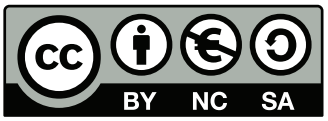
Fundamentos de Ingeniería del Software

LibreIM

Doble Grado de Informática y Matemáticas
Universidad de Granada

libreim.github.io/apuntesDGIIM



Fundamentos de Ingeniería del Software

LibreIM

Doble Grado de Informática y Matemáticas
Universidad de Granada

libreim.github.io/apuntesDGIIM

Índice

1. FUNDAMENTOS DE INGENIERÍA DEL SOFTWARE **2**

1.1. Tema 1. Introducción a la ingeniería del software 2

1.1.1. 1.1. El producto software 2

1.1.2. 1.2 El concepto de ingeniería del software 3

1.2. Tema 2. Ingeniería de requisitos 4

1.2.1. 2.1 Instrodución al modelado de requisitos 4

1.2.2. 2.2 Obtención de requisitos 5

1.2.3. 2.3 Modelado de casos de uso 7

1.2.4. 2.4 Análisis y especificación de requisitos 9

1.3. Tema 3. Diseño e implementación 11

1.3.1. 3.1 Fundamentos del diseño software 11

1.3.2. 3.2 Diseño de la arquitectura 13

1.3.3. 3.4 Diseño de la estructura de objetos 17

1 FUNDAMENTOS DE INGENIERÍA DEL SOFTWARE

1.1 Tema 1. Introducción a la ingeniería del software

1.1.1 1.1. El producto software

Naturaleza del software Producto/vehículo para distribuir un producto.

Como producto, proporciona potencial de cómputo y es un transformador de información.

Como vehículo para la distrubución de un producto, puede actuar como: - Base para el control de la computadora (sistemas opetativos). - Base para la comunicación de información (redes). - Base para la creación y control de otros programas (herramientas y ambientes de software).

Definición de software El software es: 1. Instrucciones (programas) que cuando se ejecutan proporcionan las funciones y características buscadas. 2. Estructuras de datos que permiten a los programas manipular la información adecuadamente. 3. Información en papel o en forma virtual (documentación) que describe la operación y uso de los programas.

Características del software

- El software no se fabrica en el sentido clásico. Es intangible.
- El software no se desgasta, pero se deteriora.
- La mayoría del software se construye para uso individualizado.

Tipos y dominios de aplicación del software

- Software genérico. Sistema autónomo producido por una organización de desarrollo y vendido en el mercado abierto a cualquier cliente que quiera comprarlo.
- Software a medida. Sistema desarrollado por una empresa especialmente para un cliente en particular.

Los dominios de aplicación del software son:

- Software de sistemas. Conjunto de programas que proporcionan servicios a otros programas.
- Software de aplicación. Programas que resuelven una necesidad específica de negocios.
- Software de ingeniería y ciencia. Implementa algoritmos “devoradores” de números.
- Software empotrado. Reside dentro de un producto o sistema que implementa y controla características y funciones para el usuario final y para el sistema en sí.
- Software de gestión. Proporciona una capacidad específica para uso de muchos consumidores diferentes.
- Aplicaciones Web. Software centrado en redes que agrupa una amplia gama de aplicaciones.
- Software de inteligencia artificial. Implementa algoritmos no numéricos para resolver problemas complejos difíciles de tratar computacionalmente o con análisis directo.

1.1.2 1.2 El concepto de ingeniería del software

Establecimiento y uso de principios fundamentales de ingeniería con objeto de desarrollar en forma económica software que sea fiable y trabaje con eficiencia en máquinas reales.

Aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada y requerida para el desarrollo, operación y mantenimiento de los programas.

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

Terminología usada en ingeniería del software

- Sistema. Conjunto de elementos relacionados entre sí y con el medio, que forma una unidad o un todo organizativo.
- Sistema basado en computadora. Conjunto o disposición de elementos organizados para cumplir una meta predefinida al procesar información.
- Sistema software. Conjunto de piezas o elementos software relacionados entre si y organizados en subsistemas.
- Modelos. Representación de un sistema en un determinado lenguaje.
- Principios. Elementos adquiridos mediante el conocimiento, que definen las características que debe poseer un modelo para ser una representación adecuada del sistema.
- Herramientas. Instrumentos que permiten la representación de modelos.
- Técnicas. Modo de utilización de las herramientas.
- Heurística. Conjunto de reglas empíricas, que al ser aplicadas producen modelos que se adecuan a los principios.
- Métodos. Secuencia de actividades para la obtención de producto (modelo), que describen cómo usar las herramientas y las heurísticas.

