Que es la ingenieria?:

- La ingeniería como actividad humana es la aplicación del conocimiento y los métodos científicos al diseño y la producción de productos complejos.

Difiriendo en el producto, todas las ingenierías tienen en común:

- **La ciencia de la ingeniería**: principios y mecanismos subyacentes de la disciplina.
- **Procesos de diseño**: fase de conceptualización y una fase de diseño detallado.
- **Aspectos de gestión y organización**: la tecnología que se produce implica a personas y organizaciones.

Además, las personas que crean tecnología no suelen trabajar aisladas, sino en equipos y organizaciones.

En el caso de la ingeniería del software:

- Las actividades de diseño (empleado en el sentido general de la ingeniería, no diseño de software que es una fase concreta de la ing de sw) serían asimilables a lo que normalmente conocemos como **”desarrollo”.**

- La ciencia de la ingeniería que nos interesa cuando estudiamos la ingeniería del software es la **ciencia de la computación** está asociada a esta ingeniería, pues abarca los principios matemáticos y físicos, en

su sentido más amplio, de los sistemas basados en computadora.
No obstante, es importante distinguir claramente entre ciencia de la computación e ingeniería del software, ya que lo específico de esta última es lo **concerniente al diseño y uso del software, utilizando el conocimiento que es el objeto de la ciencia de la computación.**

Ahora bien, dentro de la ciencia de la ingeniería del software, hay que separar los conocimientos científicos que se aplican en la ingeniería del software, la ciencia de la ingeniería del software en sí misma, y la práctica de la ingeniería:
Las ciencias que se aplican en la ingeniería del software son la ciencia de la computación y otras ciencias que son de utilidad para aspectos determinados, como las relativas a la organización, la economía, la psicología, y por supuesto, las matemáticas en general. Para dominios muy concretos también se necesitan **conocimientos específicos de ciertas ciencias** (biología en biotecnología…)

La ingeniería del software como ciencia es la aplicación del método científico a la teorización y creación de conocimiento sobre la propia ingeniería del software. Está dedicada al estudio de sus

actividades, y centrada en generar teorías, modelos explicativos o enunciados descriptivos sobre la práctica de la ingeniería

La práctica de la ingeniería , que está orientada a prescribir cómo deben realizarse las actividades propias de la disciplina. Es un aspecto complementario con la ciencia de la ingeniería , pues la ciencia necesita de la observación de la práctica, y la práctica a su vez se perfecciona de acuerdo con el conocimiento generado por la ciencia

SOFTWARE:

Software es el **conjunto completo de programas, procedimientos y documentación relacionada que se asocia con un sistema**, y especialmente con un sistema de computadora.
En un sentido específico, software son los programas de computadora: conjunto de instrucciones destinadas a que una computadora lleve a cabo una tarea.

El software es la tecnología o **producto resultante de las actividades de ingeniería del software**. Pero téngase en cuenta que el software tiene una naturaleza que lo diferencia de otros productos de la ingeniería moderna, lo que hace que tenga una problemática también especial. De hecho, desde los años sesenta hasta ahora, se ha

venido utilizando el término “crisis del software” para hacer referencia a ciertos problemas específicos y persistentes de la ingeniería del software.

El software es inherentemente complejo

Aunque algunos aspectos de su desarrollo tienen **complejidad accidental** (codificación o pruebas), para los cuales sí pueden crearse herramientas que gestionen su complejidad,

Su **complejidad inherente** reside en que:

- El software tiene propensión al cambio: consecuencia del uso del software, ya que al ser utilizado, debe adaptarse a nuevos requisitos y necesidades; El software evoluciona: es modificable y flexible
- El software es invisible: el software no se puede representar completamente mediante diagramas, pues dicha representación sería extremadamente compleja, un plus de complejidad en la comprensión del mismo.

QUE ES LA INGENIERÍA DEL SW?:

1. **La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, la operación y el mantenimiento del software.**
2. **El estudio de enfoques como los mencionados en** (1.)

Según la primera de estas definiciones, el ingeniero de software es un “desarrollador” en sentido amplio, que desempeña un rol como profesional en la producción de software. Por su parte, la segunda de las definiciones implica la **investigación y estudio de las actividades de la ingeniería del software,** pero no el producir software. Así, define para el ingeniero de software un perfil de “investigador”.

Características:

- Sigue un plan de manera metódica (la actividad de ing del sw **es sistemática**).
- Respeta ciertos estándares (normas o patrones) **(es disciplinada)**
- Sus resultados y el propio proceso pueden medirse **(es cuantificable)**

Historia del Software:

- 1955-1965: se desarrollan los principios iniciales de la disciplina.

- 1965-1985 se identifican los problemas en el desarrollo del software (en 1968 se acuña el término en las conferencias de la OTAN): la identificación del problema de la crisis del software se convierte en el tema central de la disciplina
- 1985: Con el artículo de Brooks comienza el desarrollo de la ingeniería pensando que el software es algo inherentemente complejo.

Conceptos básicos:

La ingeniería se desarrolla en la forma de **actividades de ingeniería.** Toda actividad sucede en un intervalo temporal determinado, y se distingue de otras actividades por los elementos que intervienen en la misma.

- **Actividad**: proceso que tiene lugar en el tiempo y en el espacio, y en el cual un agente actúa con unos objetivos determinados.
    - En la ingeniería del software abarcan por tanto cualquier acción con un propósito claro dentro de esta ingeniería, lo que incluye actividades de producción, comunicación, de prueba, de obtención de requisitos, de documentación, de gestión…
- **Artefacto**: algo tangible creado con un propósito práctico. Fruto de una actividad. Se consideran "elementos de información" (todos son susceptibles

de proporcionar información en el proceso de ingeniería).
    - Son artefactos de la ingeniería del software todos aquellos elementos creados en actividades propias de la disciplina, tales como el código, los documentos, los diagramas, lista de requisitos….

La Ingeniería del Software se materializa en términos de actividades, de sus participantes y de los artefactos que producen, transforman o utilizan, con un enfoque sistemático y disciplinado

Las actividades que tienen lugar y los artefactos que se crean en la ingeniería en general están sujetos a **normas que dictan cómo deben hacerse (métodos)**

- **Método**: especificación de una secuencia de acciones orientadas a un propósito determinado. Determinan el orden y la forma de llevar a cabo las actividades

El término metodología hace referencia al **estudio de los métodos** (sea general o específico de una disciplina.), aunque también puede utilizarse para hacer referencia a un **conjunto coherente de métodos.** Ej: Metodologías Orientada a Objetos, metodologías estructurada

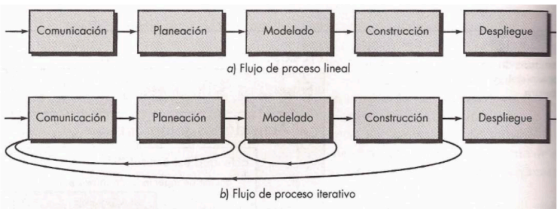
- **Metodología**: (en ing sw): conjunto de métodos coherentes y relacionados por unos principios comunes

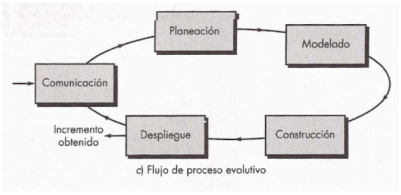
- **Especificación**: descripción detallada y precisa de algo existente (o que existirá) o de una cierta situación, presente o futura. Para elaborar especificaciones se utilizan lenguajes o notaciones.

