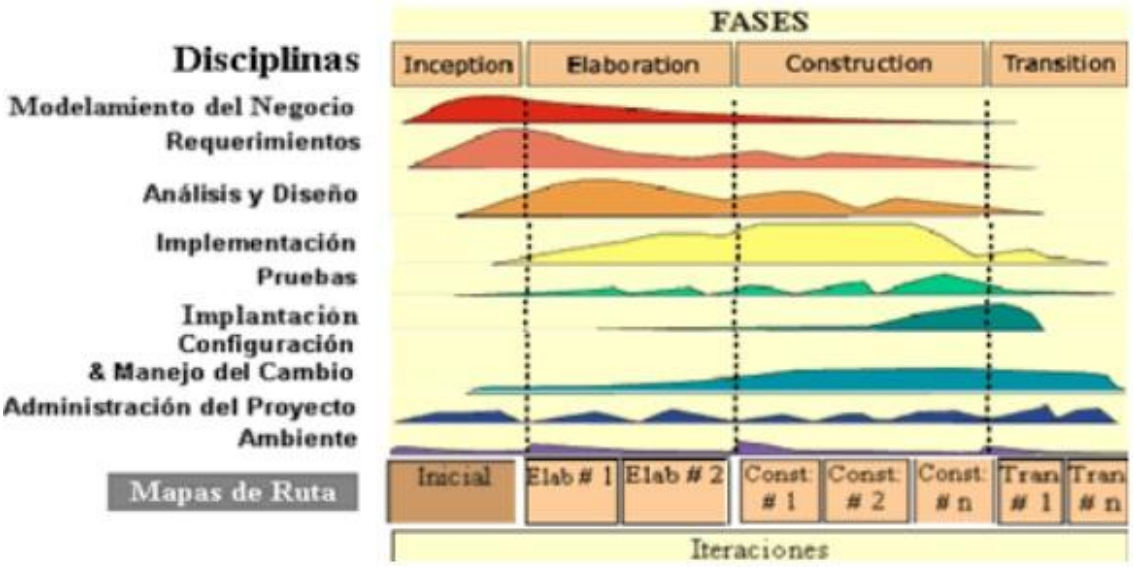
•Se estructura todos los procesos y se mide la eficiencia de la organización.

•El RUP es un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

•Describe como aplicar enfoques para el desarrollo del software, llevando a cabo unos pasos para su realización.

•Se centra en la producción y mantenimiento de modelos del sistema.

Es un proceso de desarrollo de software el cual utiliza el lenguaje unificado de modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.
El RUP es un producto de Rational (IBM). Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como, por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

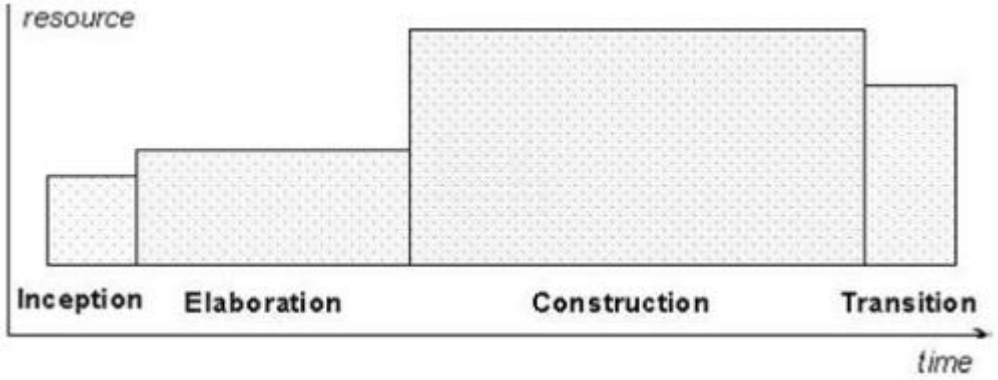


Principales características
•Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo)
•Pretende implementar las mejores prácticas en Ingeniería de Software.
•Desarrollo iterativo
•Administración de requisitos
•Uso de arquitectura basada en componentes
•Control de cambios
•Modelado visual del software
•Verificación de la calidad del software

Esfuerzo en actividades según fase del proyecto
El ciclo de vida RUP es una implementación del Desarrollo en espiral.
Fue creado ensamblando los elementos en secuencias semi-ordenadas.
El ciclo de vida organiza las tareas en fases e iteraciones.
RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades.

Fases del ciclo de vida del RUP:

1. Fase de Inicio
2. Fase de Elaboración
3. Fase de Desarrollo
4. Fase de Cierre / Transición



1. Fase de Inicio

Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones posteriores.

2. Fase de Elaboración

En la fase de elaboración se seleccionan los casos de uso que permiten definir la arquitectura base del sistema y se desarrollaran en esta fase, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar.

3. Fase de Desarrollo

El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar los cambios de acuerdo a las evaluaciones realizados por los usuarios y se realizan las mejoras para el proyecto.

4. Fase de Cierre / Transición

El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitara los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto.

La metodología RUP tiene 6 principios claves:

1. Adaptación del proceso: El proceso debe adaptarse a las características de la organización para la que se está desarrollando el software.
2. Equilibrar prioridades: Debe encontrarse un balance que satisfaga a todos los inversores del proyecto (los deseos de todos).
3. Colaboración entre equipos: Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, entre otros.
4. Demostrar valor iterativamente: Los proyectos se entregan, aunque sea de una forma interna, en etapas iteradas. En cada iteración se evaluará la calidad y estabilidad del producto y analizará la opinión y sugerencias de los inversores.
5. Elevar el nivel de abstracción: Motivar el uso de conceptos reutilizables.
6. Enfocarse en la calidad: La calidad del producto debe verificarse en cada aspecto de la producción.

Disciplina de desarrollo de RUP: Determina las etapas a realizar durante el proyecto de creación del software.
Ingeniería o modelado del negocio: Analizar y entender las necesidades del negocio para el cual se está desarrollando el software.
Requisitos: Proveer una base para estimar los costos y tiempo de desarrollo del sistema.
Análisis y diseño: Trasladar los requisitos analizados anteriormente a un sistema automatizado y desarrollar una arquitectura para el sistema.
Implementación: Crear software que se ajuste a la arquitectura diseñada y que tenga el comportamiento deseado.
Pruebas: Asegurarse de que el comportamiento requerido es correcto y que todo lo solicitado está presente.
Despliegue: Producir distribuciones del producto y distribuirlo a los usuarios.

Disciplina de soporte RUP: Determina la documentación que es necesaria realizar durante el proyecto.
Configuración y administración del cambio: Guardar todas las versiones del proyecto.
Administración del proyecto: Administrar los horarios y recursos que se deben de emplear.
Ambiente: Administrar el ambiente de desarrollo del software.
Distribución: Hacer todo lo necesario para la salida del proyecto.

Elementos del RUP
Actividades: Procesos que se han de realizar en cada etapa/iteración.
Trabajadores: Personas involucradas en cada actividad del proyecto.
Artefactos: Herramientas empleadas para el desarrollo del proyecto.
Puede ser un documento, un modelo, un elemento del modelo.