El proceso de desarrollo del software Etapas principales: - Especificación de requisitos. Análisis del problema a resolver. Documento en el que se indica la funcionalidad del software y las restricciones sobre su operación. - Diseño. Búsqueda de una solución. Descripción de los componentes, sus relaciones y funciones. - Implementación. Traducción del diseño a un lenguaje de programación entendible por una máquina. - Validación. Revisiones de todo lo que se ha obtenido junto con la prueba del código. - Mantenimiento y evolución. Reparación de fallos y adaptación a nuevos entornos. - Planificación. Estimación del tiempo y de los costes del desarrollo.

Modelo en cascada. Separación clara entre etapas. Puede producirse retroalimentación. Genera una documentación muy completa que facilita el mantenimiento posterior.

Desarrollo incremental. Se desarrolla una implementación inicial y se refina por medio de sucesivas versiones. Admite feedback del cliente desde una fase temprana. Se genera poca documentación (el proceso no es visible), lo que dificulta el mantenimiento. Degradación de la estructura del sistema. Es preferible al modelo en cascada para proyectos pequeños.

Modelos orientados a la reutilización. Dada una especificación de requisitos, se buscan los componentes más adecuados que implementen esa especificación. Se analizan los requisitos usando la información de

los componentes encontrados y se realiza un diseño a partir de estos. Disminuye los costes y el tiempo de desarrollo. La principal desventaja es la pérdida de control sobre la evolución del sistema cuando las versiones de los componentes no están bajo el control de la organización que los usa.

Modelos orientados al cambio Para evitar los costes del rehacer: - Evitar el cambio. En el proceso de desarrollo se incluyen actividades que anticipan los posibles cambios antes de que se requieran. - Tolerancia al cambio. El proceso de desarrollo se diseña de manera que lso cambios se ajusten con un coste relativamente bajo. Formas de enfrentar el cambio: - Prototipo desechable. - Entrega incremental.

Prototipo deshechable. Versión inicial del software que seutiliza para demostrar los conceptos, probar las opciones de diseño y comprender mejor el problema y sus posibles soluciones.

Ventajas: - Obtener nuevas ideas para requisitos y describir fortalezas y debilidades del software. - Comprobar la factibilidad del diseño propuesto. - Participación del usuario final en el desarrollo de interfaces gráficas. Problemas: - Que el prototipo no se utilice de la misma forma que el sistema final. - Que el tiempo de capacitación durante la evaluación del prototipo sea insuficiente. - La entrega de un prototipo como versión final del software.

Entrega incremental. Se trata de un prototipo evolutivo.

Proceso unificado. Es un modelo híbrido que apoya la creación de prototipos y la entrega incremental. Sigue cuatro fases: - Concepción. Establecer un caso empresarial para el sistema. - Elaboración. Desarrollar la comprensión del problema, establecer un marco conceptual arquitectónico para el sistema, diseñar el plan del proyecto e identificar los riesgos clave. - Construcción. Incluye diseño, programación, integración de las partes del sistema que se han desarrollado en paralelo y pruebas del sistema. - Transición. Cambio del sistema desde los desarrolladores al usuario. Poner el sistema a funcionar en una ambiente real.

1.2 Tema 2. Ingeniería de requisitos

1.2.1 2.1 Instroducción al modelado de requisitos

Requisito: propiedad que un software desarrollado o adaptado debe tener para resolver un problema concreto.

Ingeniería de requisitos son todas las actividades relacionadas con: - identificar y documentar las necesidades del cliente. - Analizar la viabilidad de las necesidades. - Negociar una solución razonable. - Crear un documento que describa un software que satisfaga las necesidades. - Analizar y validar el documento. - Controlar la evolución de las necesidades.

Factores a tener en cuenta: - La complejidad del problema a resolver. - La forma de identificar los requisitos por parte del cliente. - Dificultades de comunicación entre desarrolladores y usuario. - Dificultades de comunicación entre los miembros del equipo de desarrolladores. - Requisitos que no se pueden obtener del cliente y los usuarios. - Naturaliza cambiante de los requisitos.

Dividimos el proceso en 4 fases.

- Obtención de requisitos. Capturar el propósito y funcionalidades del sistema desde la perspectiva

del usuario para delimitar las fronteras del sistema y elaborar un glosario de términos. Es un proceso difícil apoyado por entrevistas, casos de uso, prototipado y análisis etnográfico. Genera los documentos de las entrevistas, una lsita estructurada de reuqisitos, diagramas de casos de uso, plantillas y diagramas de actividad.