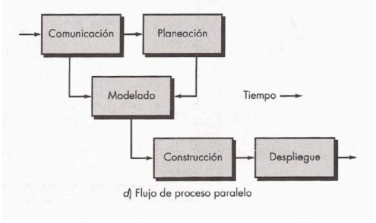
- **Ciclo de vida del software**: evolución del mismo desde el momento de su concepción hasta el momento en que deja de usarse, y puede describirse en función de las actividades que se realizan dentro de él (el concepto no se restringe a las actividades de desarrollo previas al uso del software, sino que abarca también su evolución y mantenimiento).”Comienza en el momento en que se identifica una necesidad o problema que debe resolverse mediante un sistema informático”

- **Modelos de ciclos de vida**: No se está haciendo referencia al ciclo de vida de un software determinado, sino a una especificación de cuáles deben ser las fases o el curso general de la ingeniería del software.

- **Proceso software**: conjunto coherente de políticas, estructuras organizativas, tecnologías,actividades, procedimientos y artefactos que se necesitan para concebir, desarrollar, implantar y mantener un producto software.
    - Las especificaciones de procesos de software definen cómo se deben hacer las actividades de ingeniería del software. No detallan técnicas concretas para las diferentes actividades, pero sí indican en qué secuencia se deben hacer las mismas
    - “El flujo de proceso sw es una representación gráfica de la secuencia en la que se ejecutan las actividades dentro de un proceso de software.Indica cómo se organizan las fases dentro de un proceso de desarrollo, pero no impone una metodología específica.”



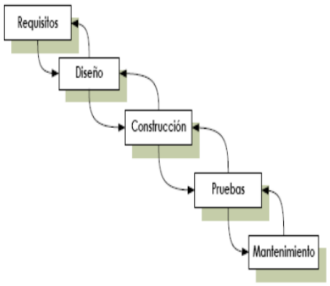




MODELOS DE CICLO DE VIDA:

1. En cascada

- Inicialmente era una serie de fases que debían realizarse en una secuencia estricta, aunque evoluciona para que una fase pueda ser corregida en la posterior.
- El sistema se entrega completo y de una sola vez al final del proyecto.
- Los entregables se asocian a fases, facilitando la gestión.
- Asume requisitos estables en todo el proyecto (++ poco realista).
- Los límites entre fases son muy rígidas.
- Se propagan los errores
- “Estructurado y fácil de gestionar. Bueno para proyectos con requisitos bien definidos y estables.”

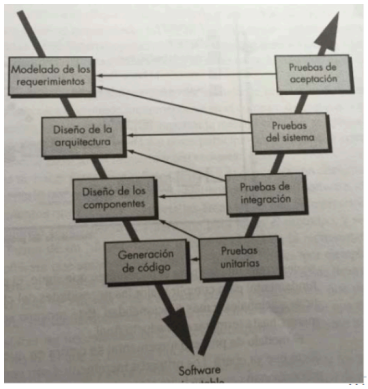


2. Modelos de ciclo de vida en V

Evolución del modelo en cascada para facilitar las actividades de verificación y validación con énfasis en la

prueba. es como el ciclo en cascada pero con una visión de cumplimiento de calidad mediante pruebas en cada etapa.
La parte izquierda de la “V” representa las fases de desarrollo previas a las actividades de prueba
Subiendo por la parte derecha, las pruebas del sistema serían las encargadas de evaluar el diseño de alto nivel, y las pruebas de aceptación serán la evaluación de que los requisitos que se tomaron reflejaban fielmente las necesidades de los usuarios.

- “Las pruebas están planificadas desde el inicio, más flexible que el modelo en cascada”
- “No permite cambios fáciles. Alto costo y tiempo por la cantidad de pruebas requeridas. . No hay software funcional hasta el final del ciclo”

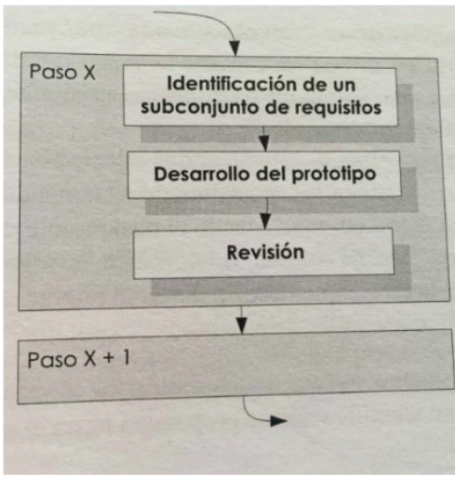


3. Modelos de ciclo de vida incremental

Útil cuando de entrada no se tiene:

- Una especificación completa del software a desarrollar, “aplicaciones con requisitos en evolución”.

- “Entrega temprana de versiones funcionales. Más flexible que el modelo en cascada. Permite la retroalimentación temprana de los usuarios.”
- Se desea entregar en etapas sucesivas versiones de software con funcionalidades que se van incrementando
- “Requiere una buena planificación para evitar inconsistencias. Mayor complejidad en la gestión del proyecto.”

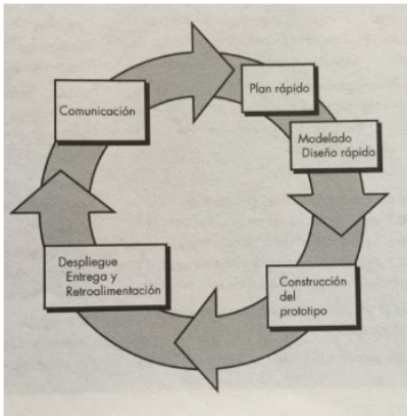


4. Modelos de ciclo de vida por prototipos

Útil cuando:

- de entrada no se tiene una especificación completa del software a desarrollar.
- Se desea entregar en etapas sucesivas versiones de software con funcionalidades que se van incrementando

- “Se basa en la creación de prototipos rápidos para validar requisitos y probar ideas antes del desarrollo completo del software.”
- El cliente quiere participar mediante sus críticas en el desarrollo. “El prototipo es una versión incompleta del sistema que se usa para obtener retroalimentación de los usuarios antes de construir el producto final.”
- “Facilita la comprensión de los requisitos antes del desarrollo completo. Involucra a los usuarios desde el inicio.”
- “Riesgo de una mala planificación del prototipo que afecte la calidad final. El cliente puede solicitar cambios constantemente, retrasando el proyecto. Mayor costo si se desarrollan múltiples prototipos innecesarios”

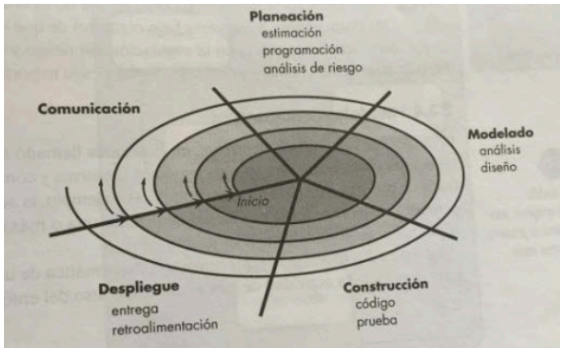


5. Modelos de ciclo de vida en espiral

Útil cuando:

- de entrada no se tiene una especificación completa del software a desarrollar.

- Se desea entregar en etapas sucesivas versiones de con funcionalidades que se van incrementando
- El cliente quiere participar mediante sus críticas en el desarrollo.
- Se realiza continuamente un control de calidad y riesgos. “Es utilizado principalmente en proyectos grandes y complejos donde hay incertidumbre o alto riesgo”
- “Alto costo y complejidad en gestión. Requiere análisis y gestión de riesgos constantes. No es óptimo para proyectos pequeños.”