RUP - Artefactos
RUP en cada una de sus fases (pertenecientes a la estructura estática) realiza una serie de artefactos que sirven para comprender mejor tanto el análisis como el diseño del sistema (entre otros). Estos artefactos (entreotros) son los siguientes:
Inicio
•Documento Visión
~~•Diagramas de caso de uso~~
•Especificación de Requisitos
~~•Diagrama de Requisitos~~
Elaboración:
•Diagramas de caso de uso
Construcción:
•Documento Arquitectura que trabaja con las siguientes vistas:
VISTA LOGICA
Diagrama de clases
Modelo Entidad-Relación (Si el sistema así lo requiere)
VISTA DE IMPLEMENTACION
Diagrama de Secuencia
Diagrama de Estados
Diagrama de Colaboración
VISTA CONCEPTUAL
Modelo de dominio
~~*VISTA FISICA*~~
~~Mapa de comportamiento a nivel de hardware.~~

El Proceso Unificado de Rational

•Un moderno modelo de proceso derivado de la labor sobre el UML y procesos asociados.
•Normalmente se describe a partir de 3 de perspectivas
-Una perspectiva dinámica, que muestra las fases del modelo en el tiempo;
-Una perspectiva estática que muestra las actividades de proceso;
-Una perspectiva práctica que sugiere buenas prácticas a utilizar durante el proceso.

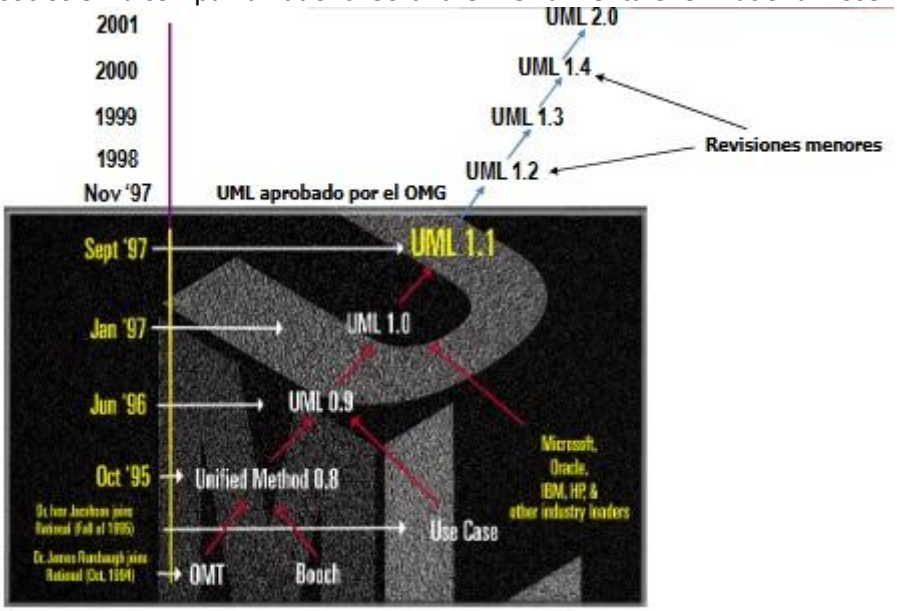
Modelado del software
Ejemplos
1-Construcción de una cucha para un perro:
Puede hacerlo una sola persona
Requiere:
-Modelado mínimo
-Proceso simple
-Herramientas simples
2-Construcción de una casa:
-Construida eficientemente y en un tiempo razonable de un equipo
-Modelado
-Proceso bien definido
-Herramientas más sofisticadas
3-Construcción de un rascacielos
Contexto de desarrollo
-Determinar configuración del proceso
-Recursos necesarios
-Herramientas más sofisticadas aún.

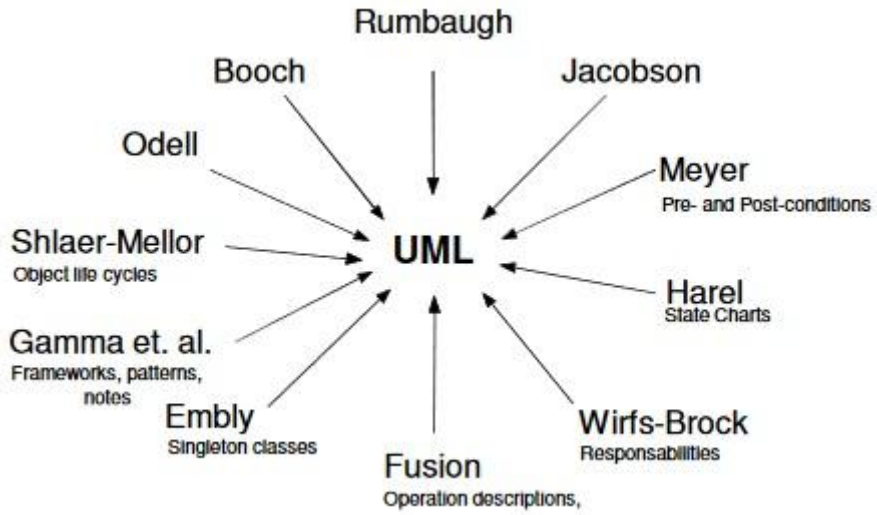
UML

UML = Unified Modeling Language
Un lenguaje de propósito general para el modelado orientado a objetos
Documento "OMG Unified Modeling Language Specification"
UML combina notaciones provenientes desde:
Modelado Orientado a Objetos
Modelado de Datos
Modelado de Componentes
Modelado de Flujos de Trabajo (Workflows)

Motivos de creación:
1-Diversos métodos y técnicas OO, con muchos aspectos en común pero utilizando distintas notaciones
2-Inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc.
3-Pugna entre distintos enfoques (y correspondientes gurús)
4-Establecer una notación estándar

Historia:
Comenzó como el "Método Unificado", con la participación de Grady Booch y Jim Rumbaugh. Se presentó en el OOPSLA'95. El mismo año se unió Ivar Jacobson. Los "Tres Amigos" son socios en la compañía Rational Software. Herramienta CASE Rational Rose.





<u>Aspectos Novedosos</u>
Definición semi-formal del Metamodelo de UML
Mecanismos de Extensión en UML:
Stereotypes
Constraints
Tagged Values
Permiten adaptar los elementos de modelado, asignándoles una semántica particular

Inconvenientes en UML
Definición del proceso de desarrollo usando UML.
UML no es una metodología
Falta integración con respecto de otras técnicas tales como patrones de diseño, interfaces de usuario, documentación, etc.
"Monopolio de conceptos, técnicas y métodos en torno a UML"

Perspectivas de UML
UML será el lenguaje de modelado orientado a objetos estándar predominante los próximos años
Razones:
-Participación de metodólogos influyentes
-Participación de importantes empresas
-Aceptación del OMG como notación estándar
Evidencias:
-Herramientas que proveen la notación UML
-"Edición" de libros
-Congresos, cursos, etc.

Modelos y Diagramas
Un modelo captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema, considerando un cierto propósito. Así, el modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.
Diagrama: una representación gráfica de una colección de elementos de modelado, a menudo dibujada como un grafo con vértices conectados por arcos.
Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés
~~El código fuente del sistema es el modelo más detallado del sistema (y además es ejecutable).~~
Sin embargo, se requieren otros modelos ...
Cada modelo es completo desde su punto de vista del sistema, sin embargo, existen relaciones de trazabilidad entre los diferentes modelos.

Diagramas de UML
Diagrama de Casos de Uso
Diagrama de Clases
Diagrama de Objetos
Diagramas de Comportamiento
Diagrama de Estados
Diagrama de Actividad
Diagramas de Interacción
Diagrama de Secuencia
Diagrama de Colaboración
Diagramas de implementación
Diagrama de Componentes
Diagrama de Despliegue

Los diagramas expresan gráficamente partes de un modelo.