- Análisis de requisitos. Proceso de estudiar las necesidades del usuario para obtener una definición detallada de los requisitos. Esto incluye detectar conflictos entre requisitos, clasificar los requisitos, negociación, creación del modelo conceptual y establecimiento de las bases del diseño.
- Especificación de requisitos. Proceso de documentar el comportamiento requerido de un sistema software, a menudo utilziando una notación de modelado u toro lenguaje de especificación. Incluye detallas los requisitos, modelo formal, casos de uso y prototipo.
- Validación de requisitos. Examinar los requisitos para asegurarte de que definen el sistema que el cliente y los suarios desean. Para facilitar este proceso se pueden crear prototipos o simulaciones.

Tipos de requisitos

- Funcionales. Especifican las funciones que un sistema, o componente de un sistema, debe ser capaz de llevar a cabo.
- No funcionales. Especifican los aspectos técnicos que debe incluir un sistema. Pueden clasificarse en restricciones, limitaciones a las que se enfrentan los desarrolladores del sistema, y cualidades, características de un sistema que importan a los clientes y usuarios del mismo.
- De información. Describen necesidades de almancenamiento de información del sistema.

A su vez, los requisitos no funcionales pueden clasificarse como - Requisitos del producto. Detallan limitaciones o comportamientos exigidos al producto resultante del desarrollo. - Requisitos de la organización. Relacionadas con normativas de funcionamiento de la organización que lleva a cabo el desarrollo, sus procedimientos y políticas. - Requisitos externos. Cubren aspectos externos al sistema y a su proceso de desarrollo.

1.2.2 2.2 Obtención de requisitos

Determinar cuáles son los requisitos del sistema a desarrollar para llegar aun conocimiento suficiente del problema a resolver.

Técnicas de obtención de requisitos: - Entrevistas - Escenarios - Prototipos - Reuniones de grupo - Observación (técnicas etnográficas) - Otras

Proceso de obtención de los requisitos 1. Obtener información sobre el dominio del problema y el sistema actual. 2. Preparar las reuniones de elicitación y negociación. 3. Identificar y revisar los objetivos del sistema. 4. Identificar y revisar los requisitos de información. 5. Identificar y revisar los requisitos funcionales. 6. Identificar y revisar los requisitos no funcionales.

Obtener información sobre el dominio del problema

- Conocer el vocabulario propio
- Conocer las características principales del dominio.
- Recopilar información sobre el dominio.
- Facilitar la comprensión de las necesidades del sistema.
- Favorecer la confianza del cliente. #### Preparar y realizar sesiones de elicitación/negociación
- Identificar a los implicados
- Conocer las necesidades de clientes y usuarios.
- Resolver posibles conflictos. #### Identificar y revisar los objetivos del sistema
- Objetivos que se desean alcanzar una vez que el software esté en explotación.
- Si el sistema es suficientemente complejo se puede realizar una jerarquía de objetivos.
- De cada objetivo se puede describir su importancia, su urgencia, su estado y su estabilidad. #### Identificar y revisar los requisitos de información Información relevante para el cliente que debe gestionar y almacenar el sistema software. De cada requisito se puede describir:
- Objetivos y requisitos asociados
- Descripción del objetivo
- Contenido
- Tiempo de vida
- Ocurrencias simultáneas
- Importancia, urgencia, estado y estabilidad #### Identificar y revisar los requisitos funcionales De cada requisito se puede describir:
- Objetivo y requisitos asociados
- Secuencia de acciones
- Frecuencia de uso
- Rendimiento
- Importancia, urgencia, estado y estabilidad #### Identificar y revisar requisitos no funcionales Restricciones de las funciones descritas en la actividad anterior. De cada requisito se puede describir:
- Objetivos y requisitos asociados.
- Su importancia, urgencia, estado y estabilidad. Algunos requisitos no funcionales importantes son:
- Relacionados con facilidad de uso
- Relacionados con fiabilidad
- Relacionados con rendimiento
- Relacionados con el soporte (facilidad de mantenimiento y portabilidad)
- Relacionados con la implementación
- Relacionados con la interfaz
- Relacionados con la operación (quién gestiona el sistema)
- Relacionados con el empaquetado (instalación del sistema)
- Requisitos legales

Veremos dos técnicas para la obtención d requisitos: técnicas etnográficas y técnicas de entrevistas.

Técnicas de entrevista Técnicas encaminadas a obtener información sobre el sistema mediante el diálogo con los expertos en el dominio del problema.