Medición

- En planificación y gestión de proyectos, la medición resulta fundamental para la estimación de recursos, coste y esfuerzo, y la evaluación del personal y de la productividad.

- La medición permite, durante la ejecución de un proyecto, conocer el estado del mismo para realizar ajustes o mejoras en los procesos si fuese necesario.
- La medición de los productos y sus características hace posible la mejora de su calidad.

¿Qué se puede medir en la Ingeniería del software?.

- Artefactos: tamaño de las especificaciones, grado de su reutilización, medidas de acoplamiento, de cohesión, tamaño del código, complejidad del mismo, % de cobertura de las pruebas…
- Procesos: Tiempo, esfuerzo o número de errores de análisis, diseño o pruebas…
- Recursos: Coste de recursos software, número de personas en el equipo, etc..

Herramientas CASE

- Surgen de la necesidad de automatizar parte de las tareas a realizar.
- **Una herramienta software** es un programa de computadora que ayuda a realizar un determinado proceso o lo automatiza completamente.
- **Una herramienta CASE** (Computer Aided Software Engineering) es una herramienta software que se utiliza en una o más fases del desarrollo de un

producto software para apoyo de alguna tarea específica de Ingeniería del Software.

- Sus objetivos más comunes: desarrollar, probar, analizar, diseñar o mantener un software o su documentación.
- Facilitan la realización de actividades:
    - De gestión: control del proyecto, elaboración de informes, planificación de las pruebas, por ejemplo.
    - Técnicas: facilidades para construcción de interfaces, elaboración de prototipos, evaluación y seguimiento de los requisitos, construcción de un ejecutable, por ejemplo.

- Sin ellas sería muy difícil recopilar, utilizar y mantener el software.
- **Integración CASE** (I-CASE) es la capacidad que tiene una herramienta CASE para compartir datos y cooperar con otras herramientas.

VENTAJAS E INCONVENIENTES

- Mejoran la productividad
- Automatización de tareas repetitivas
- Aceleran el desarrollo y reducen costos
- reducen errores (mayor fiabilidad) para que se cree sw de mayor calidad
- facilitan el trabajo y la colaboración en equipo

- Permiten elaborar informes u obtener datos del desarrollo que no sería posible realizar de otro modo.
  …….

- Tienen un coste (adquisición de la herramienta/formación en su manejo/…)
- Las herramientas CASE, como todo el software en general, evolucionan muy rápidamente.
- No todas las herramientas CASE son capaces de interoperar con otras herramientas para trabajar de manera integrada
  …..

TIPOS:

1. • Upper CASE (obtención de requisitos, análisis y diseño): Ejemplo: Enterprise Architect, Visio.
2. • Lower CASE (Desarrollo,mantenimiento y pruebas): Ejemplo: Eclipse, GitHub, Selenium.
3. • I-CASE (Integradas): Ejemplo: IBM Rational Suite, Visual Studio con DevOps. “herramientas que engloban todo el proceso de desarrollo software, desde el análisis hasta la implementación.”

Chatbots de IA en Ingenieria Software

- Herramientas como ChatGPT, Perplexity, Claude y Gemini ofrecen asistencia en desarrollo.

- Complementan las herramientas CASE al proporcionar ayuda en lenguaje natural.
- Mejoran productividad, reducen errores y automatizan tareas.
- Los chatbots no reemplazan a las herramientas CASE, pero las complementan.

TEMA 2: REQUISITOS

75% - 85% de los fallos que aparecen en un software se deben a errores cometidos en las actividades de requisitos o durante el diseño del sistema

- Proyecto software: transformación de un conjunto de requisitos en un sistema informático

- El sistema fracasará si no satisface las expectativas (si la descripci´on de los mismos requisitos no es adecuada o si se desvía de lo que el cliente o los usuarios desean)

Establecer con exactitud los requisitos de un sistema es esencial para el éxito de un desarrollo de software

- **Ingeniería de requisitos:** El proceso de establecer los servicios que el cliente requiere de un sistema y las restricciones bajo las cuales opera y se desarrolla.
- **Los requisitos** en sí mismos son las descripciones de los servicios del sistema y las restricciones que se generan durante el proceso de ingeniería de requisitos. Propiedad que un software ha de tener para resolver un cierto problema

- **documento de requisitos para el sistema:** tanto el conjunto de requisitos generales que el cliente pide (debe definir sus necesidades de una manera lo suficientemente abstracta como para que una solución no esté predefinida, para que varios contratistas puedan ofertar por el contrato), como el documento más específico que escribe el contratista definiendo el sistema para que el cliente entienda y pueda validar lo que hará el software

FACTORES QUE INFLUYEN EN LOS REQUISITOS:

- Complejidad del problema a resolver: la complejidad es un factor inherente al software, y uno de los motivos fundamentales por los que esto es así radica en el hecho de que con software se resuelven, generalmente, problemas complejos
- Factores relacionados con el cliente: Muchas veces ni siquiera sabe (en términos concretos) lo que quiere
- Dificultades de comunicación entre desarrolladores y usuarios, y entre los propios miembros del equipo de desarrolladores
- Existencia de requisitos ocultos (no visibles al usuario): dependen de detalles internos al proceso de construcción del software
- Naturaleza cambiante de los requisitos

CONCEPTOS:

- **Atributo**: información complementaria sobre un requisito que se utiliza para su gestión y que se incluye en su especificación. Esta información a menudo consiste en una descripción general del requisito, su tipo según diferentes sistemas de clasificación (funcional o no funcional, del sistema, de usuario, etc.), la fuente del requisito, la historia de cambios sufrida,.....

Para que un requisito sea útil y efectivo, debe cumplir ciertas características:.

Necesario: Indispensable para el sistema.”Si se elimina, el sistema no cumpliría con sus objetivos.”
Factible: Puede implementarse con la tecnología actual.”no debe ser algo imposible de lograr con los recursos disponibles.”
Comprensible: Claro para todos los involucrados.“No debe ser ambiguo ni demasiado técnico.”
Medible: Se puede verificar si se cumple o no. “Se debe poder evaluar su cumplimiento con pruebas o métricas.”
Independiente: No depende de otros requisitos.”cada requisito debe poder evaluarse por sí solo”
Consistente: No entra en conflicto con otros requisitos.
No redundante: No se repite en el documento.
Trazable: Se puede seguir su evolución y cambios

- **Requisitos de usuario**: Declaraciones en lenguaje natural, y diagramas de los servicios que proporciona el sistema y sus restricciones operativas. Escrito para clientes.
- **Requisitos del sistema:** “Son una versión más detallada y técnica de los requisitos de usuario. Se presentan en un documento estructurado que describe exactamente qué debe implementar el software.” Un documento estructurado que describe descripciones detalladas de las funciones, servicios y restricciones operacionales del sistema. Define qué

se debe implementar, por lo que puede ser parte de un contrato entre el cliente y el contratista

ACTIVIDADES DE REQUISITOS

Se denominan **actividades de requisitos** a la obtención, análisis, especificación y validación de requisitos software. En este proceso debe definirse exactamente que se va a construir, con objeto de permitir la verificación posterior (ya no es parte de las actividades de requisitos) donde se comprueba que el sistema construido y entregado cumple lo especificado

obtención de requisitos

- Consiste en **capturar el propósito y funcionalidades del sistema desde la perspectiva del usuario;** establecer las “fronteras” del sistema
  - Acciones a llevar a cabo:
  - 1. **Determinar las fuentes de información** de las que se obtendrán los requisitos
  - 2. **Establecer las técnicas de obtención** de requisitos a utilizar
    - Entrevistas y cuestionarios
    - Prototipos (diagramas a papel con el diseño de las pantallas, versiones beta…)
    - Escenarios (ej. Casos de uso)
    - Sistemas existentes