Organización de Modelos

4+1 vistas de Kruchten (1995)

Este enfoque sigue el browser de Rational Rose

~~-El modelado de sistemas es el proceso para desarrollar modelos abstractos de un sistema, donde cada modelo presenta una visión o perspectiva diferente de dicho sistema. En general, el modelado de sistemas se ha convertido en un medio para representar el sistema usando algún tipo de notación gráfica, que ahora casi siempre se basa en notaciones en el Lenguaje de Modelado Unificado (UML).~~

-Los modelos se usan durante el proceso de ingeniería de requerimientos para ayudar a derivar los requerimientos de un sistema, durante el proceso de diseño para describir el sistema a los ingenieros que implementan el sistema, y después de la implementación para documentar la estructura y la operación del sistema.

Es posible desarrollar modelos tanto del sistema existente como del sistema a diseñar:
1. Los modelos del sistema existente se usan durante la ingeniería de requerimientos: Ayudan a aclarar lo que hace el sistema existente y pueden utilizarse como base para discutir sus fortalezas y debilidades. Posteriormente, conducen a los requerimientos para el nuevo sistema.
2. Los modelos del sistema nuevo se emplean durante la ingeniería de requerimientos: para ayudar a explicar los requerimientos propuestos a otros participantes del sistema. Los ingenieros usan tales modelos para discutir las propuestas de diseño y documentar el sistema para la implementación. En un proceso de ingeniería dirigido por modelo, es posible generar una implementación de sistema completa o parcial a partir del modelo del sistema.

El aspecto más importante de un modelo del sistema es que deja fuera los detalles. Un modelo es una abstracción del sistema a estudiar, y no una representación alternativa de dicho sistema. De manera ideal, una representación de un sistema debe mantener toda la información sobre la entidad a representar. Una abstracción simplifica y recoge deliberadamente las características más destacadas. Por ejemplo, en el muy improbable caso de que este libro se entregue por capítulos en un periódico, la presentación sería una abstracción de los puntos clave del libro. Si se tradujera del inglés al italiano, sería una representación alternativa. La intención del traductor sería mantener toda la información como se presenta en inglés.
Desde diferentes perspectivas, usted puede desarrollar diferentes modelos para representar el sistema. Por ejemplo:
1. Una perspectiva externa, donde se modelen el contexto o entorno del sistema.
2. Una perspectiva de interacción, donde se modele la interacción entre un sistema y su entorno, o entre los componentes de un sistema.
3. Una perspectiva estructural, donde se modelen la organización de un sistema o la estructura de datos que procese el sistema.

4. Una perspectiva de comportamiento, donde se modele el comportamiento dinámico del sistema y cómo responde ante ciertos eventos.

El UML tiene numerosos tipos de diagramas y, por lo tanto, soporta la creación de muchos diferentes tipos de modelo de sistema. Sin embargo, un estudio en 2007 (Erickson y Siau, 2007) mostró que la mayoría de los usuarios del UML consideraban que **cinco tipos de diagrama** podrían representar lo **esencial** de un sistema.
1.Diagramas de actividad, que muestran las actividades incluidas en un proceso o en el procesamiento de datos.
2.Diagramas de caso de uso, que exponen las interacciones entre un sistema y su entorno.
3.Diagramas de secuencias, que muestran las interacciones entre los actores y el sistema, y entre los componentes del sistema.
4.Diagramas de clase, que revelan las clases de objeto en el sistema y las asociaciones entre estas clases.
5.Diagramas de estado, que explican cómo reacciona el sistema frente a eventos internos y externos.

El Lenguaje de Modelado Unificado es un conjunto compuesto por 13 diferentes tipos de diagrama para modelar sistemas de software. ~~Surgió del trabajo en la década de 1990 sobre el modelado orientado a objetos, cuando anotaciones similares, orientadas a objetos, se integraron para crear el UML. Una amplia revisión (UML 2) se finalizó en 2004~~. El UML es aceptado universalmente como el enfoque estándar al desarrollo de modelos de sistemas de software. ~~Se han propuesto variantes más generales para el modelado de sistemas.~~

~~Propuesta de Rational Unified Process (RUP)~~
~~M. de Casos de Uso del Negocio (Business Use-Case Model)~~
~~M. de Objetos del Negocio (Business Object Model)~~
~~M. de Casos de Uso (Use-Case Model)~~
~~M. de Análisis (Analysis Model)~~
~~M. de Diseño (Design Model)~~
~~M. de Despliegue (Deployment Model)~~
~~M. de Datos (Data Model)~~
~~M. de Implementación (Implementation Model)~~
~~M. de Pruebas (Test Model)~~

Paquetes UML
Los paquetes ofrecen un mecanismo general para la organización de los modelos/subsistemas agrupando elementos de modelado.
Cada paquete corresponde a un submodelo (subsistema) del modelo (sistema)
Un paquete puede contener otros paquetes, sin límite de anidamiento pero cada elemento pertenece a (está definido en) sólo un paquete.
Una clase de un paquete puede aparecer en otro paquete por la importación a través de una relación de dependencia entre paquetes.
Todas las clases no son necesariamente visibles desde el exterior del paquete, es decir, un paquete encapsula a la vez que agrupa.
El operador "::" permite designar una clase definida en un contexto distinto del actual.

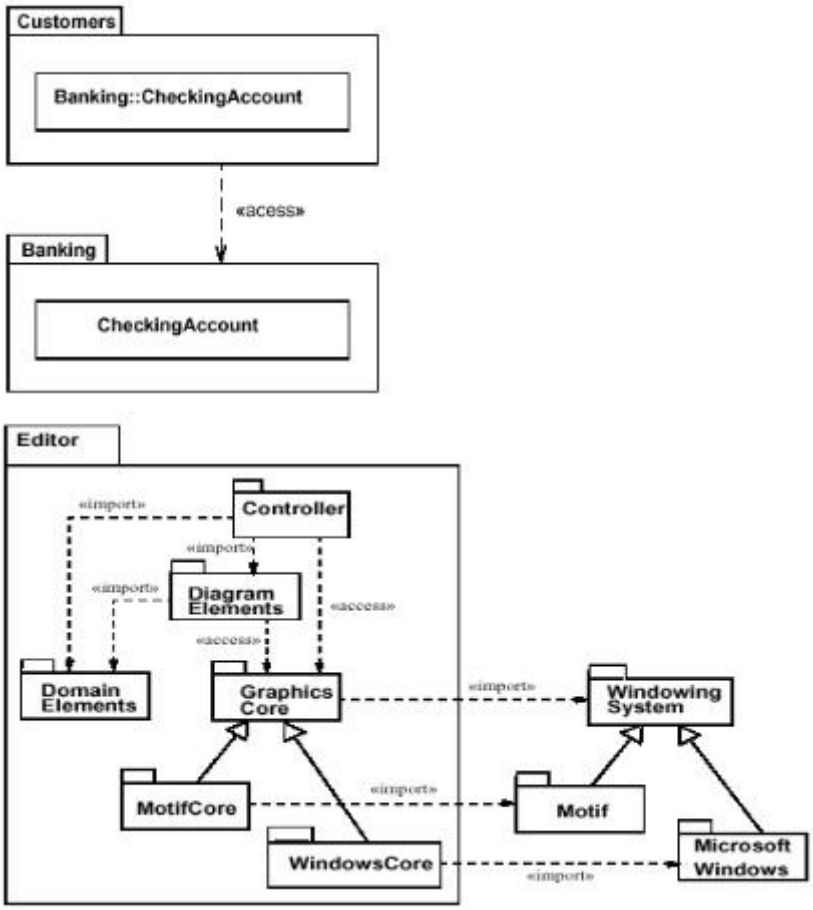
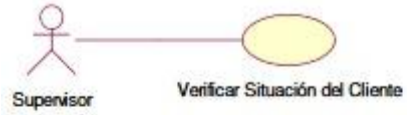