- Planificación. Clarificar cuáles son los datos que se desean obtener, determinado qué personas deben entrevistar, cuándo y en qué lugar.
- Preparación. Preparar las preguntas e informarse sobre la persona que se va a entrevistar.
- Inicio. Exposición general de cómo se desarrollará la entrevista.
- Desarrollo. Fase central de la entrevista que consiste en la realización de preguntas por parte del entrevistador y la contestación de las mismas por parte del entrevistado.
- Cierre. Fase en la ques e resume la información anotada y se comprueba que contamos con toda la información solicitada.
- Conclusiones. Elaboración de un resumen formal de la entrevista.

Técnicas etnográficas Técnicas de observación que se usan para entender los procesos operacionales y ayudar a derivar requisitos sociales y de organización, de apoyo a dichos procesos. Efectivas para dos tipos de requisitos: - Requisitos que se derivan de la forma en que realmente trabaja la gente. - Requisitos derivados de la cooperación y el conocimiento de las actividades de otras personas.

1.2.3 2.3 Modelado de casos de uso

Técnica de ingeniería de requisitos que permite delimitar el sistema a estudiar, determinar el contexto de uso del sistema y describir el punto de vista de los usuarios del sistema.

Elemenos que componen el modelo de casos de uso: - Actores - Casos de uso - Relaciones entre actores, actores y casos de uso y casos de uso.

Para la representación y descripción de estos elementos se utilzan diagramas de casos de uso de UML y plantillas estructuradas para los actores y casos de uso.

El diagrama de casos de uso es un diagrama que representa gráficamente todos los elementos que forman parte del modelo de casos de uso junto con la frontera del sistema.

Actores Abstraccón de entidades externas al sistema que interactúan directamente con él. Especifican roles que adoptan esas entidades externas cuando interactúan con el sistema. Una entidad puede desempeñar varios roles simultáneamente a lo largo del tiempo. Un rol puede ser desemepeñado por varias entidades. Existen varios tipos de actores: - Principales: además de interactuar con el caso de uso, lo activan. - Secundarios. Interactúan con el caso de uso pero no lo activan. Los actores pueden ser - Personas - Dispositivos de entrada/salida - Sistemas de información externos - Temporizador o reloj Relaciones entre actores: - Generalización. Expresa un comportamiento común entre actores, es decir, se relacionan de la misma forma con los mismos casos de uso.

Casos de uso Especifica la secuencia de acciones, incluidas secuencias variantes y de error, que un sistema o subsistema puede realizar al interactuar con actores externos. Dependiendo de su importancia pueden ser: - Primarios. Procedimientos comunes más importantes. - Secundarios. Procesos de error o poco comunes. - Opcionales. Puede que no se implementen. Características: - Son iniciados por un actor, que, junto con otros actores, intercambian datos o control con el sistema a través de él. - Son descritos desde un punto de vista de los actores que interactúan con él. - Describen el proceso de alcance de un objetivo de uno o varios actores. - Tienen que tener una utilidad concreta para algún actor. - Acotan una

funcionalidad del sistema. - Describen un fragmento de la funcionalidad del sistema de principio a fin, y tienen que acabar y proporcionar un resultado. - Se documentan con texto informal.

Tipos de descripciones - Dependiendo de procesamiento, básico (descripción general del procesamiento) o extendido (descripción de la secuencia completa de acciones entre actores y sistema). - Dependiendo de su nivel de abstracción, esencial (expresado de forma abstracta) o real (expresado en base al diseño actual, en el que aparecen relaciones con la interfaz de usuario).

Escenarios Secuencia específica y concreta de acciones e interacciones entre los actores y el sistema objeto de estudio (historia particular). Tipos de escenarios: - Básico. Se corresponde con la funcionalidad básica y normal del caso de uso. - Secundarios. Se corresponde con funcionalidades alternativas y situaciones de error.

Relaciones en el modelo de casos de uso

- Asociación. Comunicación entre un actor y un caso de uso en el que participa.
- Inclusión. Inserción de comportamiento adicional dentro del caso de uso base que explícitamente hace referencia al caso de uso de inclusión.
- Extensión. Inserción de fragmentos de comportamiento adicional sin que el caso de uso base sepa de los casos de uso de extensión.
- Generalización. Relación entre un caso de uso general y otro más específico, que hereda y añade características al caso de uso general.