        - Tormentas de ideas
        - …

Análisis de requisitos

- Es el proceso de estudiar las necesidades del usuario para obtener una definición detallada de los requisitos.
- Enlaza la definición de alto nivel del sistema desde la perspectiva del usuario –externa–, con el diseño del software que permitirá implementar dicho sistema –interna –
- Consiste en comprender cuáles son los comportamientos que se desea tenga el software a desarrollar, delimitando y determinando detalladamente todos los requisitos del producto a partir de lo solicitado por el cliente y los usuarios del futuro sistema.
- Escribir las tareas que deben llevarse a cabo para examinar los requisitos para delimitarlos y definir exactamente cada uno
    - Acciones a llevar a cabo:
    - 1. Detectar y resolver conflictos entre requisitos
    - 2. Delimitar el software y establecer con qué elementos externos interacciona
    - 3. Elaborar los requisitos del sistema para obtener, a partir de ellos, los requisitos del software a desarrollar

- Realizar un modelo conceptual de los requisitos (pero no solo); clasificación de los requisitos, diseño de la arquitectura del sistema y situación de requisitos en la misma (decidir qué elementos del sistema van a ser los responsables de satisfacer las demandas planteadas por los distintos requisitos.), toma de decisiones de compromiso en los casos de conflictos entre varios requisitos (negociación)....

Estos requisitos deberán ser documentados por los analistas en un proceso denominado especificación de requisitos, donde se registrará el comportamiento del sistema software que se va a construir.

Especificación de requisitos

- Es el proceso de documentar el comportamiento requerido de un sistema software, a menudo utilizando una notación de modelado u otro lenguaje de especificación.
- Describir completamente los requisitos del sistema a desarrollar
- Implica con frecuencia la construcción de un modelo (o varios) del sistema a construir desde el punto de vista de los usuarios del sistema, que incluya los requisitos obtenidos

- De manera generalizada, los requisitos se plasman en un documento denominado especificación de requisitos del software (**SRS**), un documento contractual (importante), aunque en proyectos complejos, 3 documentos:
    - Documento de definición del sistema
    - Documento de requisitos del sistema
    - Especificación de requisitos del software (SRS)

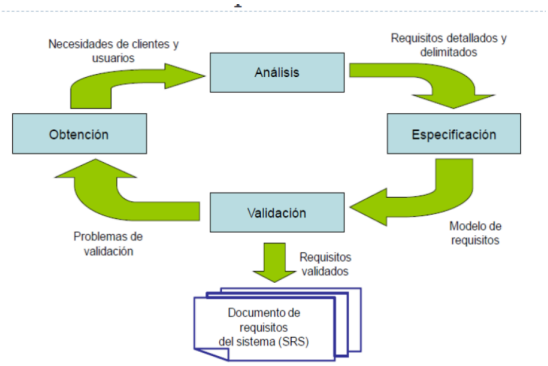
Modelado de los requisitos

- Parte de la especificación
- En un modelo conceptual cada entidad se corresponde de manera unívoca con un objeto en el mundo real
- A menudo se plasman en modelos visuales. Para crear estos modelos es necesario un lenguaje y un conjunto de reglas que establezcan cómo se han de usar los elementos del lenguaje:
    - Casos de uso
    - Diagramas de clases de análisis UML
    - Diagramas Entidad-Relación
    - Notaciones formales

Validación de requisitos

- Consiste en examinar los requisitos para asegurarse de que definen el sistema que el cliente y los usuarios desean; buscar errores y contradicciones en los

requisitos, descripciones poco claras y desviaciones de las prácticas estándar



Actores

“En el proceso de desarrollo de software, los actores son todas las personas o entidades que interactúan o tienen interés en el sistema”

- Usuarios: operan con el software
- Clientes: tienen interés en adquirir el software
- Analistas de mercado: identifican requisitos a través de potenciales clientes
- Reguladores: autoridades que establecen normativas específicas o requisitos legales
- Ingenieros del software: tienen intereses en el software en sí. Ej. en la reutilización de ciertos componentes para futuros proyectos

Características de los requisitos sw

- **Verificable**: es posible comprobar que el software da cumplimiento al requisito tal y como fue especificado. Para ello, especificarlos de manera cuantificable (evitar especificaciones ambiguas)
- Propiedades no esenciales (externas):
    - Correctos: Reflejan las necesidades reales del cliente.
    - Completos: No dejan información sin especificar.
    - No ambiguos: Claros y sin interpretaciones múltiples.
    - Consistentes: No se contradicen entre sí.
    - Priorizados: Ordenados según importancia.
    - Modificables: Permiten ajustes sin afectar todo el sistema.
    - Rastreables: Se puede seguir su origen y cambios

Tipos de requisitos

- **Funcionales**: funciones, servicios o tareas que un sw debe poder llevar a cabo. Son más fácilmente verificables, en general. LOS DE SEGURIDAD (pueden ser NF-P pero acaban siendo F siempre pq hay que implementarlo directamente…)

- **No funcionales**: aspectos técnicos que debe incluir el sistema desarrollado. En general, son más difíciles de validar. 2 tipos principales:
    - **Restricciones**: limitaciones a que se enfrentan los desarrolladores del sistema (ej. el sistema operativo del entorno del usuario, la plataforma hardware a utilizar o el perfil de los desarrolladores)
    - **Calidades**: características que importan a los clientes y usuarios, relevantes para su satisfacción con el sistema final (rendimiento, fiabilidad o disponibilidad del sistema)

-Requisitos de dominio: “Son restricciones específicas del sector o entorno donde se usará el software. Dependen del dominio de aplicación, como medicina, banca, aviación, etc.” son restricciones en el sistema desde el dominio de operación

Clasificación de requisitos no funcionales

- **Requisitos del producto**: detallan limitaciones o comportamientos exigidos al producto resultante del desarrollo. Relacionados con el rendimiento y la experiencia del usuario. Ejemplos:
    - Eficiencia: "El sistema debe responder en menos de 2 segundos."

    - Seguridad: "Los datos del usuario deben estar cifrados con AES-256."
    - Usabilidad: "El sistema debe ser accesible para personas con discapacidad."
    - Fiabilidad: "El sistema debe estar disponible el 99.9% del tiempo.", permite accesos concurrentes …
    - Ctd de memoria requerida, el historial del cliente guarda x tipo de datos (el sistema guarda el historial es FUNCIONAL)

- **Requisitos de la organización**: relacionados con las normativas de funcionamiento de la organización (que lleva a cabo el desarrollo), sus procedimientos y políticas. Ejemplos:
    - Estándares: "El sistema debe cumplir con la norma ISO 27001."
    - Plataformas: "El software debe ejecutarse en servidores Linux."
    - Herramientas de Desarrollo: "El código debe ser mantenido en GitHub con CI/CD automatizado."
    - Documentación a entregar/plazos de entrega/mantenimiento(“debe poder permitir actualizaciones sin afectar el servicio)...
    - Protocolo usado, conexión con otros equipos en LAN, logotipo de la empresa….

- **Requisitos externos:** cubren aspectos externos al sistema y a su proceso. Ejemplos:
    - Interoperabilidad con otros sistemas, requisitos legales

Implementación/problemas de requisitos no funcionales.