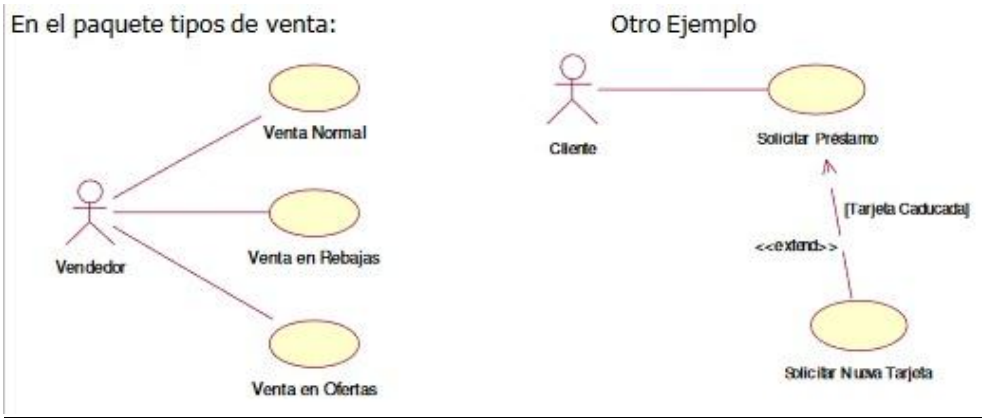
Diagrama de Casos de Uso

Casos de Uso es una técnica para capturar información de cómo un sistema o negocio trabaja, o de cómo se desea que trabaje

No pertenece estrictamente al enfoque orientado a objeto, es una técnica para capturar de requisitos



Ejemplos



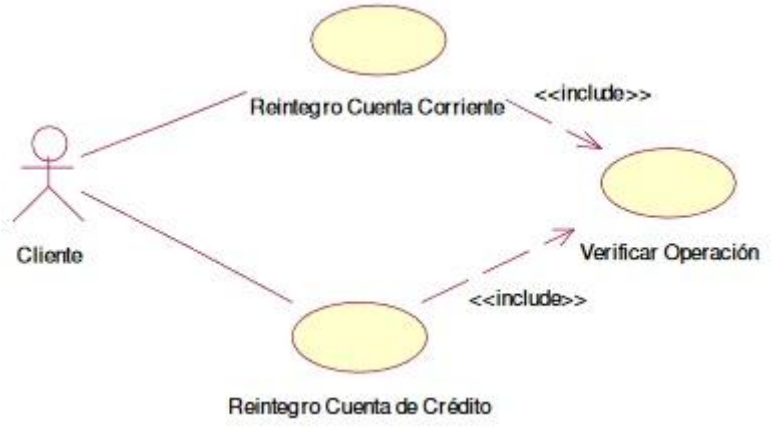


Diagrama de secuencias

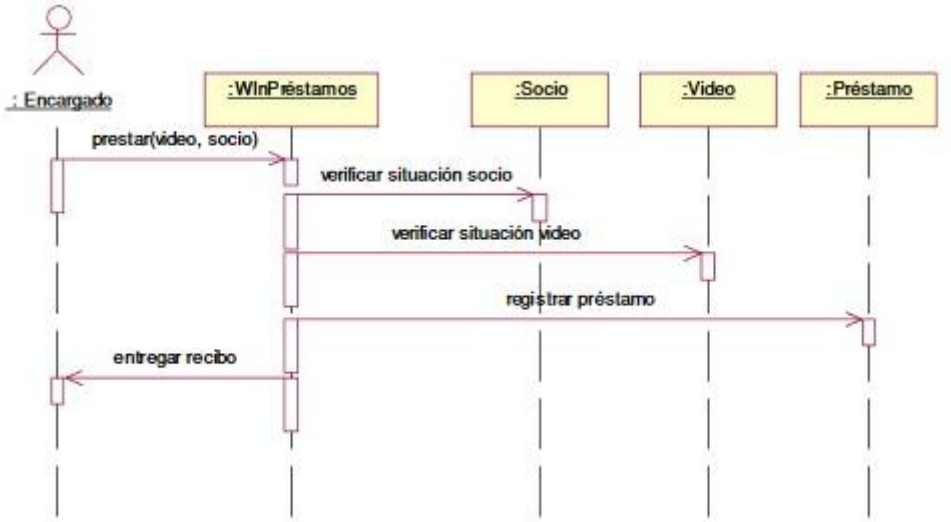


Diagrama de Colaboración

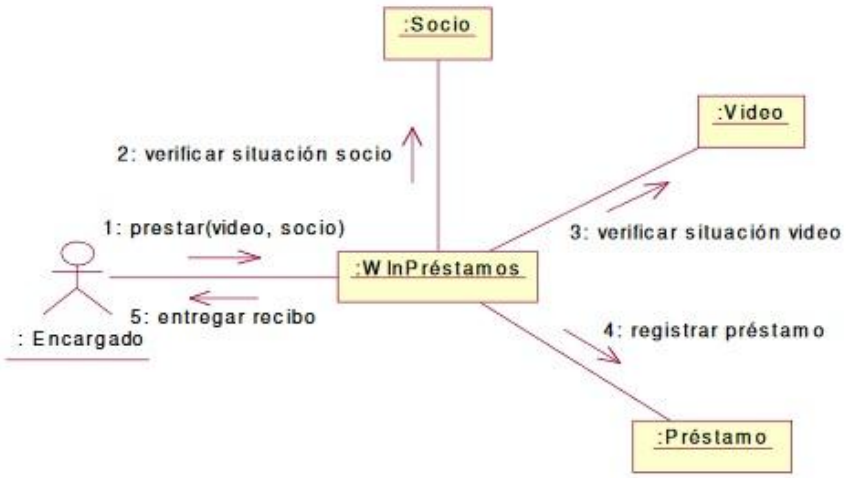


Diagrama de Clases
El Diagrama de Clases es el diagrama principal para el análisis y diseño
Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia
La definición de clase incluye definiciones para atributos y operaciones
El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones

UNIDAD II

Requerimientos

•El proceso de establecimiento de los servicios que el cliente requiere de un sistema y las limitaciones con las que opera y se desarrolla.
•Los requerimientos son la descripción de los servicios del sistema y las limitaciones que se generan durante el proceso de ingeniería de requerimientos.

¿Qué es un requerimiento?
Puede ir desde una declaración de un servicio con un alto nivel de abstracción o de una limitación del sistema a una detallada especificación funcional formal.
Esto es inevitable, ya que los requerimientos pueden servir una doble función
-Puede ser la base para una oferta para un contrato - por lo tanto debe estar abierto a la interpretación;
-Puede ser la base del contrato en sí - por lo tanto, debe definirse en detalle;
-Ambas declaraciones pueden llamarse requerimientos.