- Los requisitos no funcionales **pueden afectar a la arquitectura general de un sistema** en lugar de a los componentes individuales. Por ejemplo, para garantizar que se cumplan los requisitos de rendimiento, es posible que deba organizar el sistema para minimizar las comunicaciones entre los componentes.
- **Un solo requisito no funcional, como un requisito de seguridad, puede generar una serie de requisitos funcionales relacionados que definen los servicios del sistema que se requieren.**
- T**ambién puede generar requisitos que restringen los requisitos existentes**

Imprecisión en requisitos.

- Los problemas surgen cuando los requisitos no se establecen con precisión.
- Los desarrolladores y usuarios pueden interpretar los requisitos ambiguos de diferentes maneras.

ej: Considere el término "búsqueda" en el requisito 1. Intención del usuario: busque el nombre de un paciente en todas las citas en todas las clínicas; Interpretación del desarrollador: busque el nombre de un paciente en una clínica individual. El usuario elige la clínica y luego busca.

Pautas para la escritura de requisitos

1. Inventar un formato estándar y usarlo para todos los requisitos.Usar el lenguaje de una manera consistente.
2. Distinguir entre requisitos obligatorios, de requisitos deseables.
3. Utilizar el resaltado de texto para identificar las partes clave del requisito.
4. Evitar el uso de la jerga informática.
5. Incluir una explicación (razón) de por qué es necesario un requisito

Problemas con el lenguaje natural.

- Falta de claridad: La precisión es difícil sin hacer que el documento sea difícil de leer.
- Confusión de requisitos: Los requisitos funcionales y no funcionales tienden a confundirse.
- Amalgamación de requisitos: Varios requisitos diferentes se pueden expresar juntos

Gestión del proceso de requisitos

Seguimiento + Métricas + Herramientas de gestión

“La gestión del proceso de requisitos es un conjunto de prácticas que permiten organizar, rastrear y controlar los requisitos a lo largo del desarrollo del software. Esto incluye el seguimiento de cambios, métricas y herramientas de gestión, con el objetivo de evitar inconsistencias y asegurar que el software cumpla con lo solicitado.”

- **Seguimiento**: más improbable que el sistema final tenga inconsistencias por culpa de cambios realizados sin un control exhaustivo. “Permite controlar cambios y evitar inconsistencias en el desarrollo. Evita errores por modificaciones sin un control adecuado.”
- Los requisitos evolucionan iterativamente hasta alcanzar un nivel de calidad y detalle suficiente que permita iniciar los trabajos de diseño de las funcionalidades que les dan soporte.
- Para facilitar la comprensión completa de todos los aspectos asociados con los requisitos han de definirse líneas base. Una **línea base** es un **conjunto de requisitos que deberá contener una entrega del producto**. “Son puntos de referencia que

permiten mantener el control sobre los cambios. Una vez que una línea base es aprobada, cualquier modificación debe ser controlada mediante un proceso formal de gestión de cambios.”

Matriz de seguimiento:

- tabla donde se relacionan dos documentos, los cuales pertenecen probablemente a etapas distintas del desarrollo.
- Su utilización más frecuente es seguir la pista de los requisitos a lo largo de todo el desarrollo, fundamentalmente en diseño, plan de pruebas y casos de prueba
- Hace corresponder los requisitos con los componentes software que los implementarán, y especialmente con los casos de prueba
- “Se usa para rastrear requisitos a lo largo de todo el ciclo de vida del software (desde la especificación hasta la prueba final). Permite verificar que cada requisito esté correctamente implementado y probado”

	Requisito 1	Requisito 2	Requisito 3	...	Requisito n
Caso prueba 1		•	•		
Caso prueba 2	•	•			•
Caso prueba 3	•				
Caso prueba 4			•		
Caso prueba 5		•			•
Caso prueba 6	•	•			

Métricas de requisitos

- Es posible tomar medidas de los requisitos software que indiquen el alcance del proyecto, su crecimiento potencial, su estabilidad y progreso.
- Las medidas de requisitos son únicas: caracterizan el espacio del problema. Otras medidas del desarrollo caracterizan el espacio de la solución.

Ejemplos:

- Número de requisitos nuevos (o modificados) por mes. Refleja la estabilidad o inestabilidad del desarrollo.
- Indicador de cobertura: porcentaje de requisitos actualmente incluidos en alguna línea base que han sido soportados por un componente.
- Número de requisitos derivados a partir de cada requisito inicial: muestra la explosión de requisitos consecuencia de una incorrecta especificación o de su descripción a un nivel de detalle incorrecto

Gestión de cambios de requisitos.

- Decidir si un cambio de requisitos debe ser aceptado.
- **Análisis de problemas y especificación de cambios**: Durante esta etapa, se analiza el problema o la propuesta de cambio para verificar que sea válida.

- Este análisis se retroalimenta al solicitante de cambios, que puede responder con una propuesta de cambio de requisitos más específica o decidir retirar la solicitud.
- **Análisis de cambio y costeo:** “Se evalúa el efecto del cambio en el diseño, implementación y pruebas. Se analiza la trazabilidad para ver qué partes del sistema se verán afectadas y se calcula cuánto costará implementar el cambio.”
    - **El efecto del cambio propuesto se evalúa** utilizando la información de trazabilidad y el conocimiento general de los requisitos del sistema. Una vez que se completa este análisis, **se toma la decisión de continuar o no con el cambio de requisitos.**

- **Cambio de implementación**: El documento de requerimientos y, cuando sea necesario, el diseño e implementación del sistema, son modificados. Lo ideal es que el documento se organice para que los cambios se puedan implementar fácilmente.

Herramientas de requisitos

- El tamaño y complejidad de los desarrollos ha convertido el uso de herramientas en algo esencial
- Las herramientas de gestión de requisitos:

    - Facilitan un modo de seguimiento de los cambios que se producen durante el ciclo de vida del proyecto
    - Permiten automatizar algunas de las tareas relacionadas con los cambios y el seguimiento de requisitos (ej. Identificación, medición…)
    - Mejoran la productividad y la calidad en el desarrollo

TEMA 2.2: MODELADO DE SISTEMAS

- El modelado de sistemas es el proceso de desarrollar modelos abstractos de un sistema, con cada modelo presentando una vista o **perspectiva** diferente de ese sistema: “representan una abstracción de un sistema”
- El modelado del sistema significa representar un sistema utilizando algún tipo de notación gráfica, que casi siempre se basa en anotaciones en el lenguaje de modelado unificado (**UML**).
- El modelado del sistema ayuda al analista a comprender la funcionalidad del sistema y los

modelos se utilizan para comunicarse con los clientes.

Modelos de sistemas existentes y planificados

- Los modelos de **sistemas existentes** se utilizan durante la ingeniería de requisitos. Ayudan a aclarar lo que hace el sistema existente y pueden usarse como base para discutir sus fortalezas y debilidades. Estos conducen a requisitos para el nuevo sistema.
- Los modelos de un **nuevo sistema** se utilizan durante la ingeniería de requisitos para ayudar a explicar los requisitos propuestos a otras partes interesadas en el sistema. Los ingenieros usan estos modelos para discutir propuestas de diseño y documentar el sistema para su implementación.

Uso de modelos gráficos

- Como un medio para facilitar la discusión sobre un sistema existente o propuesto
    - Los modelos incompletos e incorrectos están bien, ya que su función es apoyar la discusión.
- Como una forma de documentar un sistema existente o que se va a desarrollar
    - Los modelos deben ser una representación precisa del sistema pero no necesitan estar completos.

- Como una descripción detallada del sistema que se puede utilizar para generar una implementación del sistema
    - Los modelos tienen que ser correctos y completos

Perspectivas del Sistema

Las perspectivas del sistema definen diferentes enfoques para analizar y modelar un sistema. Ayudan a entender cómo funciona el sistema desde distintos puntos de vista

- **Perspectiva externa**: modela el contexto o el entorno del sistema.
- **Perspectiva de interacción:** modela las interacciones entre un sistema y su entorno, o entre los componentes de un sistema.
- **Perspectiva estructural:** modela la organización de un sistema o la estructura de los datos que procesa el sistema.
- **Perspectiva de comportamiento:** modela el comportamiento dinámico del sistema y cómo responde a los eventos.

Tipos de diagramas UML

(representar modelos de sistemas)

- **Diagramas de actividades**: muestran las actividades involucradas en un proceso o en el procesamiento de datos.
- **Diagramas de casos de uso**: muestran las interacciones entre un sistema y su entorno.
- **Diagramas de secuencia:** muestran las interacciones entre los actores y el sistema y entre los componentes del sistema.
- **Diagramas de clases:** muestran las clases de objetos en el sistema y las asociaciones entre estas clases.
- **Diagramas de estados:** muestran cómo reacciona el sistema a eventos internos y externos

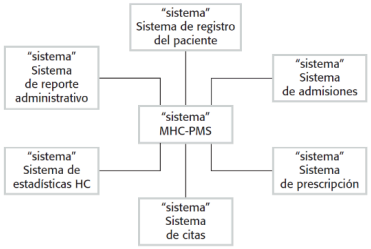
Modelos de contexto

- Los modelos de contexto se utilizan para ilustrar el contexto operativo de un sistema: muestran lo que se encuentra fuera de los **límites del sistema. (“perspectiva externa”).** muestran el sistema y su relación con otros sistemas.

Limites del sistema

- Los límites del sistema se establecen para definir lo que está dentro y lo que está fuera del sistema.

- Muestran otros sistemas que se utilizan o dependen del sistema que se está desarrollando.
- La posición del límite del sistema tiene un profundo efecto en los requisitos del sistema.
- Las **preocupaciones sociales** pueden afectar la decisión sobre dónde ubicar los límites del sistema.
    - “La ubicación de los límites del sistema no solo es una decisión técnica, sino que también puede estar influenciada por factores sociales, políticos y organizacionales, como regulaciones, privacidad, impacto en la comunidad o expectativas de los usuarios.”

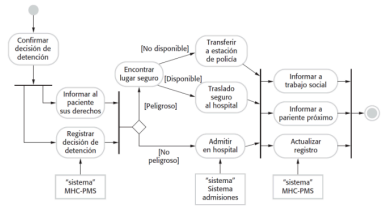


Modelo de Contexto: componentes de un sistema de cuidado de salud mental

modelos de proceso

- (Los modelos de contexto simplemente muestran los otros sistemas en el entorno, no cómo se utiliza el sistema que se está desarrollando en ese entorno.)
- **Los modelos de proceso revelan cómo se utiliza el sistema que se está desarrollando en procesos comerciales más amplios/”en su entorno”.**

- Los **diagramas de actividad UML** pueden usarse para definir modelos de procesos de negocio



Modelo de Actividad: proceso de detención involuntaria

Modelos de interacción

- Modelar la interacción del usuario es importante ya que ayuda a identificar los requisitos del usuario.
- El modelo de interacción de sistema a sistema resalta los problemas de comunicación que pueden surgir.
- La interacción de componentes de modelado nos ayuda a comprender si una estructura de sistema propuesta es probable que brinde el rendimiento y la confiabilidad requeridos del sistema.
- Se pueden usar **diagramas de casos de uso y diagramas de secuencia** para el modelado de interacción.

Diagrama de secuencias

- Los diagramas de secuencia son parte del UML y se usan para modelar las **interacciones entre los actores y los objetos dentro de un sistema.**

- Un diagrama de secuencia muestra la **secuencia de interacciones que tienen lugar durante un caso de uso particular** o una instancia de caso de uso.
- Los objetos y actores involucrados se enumeran en la parte superior del diagrama, con una línea de puntos dibujada verticalmente a partir de estos. Las interacciones entre objetos se indican mediante flechas anotadas.

Diagrama de secuencia para "ver información del paciente

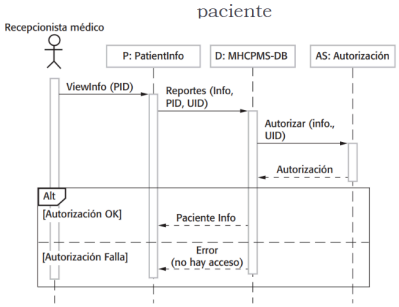
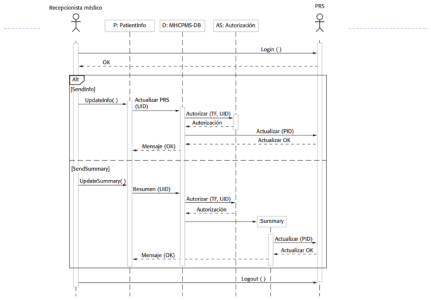


Diagrama de secuencia para tranferencia de datos



Modelos de caso de uso

- Los casos de uso se desarrollaron originalmente para la obtención de requisitos y se incorporaron al UML.
- Cada caso de uso representa una tarea discreta que involucra interacción externa con un sistema.
- **Los actores en un caso de uso pueden ser personas u otros sistemas.**
- Representado esquemáticamente para proporcionar una visión general del caso de uso y en una forma textual más detallada:

MHC-PMS: Transferencia de datos	
Actores	Recepcionista médico, sistema de registros de paciente (PRS).
Descripción	Un recepcionista puede transferir datos del MHC-PMS a una base de datos general de registro de pacientes, mantenida por una autoridad sanitaria. La información transferida puede ser información personal actualizada (dirección, número telefónico, etc.) o un resumen del diagnóstico y tratamiento del paciente.
Datos	Información personal del paciente, resumen de tratamiento.
Estímulo	Comando de usuario emitido por recepcionista médico.
Respuesta	Confirmación de que el PRS se actualizó.
Comentarios	El recepcionista debe tener permisos de seguridad adecuados para acceder a la informa del paciente y al PRS.

Modelos estructurales

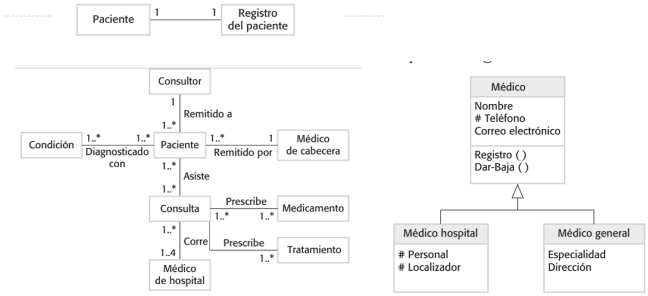
- Los modelos estructurales de software muestran la organización de un sistema en términos de los componentes que conforman ese sistema y sus relaciones.
- Los modelos estructurales pueden ser modelos estáticos, que muestran la estructura del diseño del sistema, o modelos dinámicos, que muestran la organización del sistema cuando se está ejecutando.
- Pueden crearse los modelos estructurales de un sistema cuando se debate y diseña la arquitectura del sistema

Diagramas de clase (estructural)

- Los diagramas de clase se utilizan al desarrollar un modelo de sistema orientado a objetos para mostrar las clases en un sistema y las asociaciones entre estas clases.
- Una clase puede considerarse como una definición general de un tipo de objeto del sistema.

- Una asociación es un enlace entre clases que indica que existe alguna relación entre estas clases.
- Cuando desarrolla modelos durante las primeras etapas del proceso de ingeniería de software, los objetos representan algo en el mundo real, como un paciente, una receta...