Abstracción de requerimientos (Davis)

Si una compañía desea establecer un contrato para un proyecto de desarrollo de software grande, debe definir sus necesidades de una forma suficientemente abstracta para establecer a partir de ella una solución. Los requerimientos deben redactarse de tal forma que varios contratistas pueden licitar el contrato, ofreciendo, quizás, formas diferentes de cumplir las necesidades de los clientes en la organización. Una vez que el contrato se asigna, el contratista debe redactar una definición del sistema para el cliente más detalladamente de forma que éste comprenda y pueda validar lo que hará el software. Ambos documentos se pueden denominar documento de requerimientos para el sistema.

Tipos de requerimientos

Requerimientos de usuario

Declaraciones en lenguaje natural y los esquemas de los servicios que proporciona el sistema y sus limitaciones operacionales. Escrito para los clientes.

Requerimientos del sistema

Un documento estructurado que establece la descripción detallada de las funciones del sistema, los servicios y las limitaciones operacionales. Define lo que debe aplicarse, de manera que puede ser parte de un contrato entre el cliente y el contratista.

Ejemplo:

El sistema LIBSYS

Un sistema de bibliotecas que proporciona una única interfaz para una serie de bases de datos de artículos en diferentes bibliotecas. Los usuarios pueden buscar, descargar e imprimir estos artículos para su estudio personal.

Definiciones y especificaciones

Definición del requerimiento del usuario

1. LIBSYS controlará todos los datos requeridos por las agencias que licencian los derechos de autor en el Reino Unido y en otra parte.

Especificación de los requerimientos del sistema

1.1 Al hacer una petición de un documento del LIBSYS, el solicitante se presentará con un formulario que registre los detalles del usuario y de la petición hecha.
1.2 El formulario de petición del LIBSYS será almacenado en el sistema durante cinco años desde la fecha de la petición.
1.3 Todos los formularios de peticiones del LIBSYS se deben indexar por usuario, por el nombre del material solicitado y por el proveedor de la petición.
1.4 El LIBSYS mantendrá un fichero en el que se registrarán todas las peticiones que se han hecho al sistema.
1.5 Para el material donde se aplican los derechos de préstamo del autor, los detalles del préstamo serán enviados mensualmente a las agencias que licencian los derechos de autor que se han registrado con LIBYSS.

Requisitos de los lectores

Requerimientos funcionales y no funcionales

Requerimientos funcionales

Declaraciones de los servicios que debe proporcionar el sistema, la forma en que el sistema debe reaccionar a las entradas y la forma en que el sistema debe comportarse en situaciones particulares.

Requerimientos no funcionales

limitaciones en los servicios o funciones ofrecidas por el sistema como de tiempo, limitaciones en el proceso de desarrollo, normas, etc.

Requerimientos del dominio

Requerimientos que se derivan del dominio de aplicación del sistema y que reflejan las características de ese dominio.

Los requisitos funcionales definen qué debe hacer un sistema.
Los requisitos no funcionales definen cómo debe ser el sistema.

Los requerimientos funcionales

-Describir las funciones o servicios del sistema.
-Dependerá del tipo de software, de los posibles usuarios y del tipo de sistema en el que el software se utiliza.
-Los requerimientos funcionales de los usuarios señalan a un alto nivel de abstracción lo que el sistema debe hacer, pero los requerimientos funcionales del sistema deben describir los servicios del sistema de forma detallada.

Ejemplos de requerimientos funcionales

-El usuario será capaz de buscar, ya sea la totalidad de la serie inicial de las bases de datos o seleccionar un subconjunto de ella.
-El sistema deberá proporcionar vistas apropiadas para que el usuario pueda leer los documentos en la tienda de documentos.
-A cada orden se le asignará un identificador único (ORDEN_ID) que el usuario será capaz de copiar a la cuenta del área de almacenamiento permanente.

Imprecisión de los requerimientos

-Los problemas surgen cuando los requerimientos no son declarados con precisión.

-Requerimientos ambiguos pueden interpretarse de diferentes maneras por los desarrolladores y usuarios.
-Considerar el término “visores apropiados”
*Intención del usuario: visor de propósito especial para cada tipo de documento diferente;
*Interpretación del desarrollador: Proporcionar un visor de texto que muestra el contenido del documento.

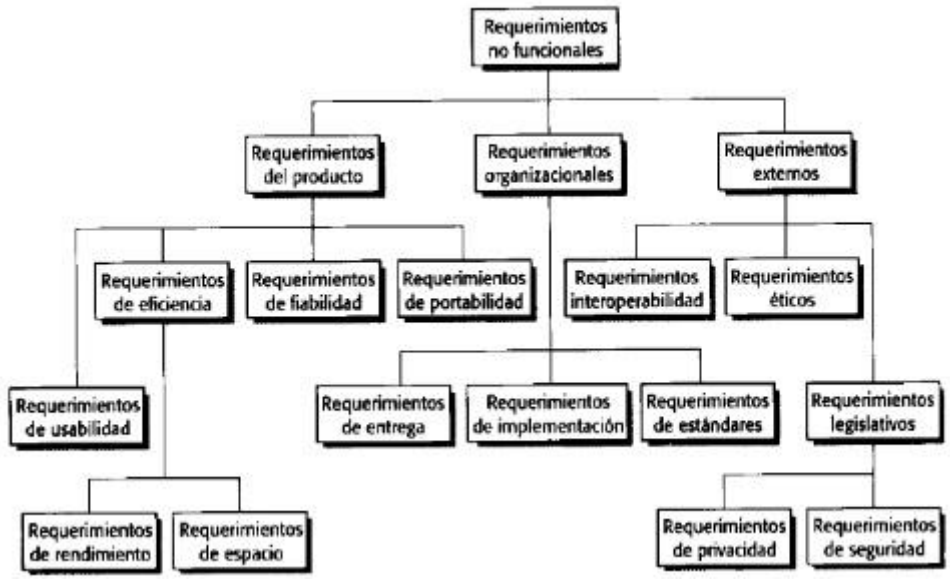
Integridad de requerimientos y consistencia
-En principio, los requerimientos deben estar completos y ser consistentes.
-Completitud: Todos los servicios solicitados por el usuario deben estar definidos.
-Consistencia: No debería haber conflictos o contradicciones en las descripciones de los requerimientos del sistema.
-En la práctica, es imposible producir un documento de requerimientos completo y consistente.

Requerimientos no funcionales
-Estos definen las propiedades emergentes del sistema y las limitaciones, por ejemplo fiabilidad, tiempo de respuesta y los requerimientos de almacenamiento. Las limitaciones son, capacidad de dispositivos de E / S, representaciones del sistema, etc.
-El proceso de requerimientos también puede ser especificado al asignarse una herramienta CASE particular, lenguaje de programación o metodología de desarrollo.
-Los requerimientos no funcionales pueden ser más críticos que los requerimientos funcionales. Si estos no se encuentran, el sistema es inútil.

Tipos de requerimientos no funcionales

-Requerimientos del producto: Requerimientos que especifican que el producto entregado debe comportarse de una manera particular, por ejemplo la velocidad de ejecución, la fiabilidad, etc.
-Requerimientos organizacionales: Requerimientos que son consecuencia de las políticas de la organización y procedimientos como, por ejemplo, estándares de procesos utilizados, requerimientos de implementación, etc.
-Requerimientos externos: Requisitos que derivan de factores que son externos al sistema y su proceso de desarrollo, por ejemplo, los requerimientos de interoperabilidad, los requerimientos legislativos, etc.

A los requisitos no funcionales se les suele llamar coloquialmente “cualidades” del sistema [“ilities” en inglés”] y pueden dividirse en dos categorías:
-Cualidades de ejecución, como la seguridad o la usabilidad, observables en tiempo de ejecución.
-Cualidades de evolución, como la “testabilidad”, mantenibilidad, extensibilidad o escalabilidad, determinadas por la estructura estática del software.



Ejemplos de requerimientos no funcionales

Requerimientos del producto: La interfaz de usuario para LIBSYS se ejecutará como HTML simple sin marcos o applets de Java.

Requerimientos organizacionales: El proceso de desarrollo del sistema y la entrega de documentos se ajustará conforme al proceso y entregas definidos en XYZCo-SP-STAN-95.

Requerimientos externos: El sistema no podrá divulgar cualquier información personal sobre los clientes, aparte de su nombre y número de referencia a los operadores del sistema.

Metas y requerimientos

-Los requerimientos no funcionales pueden ser muy difíciles de precisar y requerimientos imprecisos pueden ser difíciles de verificar.

-Meta: En general, la intención del usuario, como la facilidad de uso del sistema.

-Requerimiento no funcional verificable: Una declaración tangible puede ser objetivamente probada.

-Las metas son provechosas para los desarrolladores pues transportan las intenciones de los usuarios del sistema.

Ejemplos

Una meta del sistema

El sistema debería ser fácil de utilizar por los controladores con experiencia, y debe organizarse de tal manera que se reduzcan al mínimo los errores de usuario.

Un requerimiento no funcional verificable

Controladores experimentados deberán ser capaces de utilizar todas las funciones del sistema después de un total de dos horas de formación. Después de esta formación, el número promedio de errores cometidos por los usuarios experimentados no deberá exceder de dos por día.

Métricas para los requerimientos

Propiedad	Medida
Rapidez	Transacciones procesadas por segundo. Tiempo de respuesta al usuario y a eventos Tiempo de actualización de la pantalla
Tamaño	K Bytes Número de chips de RAM
Facilidad de uso	Tiempo de formación Número de cuadros de ayuda
Fiabilidad	Tiempo medio entre fallos Probabilidad de no disponibilidad Tasa de ocurrencia de fallos Disponibilidad
Robustez	Tiempo de reinicio después de fallos Porcentaje de eventos que provocan fallos Probabilidad de corrupción de los datos después de fallos
Portabilidad	Porcentaje de declaraciones dependientes del objetivo Número de sistemas objetivo

Interacción de requerimientos

Los conflictos entre los diferentes requerimientos no funcionales son comunes en sistemas complejos.

Ejemplo:

•Sistema de nave espacial

-Para minimizar el peso, el número de chips separados en el sistema debe reducirse al mínimo.

-Para reducir al mínimo el consumo de energía, los chips de menor consumo de energía se debe utilizar.

-Sin embargo, utilizar chips de bajo consumo de energía puede significar que más chips tienen que ser utilizados. Cuál es el requerimiento más crítico?

Requerimientos del dominio

•Se derivan del dominio de la aplicación del sistema y reflejan los fundamentos del dominio de la aplicación.

•Pueden ser requerimientos funcionales nuevos, restringir los existentes o establecer cómo se deben ejecutar cálculos particulares.

•Si los requerimientos del dominio no se satisfacen, puede ser imposible hacer que el sistema funcione de forma satisfactoria.

Ejemplos

Requerimientos del dominio de un sistema de biblioteca

-Se establece un estándar de interfaz de usuario para todas las bases de datos que se basará en la norma Z39.50.

-Debido a restricciones de derechos de autor, algunos de los documentos deberán suprimirse de inmediato a su llegada. Dependiendo de las exigencias del usuario, estos documentos se imprimirán localmente en el servidor del sistema para su distribución manual al usuario o enviarse a una impresora de red.

Sistema de protección del tren

La desaceleración del tren se calculará como:

Dtren = Dcontrol + Dgradiente

Donde Dgradiente = 9.81ms2 * gradiente compensado / alfa y donde los valores de 9.81ms2 / alfa son conocidos para los diferentes tipos de tren.

Problemas de los requerimientos del dominio

Comprensibilidad

•Los requerimientos son expresados en el lenguaje del dominio de la aplicación;

•Esto a menudo no es entendido por los ingenieros de software que desarrollan el sistema.

Lo implícito

•Los expertos en el dominio pueden dejar fuera de un requerimiento información, sencillamente porque para ellos es obvia, por lo que no piensan en hacer explícitos los requerimientos del dominio.

Requerimientos del usuario

-Deben describir los requerimientos funcionales y no funcionales de tal forma que sean comprensibles por los usuarios del sistema que no tienen conocimientos técnicos detallados.

-Los requerimientos del usuarios se definen utilizando lenguaje natural, tablas y diagramas que puedan ser comprendidas por todos los usuarios.

Problemas con el lenguaje natural

-Falta de claridad: Es difícil utilizar el lenguaje de forma precisa sin hacer el documento poco conciso y difícil de leer.

-Confusión de requerimientos: Los requerimientos funcionales y no funcionales tienden a ser mezclados.

-Conjunción de requerimientos: Varios requerimientos diferentes, pueden expresarse de manera conjunta.

Problemas con especificaciones en lenguaje natural

Ambigüedad: Los lectores y escritores de los requerimientos deben interpretar las mismas palabras de la misma manera. El lenguaje natural es ambiguo por lo que este, naturalmente, es muy difícil.

El exceso de flexibilidad: Lo mismo puede decirse en una serie de diferentes maneras en la especificación.

La falta de modularización: Estructuras en lenguaje natural no son suficientes para estructurar los requisitos del sistema.

Ejemplos:

Requerimiento de usuario para un sistema de compatibilidad LIBSYS

4 .. 5 LIBSYS presentará un sistema de contabilidad financiera, que mantiene registros de todos los pagos efectuados por los usuarios del sistema.

Los administradores del sistema pueden configurar el sistema para que los usuarios habituales puedan recibir tarifas de descuento.

Requerimiento de usuario para un editor de cuadrícula

Recursos de la cuadrícula: Para ayudar a la ubicación de entidades en un diagrama, el usuario puede activar una cuadrícula en centímetros o pulgadas, a través de una opción en el panel de control. Inicialmente, la cuadrícula está desactivada. La cuadrícula se puede activar y desactivar en cualquier momento durante una sesión de edición y poner en pulgadas y centímetros. La opción de cuadrícula se proporcionará en vista de reducción de ajuste, pero el número de líneas de la cuadrícula a mostrar se reducirá para evitar saturar el diagrama más pequeño con líneas de cuadrícula.