Asociacion de clases con UML



Generalización (d.clase)

- La generalización es una técnica cotidiana que utilizamos para gestionar la complejidad.
- En lugar de aprender las características detalladas de cada entidad que experimentamos, colocamos estas entidades en clases más generales (animales, automóviles, casas, etc.) y aprendemos las características de estas clases.
- Esto nos permite inferir que diferentes miembros de estas clases tienen algunas características comunes.

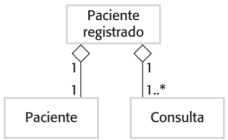
- En los sistemas de modelado, a menudo es útil examinar las clases en un sistema para ver si hay

margen para la generalización. Si se proponen cambios, entonces no tiene que mirar todas las clases en el sistema para ver si se ven afectados por el cambio.

- En lenguajes orientados a objetos, como Java, la generalización se implementa utilizando los mecanismos de **herencia** de clase integrados en el lenguaje.
- En una generalización, los atributos y operaciones asociados con las clases de nivel superior también se asocian con las clases de nivel inferior. Las clases de nivel inferior son subclases que heredan los atributos y operaciones de sus superclases. Estas clases de nivel inferior luego agregan atributos y operaciones más específicos.

Modelo de agregación (estructural)

- Un modelo de agregación muestra cómo las clases que son colecciones se componen de otras clases.
- Los modelos de agregación son similares a la parte de la relación en los modelos de datos semánticos.



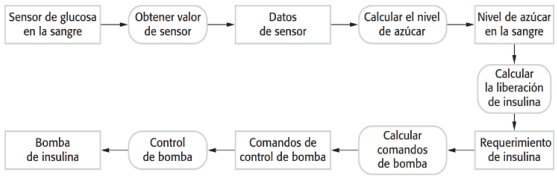
Modelos de conducta/comportamiento

- Los modelos de comportamiento son modelos del **comportamiento dinámico de un sistema mientras se ejecuta**. Muestran lo que sucede o lo que se supone que sucede cuando un sistema responde a un estímulo de su entorno.
- Puede pensar que estos estímulos son de dos tipos:
    - **Datos:** Llegan algunos datos que deben ser procesados por el sistema.
    - **Eventos**: Se produce algún evento que desencadena el procesamiento del sistema. Los eventos pueden tener datos asociados, aunque este no es siempre el caso

Modelos basados en datos (conducta)

- Muchos sistemas comerciales son sistemas de procesamiento de datos que se basan principalmente en datos.
- Son controlados por la entrada de datos al sistema, con relativamente poco procesamiento de eventos externos.
- Los modelos basados en datos **muestran la secuencia de acciones involucradas en el procesamiento de datos de entrada y la generación de una salida asociada.**

- Son particularmente **útiles durante el análisis de requisitos, ya que pueden usarse para mostrar el procesamiento de extremo a extremo en un sistema**
- “Diagramas de flujo de datos (DFD),Modelos entidad-relación (ER),Diagramas de actividad UML”



Modelos basados en eventos (conducta)

- Los sistemas en tiempo real a menudo están controlados por eventos, con un procesamiento mínimo de datos. Por ejemplo, un sistema de conmutación de teléfono fijo responde a eventos tales como "receptor descolgado" generando un tono de marcado.
- El modelado basado en eventos muestra cómo un sistema responde a eventos externos e internos.
- Se basa en el supuesto de que un sistema tiene un número finito de estados y que los eventos (estímulos) pueden causar una transición de un estado a otro.
- Diagramas de Estados UML, Diagramas de Secuencia UML

Modelos de máquina de estado (eventos)

- Modelan el comportamiento del sistema en respuesta a eventos externos e internos.
- Muestran las respuestas del sistema a los estímulos, por lo que a menudo se usan para modelar sistemas en tiempo real.
- Los modelos de máquina de estado muestran los estados del sistema como nodos y los eventos como arcos entre estos nodos. Cuando ocurre un evento, el sistema se mueve de un estado a otro.
- Los gráficos de estado son una parte integral del UML y se utilizan para representar modelos de máquinas de estado

Ejemplo de modelo de máquina de estado

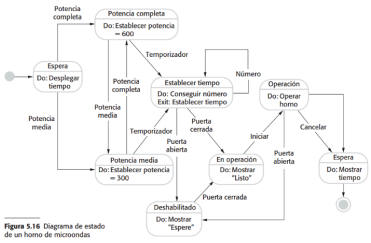


Figura 5.16 Diagrama de estado de un horno de microondas

Ingeniería basada en modelos (MDE)

- La ingeniería basada en modelos (MDE) es un enfoque para el **desarrollo de software** donde **los modelos, en lugar de los programas, son los principales resultados del proceso de desarrollo.**
- Los programas que se ejecutan en una plataforma de hardware / software se generan automáticamente a partir de los modelos.
- Los defensores de MDE argumentan que esto **eleva el nivel de abstracción** en la ingeniería de software para que los ingenieros ya no tengan que preocuparse por los detalles del lenguaje de programación o los detalles de las plataformas de ejecución.

VENTAJAS/DESVENTAJAS DE LA MDE:

La ingeniería basada en modelos aún se encuentra en una etapa temprana de desarrollo, y no está claro si tendrá un efecto significativo en la práctica de la ingeniería de software.

Pros:

- Permite que los sistemas sean considerados en niveles más altos de abstracción
- Generar código automáticamente significa que es más barato adaptar los sistemas a las nuevas plataformas.

Contras:

- Modelos para la abstracción y no necesariamente adecuados para la implementación.
- Los ahorros de generar código pueden ser compensados por los **costos de desarrollar traductores** para nuevas plataformas

Arquitectura basada en modelos (MDA)

- La arquitectura basada en modelos (MDA) fue el precursor de la ingeniería más general basada en modelos
- MDA es un enfoque centrado en el modelo para el diseño y la implementación de software que utiliza un subconjunto de modelos UML para describir un sistema.
- **Se crean modelos en diferentes niveles de abstracción.**
- Desde un modelo independiente de plataforma de alto nivel, es posible, en principio, generar un programa de trabajo sin intervención manual

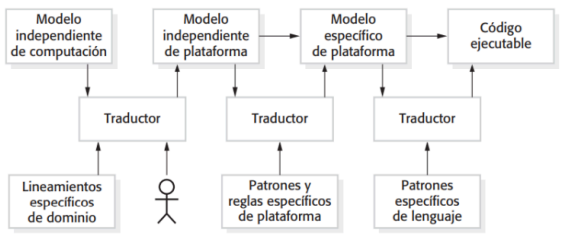
Tipos de modelos MDA

- Un modelo independiente de computación (CIM)
    - Estos modelan las abstracciones de dominio importantes utilizadas en un sistema. Los CIM a veces se denominan modelos de dominio.” Define los requisitos sin detalles técnicos. El CIM

representa el sistema desde el punto de vista del negocio y los requisitos, sin considerar aspectos técnicos ni de implementación.”