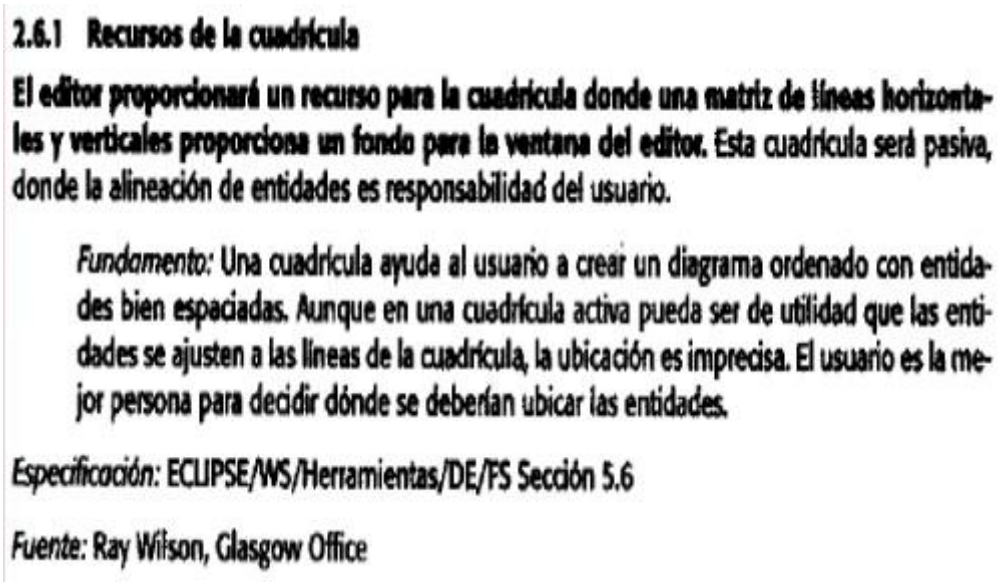
Problemas de requerimientos

Los requerimientos de la base de datos incluyen información tanto conceptual como detallada

-Describe el concepto de un sistema de contabilidad financiera que debe ser incluido en LIBSYS;

-Sin embargo, también se incluye el detalle de que los administradores pueden configurar este sistema – esto es innecesario en este nivel.

Los requerimientos para con la cuadrícula mezcla tres tipos de requerimientos
-Un requerimiento funcional conceptual (la necesidad de una cuadrícula);
-Un requerimiento no funcional (unidades de la cuadrícula);
-Un requerimiento de la interfaz de usuario no funcional (activación o desactivación de la cuadrícula).



2.6.1 Recursos de la cuadrícula

El editor proporcionará un recurso para la cuadrícula donde una matriz de líneas horizontales y verticales proporciona un fondo para la ventana del editor. Esta cuadrícula será pasiva, donde la alineación de entidades es responsabilidad del usuario.

Fundamento: Una cuadrícula ayuda al usuario a crear un diagrama ordenado con entidades bien espaciadas. Aunque en una cuadrícula activa pueda ser de utilidad que las entidades se ajusten a las líneas de la cuadrícula, la ubicación es imprecisa. El usuario es la mejor persona para decidir dónde se deberían ubicar las entidades.

Especificación: ECLIPSE/WS/Herramientas/DE/FS Sección 5.6

Fuente: Ray Wilson, Glasgow Office

Directrices para la redacción de los requerimientos
-Inventar un formato estándar y asegurar la adherencia al mismo para todos los requerimientos.
-Utilizar el lenguaje de manera consistente. Para con los requerimientos obligatorios, debería usarse para con los requerimientos deseable.
-Resaltar el texto para distinguir las partes clave del requerimiento.
-Evite el uso de jerga informática.

Requerimientos del sistema
•Especificaciones más detalladas de las funciones del sistema, los servicios y las limitaciones que con los requerimientos del usuario.
•Su intención es la de ser una base para el diseño el sistema.
•Pueden ser incorporados en el contrato del desarrollo del sistema.
•Los requerimientos del sistema se puede definir o ilustrar mediante modelos de sistemas.

Requerimientos y diseño
En principio, los requerimientos deben indicar qué debe hacer el sistema y el diseño debe describir cómo se hace esto.
En la práctica, los requerimientos y el diseño son inseparables
-Una arquitectura de sistema puede ser diseñada para estructurar los requerimientos;
-El sistema puede inter-operar con otros sistemas que generan requerimientos de diseño;
-El uso de un diseño específico puede ser un requerimiento del dominio.

Alternativas a la especificación en lenguaje natural

Notación	Descripción
Lenguaje natural estructurado	Este enfoque depende de la definición de formularios o plantillas estándares para expresar la especificación de requerimientos.
Lenguajes de descripción de diseño	Este enfoque utiliza un lenguaje similar a uno de programación, pero con características más abstractas, para especificar los requerimientos por medio de la definición de un modelo operativo del sistema. Este enfoque no se utiliza ampliamente en la actualidad, aunque puede ser útil para especificaciones de interfaces.
Notaciones gráficas	Para definir los requerimientos funcionales del sistema, se utiliza un lenguaje gráfico, complementado con anotaciones de texto. Uno de los primeros lenguajes gráficos fue SADT (Ross, 1977) (Schoman y Ros, 1977). Actualmente, se suelen utilizar las descripciones de casos de uso (Jacobsen *et al.*, 1993) y los diagramas de secuencia (Stevens y Pooley, 1999).
Especificaciones matemáticas	Son notaciones que se basan en conceptos matemáticos como el de las máquinas de estado finito o los conjuntos. Estas especificaciones no ambiguas reducen los argumentos sobre la funcionalidad del sistema entre el cliente y el contratista. Sin embargo, la mayoría de los clientes no comprenden las especificaciones formales y son reacios a aceptarlas como un contrato del sistema.

Especificaciones en lenguaje estructurado
-La libertad del escritor de los requerimientos está limitada por una plantilla predefinida para requerimientos.
-Todos los requerimientos están escritos en una manera estándar.
-La terminología utilizada en la descripción puede ser limitada.
-La ventaja es que la mayor parte de la expresividad del lenguaje natural es mantenida, pero un grado de uniformidad se impone en la especificación.

Especificaciones basadas en formulario
-Definición de la función o entidad.
-Descripción de las entradas y de dónde vienen.
-Descripción de las salidas y hacia dónde van.
-Indicación de otras entidades requeridas.
-Pre y post condiciones (si es apropiado).
-Descripción de los efectos colaterales (si los hubiera) de la operación.

SoftwaredecontroI/bombaInsulina/SRS/3.3.2

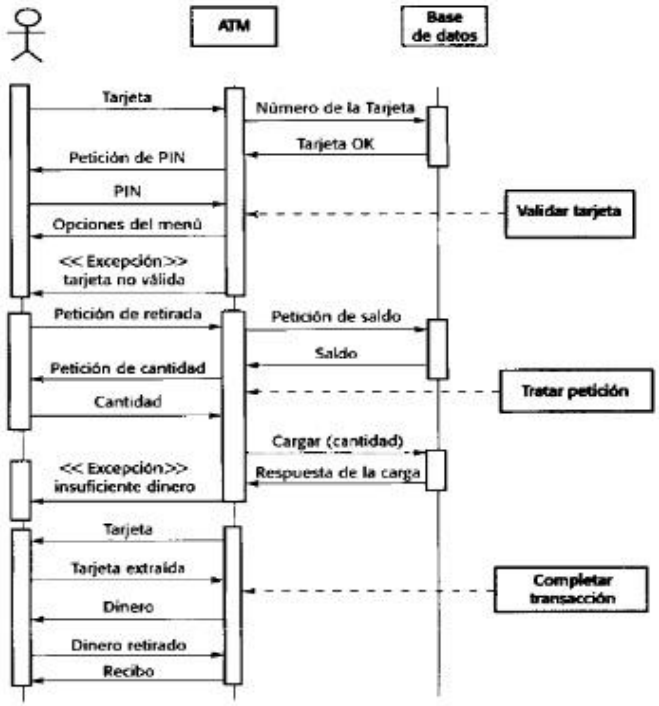
Función	Calcular la dosis de insulina: nivel de azúcar en sangre.
Descripción	Calcula la dosis de insulina a suministrar cuando el nivel medido actual del casured sugar level is in azúcar está en la zona segura entre 3 y 7 unidades.
Inputs	Lectura del azúcar actual (r2), las dos lecturas previas (r0 y r1).
Fuente	Lectura actual del azúcar del sensor. Otras lecturas de la memoria.
Outputs	CompDose: la dosis en insulina a suministrar.
Destino	Bucle de control principal.
Acción:	CompDose es cero si el nivel de azúcar está estable o disminuyendo o si el nivel está aumentando pero la tasa de incremento disminuyendo. Si el nivel está aumentando y la tasa de incremento disminuyendo, CompDose se calcula dividiendo la diferencia entre el nivel de azúcar actual y el nivel anterior por 4 y redondeando el resultado. Si el resultado se redondea a cero, se fija CompDose a la dosis mínima que puede ser suministrada.
Requerimientos	Las dos lecturas previas para poder calcular la tasa de cambio del azúcar.
Pre condición	La reserva de insulina contiene al menos el máximo permitido para una única dosis de insulina.
Post-condición	r0 es reemplazada por r1 y r1 es reemplazada por r2.
Efectos colaterales	Ninguno.