- Un modelo independiente de plataforma (PIM)
    - Estos modelan el funcionamiento del sistema sin referencia a su implementación. El PIM generalmente se describe utilizando modelos UML que muestran la estructura del sistema estático y cómo responde a eventos externos e internos. “ El PIM es un modelo que representa la estructura y lógica del sistema sin depender de una plataforma/tecnología específica.”
- Modelos específicos de plataforma (PSM)
    - Estas son transformaciones del PIM con un PSM separado para cada plataforma de aplicación. En principio, puede haber capas de PSM, y cada capa agrega algunos detalles específicos de la plataforma. “Traduce el modelo PIM a un modelo dependiente de una plataforma concreta”

Transformaciones MDA



MDA y métodos ágiles

- Los desarrolladores de MDA afirman que está destinado a admitir un enfoque iterativo para el desarrollo y, por lo tanto, puede usarse dentro de métodos ágiles “(Un método ágil es un enfoque de desarrollo de software que prioriza la flexibilidad, la entrega iterativa, la colaboración con el cliente y la adaptación a los cambios)”
- La noción de modelado inicial extenso contradice las ideas fundamentales en el manifiesto ágil y pocos desarrolladores ágiles se sienten cómodos con la ingeniería basada en modelos (“manif agil promueve la entrega rápida sin una gran planificación inicial. Los desarrolladores ágiles suelen preferir iteraciones rápidas en lugar de construir modelos detallados antes de escribir código.”)
- Si las transformaciones pueden ser completamente automatizadas y un programa completo generado a partir de un PIM, entonces, en principio, MDA podría usarse en un proceso de desarrollo ágil ya que no se requeriría una codificación separada

UML ejecutable

- La noción fundamental detrás de la ingeniería basada en modelos es que la transformación completamente automatizada de modelos a código debería ser posible.
- Esto es posible usando un subconjunto de UML 2, llamado UML ejecutable o xUML.
- Para crear un subconjunto ejecutable de UML, el número de tipos de modelo se ha reducido drásticamente a estos 3 tipos clave:
    - **Modelos de dominio** que identifican los principales principios y relaciones en un sistema. Se definen utilizando diagramas de clase UML e incluyen objetos, atributos y asociaciones.
    - **Modelos de clase** en los que se definen las clases, junto con sus atributos y operaciones.
    - **Modelos de estado** en los que se asocia un diagrama de estado con cada clase y se utiliza para describir el ciclo de vida de la clase.
- El comportamiento dinámico del sistema puede especificarse declarativamente usando el lenguaje de restricción de objetos (OCL), o puede expresarse usando el lenguaje de acción de UML

EJERCICIOS

Enunciado 1:

Clasifique los siguientes requisitos software en funcionales o no funcionales. Si se trata de

requisitos no funcionales, diga a cuál de las tres categorías (de producto, de la organización o externos) pertenece:

1. La interfaz debe seguir la normativa de colores e imagen corporativa de la empresa. **NF-O**
2. La web debe seguir la normativa internacional en las áreas de protocolos, contenidos, herramientas, formatos y lenguajes relacionados con la accesibilidad a Internet. NF-E
3. El área de trabajo debe poder ampliarse o reducirse mediante los correspondientes botones de acceso directo en la barra de herramientas. F
4. La aplicación permitirá consultar las facturas por diversos criterios, tales como facturas de pronto vencimiento, por cliente, por importe, etc. F
5. El acceso a la información personal sobre clientes no podrá ser consultada por los usuarios, excepto aquellos a los que les hayan sido concedido privilegios especiales para ello. NF-P
6. El programa cliente que realiza las búsquedas en la base de datos debe poder estar seguro de que la respuesta del sistema no ha sido falsificada o alterada. NF-P
7. El lenguaje de programación a utilizar será obligatoriamente Java. NF-O
8. La aplicación en desarrollo sigue la filosofía del software libre y debe por tanto ponerse a

disposición de la comunidad su código fuente. NF-O
9. La información de enrutamiento debe protegerse contra modificaciones no autorizadas. NF-P
10. Deben almacenarse todas las facturas emitidas por el sistema para su posterior consulta. F
11. El sistema incluirá algún tipo de comprobación de la identidad para las operaciones
Sensibles. F
12. El sistema debe permitir la realización periódica de copias de seguridad para toda la
información de clientes y ventas. F

Requisitos no funcionales: 1 (organización), 2 (externo), 3 (producto), 5 (producto), 6 (producto), 7 (organización), 8 (organización), 9 (producto) y 12 (producto).
Requisitos funcionales: 4, 10, 11 y 12.
Tal y como comentamos en clase los requisitos no funcionales relativos a aspectos de seguridad inicialmente se consideran como No Funcionales, pero conforme avanza el desarrollo del proyecto pasan a ser Funcionales por la interacción necesaria en su implementación. Por lo tanto, el requisito 11 se considera Funcional.

Elaborar la lista de requisitos del sistema informático que necesita PhoneMad para gestionar
las ventas, clasificándolos como requisitos funcionales o no funcionales, y los no funcionales
según su categoría (P,O,E). Identificar los actores que intervienen en el sistema software.

La empresa PhoneMad se dedica desde hace años a vender teléfonos móviles, pero con la
aparición de teléfonos de nueva generación en los que se puede instalar aplicaciones, como el
iPhone de Apple o Android de Google y otros smartphones, PhoneMad ha decidido crear una aplicación web para vender software para estos y otros dispositivos móviles como el iPad o el
Samsung Galaxy. Es importante que el sistema desarrollado cumpla el nivel A (como mínimo) de las pautas de accesibilidad web, ya que se quiere orientar la campaña de marketing a un cierto sector de mercado identificado como sensible a estas cuestiones.
Se ha pensado en un sistema con acceso web, que funcione correctamente en los navegadores más

populares (IExplore, Firefox, Chrome y Safari) por lo que además de cumplir los estándares de desarrollo web del W3C (uso de XHTML para el contenido y CSS para la apariencia, etc.) tendrá que comprobarse que estos navegadores visualizan correctamente toda la información y permiten llevar a cabo todas las funcionalidades correctamente.
Se desea vender software agrupado por categorías (aplicaciones sociales, mapas, juegos, utilidades GPS, gestores de documentos, ...), por lo que la aplicación de PhoneMad deberá permitir añadir categorías y programas concretos dentro de la categoría que corresponda (a quienes introduzcan los datos desde PhoneMad), y lógicamente al usuario/comprador deberá facilitársele la navegación, browsing, búsqueda por diferentes criterios, etc. para finalmente seleccionar (y eventualmente comprar). El software se agrupa por el Sistema Operativo (S.O.) que lo soporta, y así se debe poder visualizar.
No todos los teléfonos pueden ejecutar los mismos programas, por lo que se podrán hacer consultas sobre los programas que soporta un determinado teléfono. Cada software que vendemos puede tener varias versiones disponibles, ya sean para distintos S.O., o distintas

versiones para el mismo S.O. En cualquier caso, cada versión aportará una serie de mejoras a la anterior, que deben poder mostrarse a los usuarios.
Los usuarios del sistema hacen pedidos, para lo que deben darse de alta e identificarse al inicio de la sesión. La posibilidad de realizar pedidos se garantiza en un máximo de 24h tras haber sido dado de alta en el sistema. Cada pedido puede constar de uno o varios productos. Además de hacer pedidos, los usuarios registrados pueden incluir opiniones sobre los artículos que se venden. Es necesario que quede registro del nombre y el email de los usuarios registrados, así como de todas sus consultas, las aplicaciones en las que tiene interés, y los comentarios realizados. Esto es importante porque todo dato personal que se registre o almacene en la aplicación deberá estar amparado por la Ley Orgánica de Protección de Datos española.
Una vez en funcionamiento, la aplicación permitirá accesos concurrentes del orden de cientos de usuarios y tendrá una disponibilidad total, 24 horas al día, 7 días a la semana, 365 días al año.
Deberá almacenar los datos en una base de datos Oracle (lo cual viene determinado por contratos previos con dicha empresa), residiendo dichos datos, de manera obligatoria, en

servidores externos a las instalaciones de PhoneMad por cuestiones de seguridad y para ahorrar en costes de mantenimiento fijo