Modelo gráfico

Son muy útiles cuando se necesita mostrar cómo cambia el Estado o en las necesita describir una secuencia de acciones.

Diagramas de secuencia
•Estas muestran la secuencia de los eventos que tienen lugar durante la interacción del usuario con el sistema.
•Usted lee de arriba a abajo para ver el orden de las acciones que se llevan a cabo.
•Retiro de efectivo de un cajero automático
-Validar la tarjeta;
-Tratar la petición;
-Completar la transacción.

Diagrama de secuencia de retiro en cajeros automáticos



Especificación de la interfaz
*La mayoría de los sistemas deben funcionar con otros sistemas y las interfaces operativas deben ser especificadas como parte de los requerimientos.
*Tres tipos de interfaz puede que tengan que ser definidas
-Interfaces de procedimientos;
-Estructuras de datos que se intercambian;
-Representaciones de datos.
*Las notaciones formales son una técnica eficaz para la especificación de la interfaz

Descripción en PDL de una interfaz

```
Interface PrintServer {
// define un servidor de impresión abstracto
// requiere: interfaz Printer, interfaz PrintDoc
// proporciona: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter
   void initialize ( Printer p ) ;
   void print ( Printer p, PrintDoc d ) ;
   void displayPrintQueue ( Printer p ) ;
   void cancel PrintJob (printer p, PriDoc d) ;
   void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

El documento de requerimientos
-El documento de requerimientos es la declaración oficial de lo que se requiere de los desarrolladores del sistema.
-Debe incluir una definición de los requerimientos de usuario y una especificación de los requerimientos del sistema.
-No se trata de un documento de diseño. Como máximo, debería establecer lo que el sistema debe hacer en lugar de cómo debe hacerlo.

Los usuarios de un documento de requerimiento

Requisitos estándar IEEE
Define una estructura genérica para un documento de requerimientos que debe ser instanciada para cada sistema específico.
-Introducción.
-Descripción general.
-Requerimientos específicos.
-Apéndices.
-Índice.

Estructura de un documento de requerimientos
-Prefacio
-Introducción
-Glosario
-Definición de requerimientos del usuario
-Arquitectura del sistema
-Especificación de requerimientos del sistema
-Modelos del sistema
-Evolución del sistema
-Apéndices
-Índice

Puntos clave
-Los requerimientos determinan lo que debe hacer el sistema y definen las restricciones en su funcionamiento e implementación.
-Los requerimientos funcionales establecen los servicios que el sistema debe proporcionar.
-Los requerimientos no funcionales restringen el sistema en desarrollo y el proceso de desarrollo que se debe utilizar.
-Los requerimientos de usuario son declaraciones de alto nivel de lo que el sistema debe hacer. Los requerimientos de usuario deben ser escritos utilizando el lenguaje natural, tablas y diagramas.
-Los requerimientos del sistema tienen por objeto comunicar las funciones o servicios que el sistema debe proporcionar.
-Un documento de requerimientos de software es una declaración de los requerimientos del sistema.
-La estándar de la IEEE es un punto de partida útil para la definición de estándares de requerimientos más detallados.

Los casos de uso...
- Describen el modo en que un actor interactúa con el sistema (descripción de un rol en lenguaje natural).
- Narran el comportamiento dinámico del sistema desde un punto de vista concreto (el del actor).
- Pueden expresar tanto requerimientos funcionales como no funcionales.
- Son muy útiles para explicar el funcionamiento del sistema, priorizar requerimientos cuando el sistema se desarrolla de forma incremental, elaborar manuales de usuario y especificar pruebas de aceptación.
- Mejoran la trazabilidad de los requerimientos durante el proceso de desarrollo de software.
- Se pueden desarrollar en paralelo con los requerimientos del sistema de forma iterativa.

Dependiendo de la situación, los casos de uso se pueden especificar con distinto grado de detalle:
Especificación textual de un caso de uso (enumeración de pasos del caso de uso).
Especificación "esencial" de un caso de uso (eliminando todos los detalles no estrictamente necesarios).
Especificación detallada de un caso de uso(utilizando una plantilla para no olvidarnos de nada).

Especificación textual de un caso de uso

Actor	Profesor
Rol	Consultar estadísticas

- El profesor ejecuta el programa de consulta de estadísticas.
- Se le pide su identificativo (*login*) y palabra clave de acceso (*password*).
- El sistema verifica la identificación del usuario.
- Si la identificación es positiva, se presenta una lista con las estadísticas disponibles:
  - Nº de alumnos y porcentaje de repetidores de sus asignaturas.
  - Clasificación de alumnos por nota en cada asignatura.
- Una vez que el profesor ha seleccionado una de las estadísticas, el programa presenta los datos correspondientes a la misma, agrupando la información por asignaturas y, al final, para todas sus asignaturas en conjunto.
- Al profesor se le da la opción de imprimir la estadística.
- Cuando el profesor termina de ver la estadística, se presenta de nuevo la lista de estadísticas disponibles.
- Si no desea ver otra estadística, termina la ejecución de la aplicación.

Especificación esencial de un caso de uso

Profesor	Sistema
El profesor se identifica.	
	El sistema autentifica al profesor y le ofrece una lista de estadísticas disponibles.
El profesor selecciona una de las opciones disponibles.	
	El sistema presenta un informe con los datos solicitados.
Si así lo desea, el profesor imprime el informe.	

Especificación detallada de un caso de uso

Nombre	Consulta de estadísticas
Descripción	Se permite a los profesores consultar las estadísticas correspondientes a sus asignaturas
Dependencias	Autentificación de usuarios
Actores	Profesor (principal e iniciador)
Precondiciones	-
Postcondiciones	-

Escenario principal	**Profesor**	**Sistema**
	1. El profesor se identifica.	
		2. El sistema autentifica al profesor y le ofrece una lista de estadísticas disponibles.
	3. El profesor selecciona una de las opciones.	
		4. El sistema presenta un informe con los datos solicitados.
	5. Si así lo desea, el profesor imprime el informe.	
Alternativas		2. Si, tras un tercer intento, la autentificación no se realiza con éxito, se guarda la incidicencia en un registro y se impide volver a acceder a la aplicación desde la misma IP durante 15 minutos.
Observaciones	-	
Requisitos no funcionales	El sistema debe estar preparado para aceptar 100 sesiones simultáneas de profesores consultando sus estadísticas sin degradar su rendimiento más de un 50% con respecto a un usuario único.	