Características de la relación de inclusión: - Relación de dependencia que permite incluir el comportamiento de un caso de uso en el flujo de otro caso de uso. - El caso de uso que incluye se denomina caso base y al incluido, caso de uso de inclusión. - El caso de uso base se ejecuta hasta que alcanza el punto en el que encuentra la referencia al caso de uso de inclusión, momento en el que se pasa la ejecución a dicho caso. Cuando este se termina, en control regresa al caso de uso base. - El caso de uso de inclusión es utilizado completamente por el caso de uso base. - El caso de uso base no está completo sin todos sus casos de uso de inclusión. - El caso de uso de inclusión puede ser compartido por varios casos de uso base. - EL caso de uso de inclusión no opcional y es necesario para que tenga sentido el caso de uso base.

Características de la relación de extensión: - Relación de dependencia que especifica que el comportamiento de un caso de uso base puede ser extendido por otro caso de uso bajo determinadas condiciones. - El caso de uso base declara uno o más puntos de extensión que son como enganches en los que se pueden añadir nuevos comportamientos. - El caso de uso de extensión define segmentos de inserción los cuales se pueden insertar en esos puntos de enganche cuando se cumpla una determinada condición. - El caso de uso base no sabe nada de los casos de uso de extensión y está completo sin sus extensiones; de hecho, los puntos de extensión no tiene numeración en el flujo de eventos del caso de uso base. - El caso de uso de extensión no tiene sentido de forma separada de un caso de uso base.

Características de la relación de generalización: - Es una generalización entre un caso de uso general (caso de uso padre) y otros más especializados (casos de uso hijo). - Los casos de uso hijos heredan todas las características del caso de uso general. Pueden añadir nuevas características y anular características del caso de uso general, salvo relaciones, puntos de extensión y precondiciones.

Recomendaciones de uso de las relaciones: - Usar las relaciones entre casos de uso cuando simplifiquen el modelo. - Un uso sencillo de modelo de casos de uso es preferible a uno con demasiadas relaciones, ya que son más fáciles de entender. - El uso de muchos include hace que se tenga que ver más de un caso de uso para tener una idea completa. - Las relaciones extend son complejas y difíciles de entender para la comunidad de usuarios/clientes. - La generalización de casos de uso debería evitarse a menos que se utilicen casos de uso padres abstractos.

Proceso de construcción de casos de uso

1. Identificación de los actores
2. Identificación de los principales casos de uso de cada actor
3. Identificación de nuevos caso de uso a partir de los existentes
4. Creación del diagrama de casos de uso
5. Creación de las plantillas de los casos de uso básicos
6. Definición de prioridades y selección de casos de uso primarios
7. Escribir los casos de uso extendidos y crear prototipos de la interfaz de usuario

Estructura del diagrama de casos de uso

- Diagrama de paquetes. Diagrama de UML usado para describir la estructuración de un sistema en base a agrupaciones lógicas. También muestra las dependencias entre agrupaciones.
- Diagramas de actividad. Diagrama UML para la descripción del comportamiento que tiene un conjunto de tareas o procesos.

1.2.4 2.4 Análisis y especificación de requisitos

Fase de la ingeniería de requisitos en la que se examinan los requisitos (de los casos de uso) para delimitarlos y definir exactamente cada uno de ellos. Se trata fundamentalmente de: - Detectar y resolver conflictos entre los diferentes requisitos. - Delimitar el software y establecer con qué elementos externos interacciona. - Elaborar los requisitos del sistema para obtener, a partir de ellos, los requisitos del software.

Actividades durante el análisis: - Clasificación de los requisitos - Priorización de los requisitos - Modelado conceptual - Situación de los requisitos en la arquitectura del sistema - Negociación de los requisitos.

Análisis orientado a objetos El análisis orientado a objetos examina y representa los requisitos desde la perspectiva de los objetos que se encuentran en el dominio del problema. Los métodos de anaĺsis orientado a objetos se centran en obtener dos tipos de modelos: - Estático o de estructura - Dinámico o de comportamiento ¿Por qué usar análisis orientado a objetos? - Los términos usados en los modelos están cercanos a los del mundo real. - Se modelan tanto elementos y propiedades estáticas como dinámicas del ámbito del problema. - Se manejan conceptos comunes durante el análisis, diseño e implementación del software.

Obteción del modelo estático Proceso general: - Identificar los principales conceptos y sus relaciones y documentarlos.

1. Identificar los conceptos (cada uno será una clase)
2. Seleccionar los conceptos relevantes en el problema
3. Representarlos como clases en el diagrama de conceptos

- Partir del modelo de casos de usos, de la lista de requisitos y del glosario de términos.
- Representarlos con un diagrama de clases de UML en el que podrá haber conceptos o clases conceptuales, asociaciones entre conceptos, generalizaciones de conceptos y atributos de los conceptos.

Identificar e incorporar conceptos 1. Identificar los conceptos 2. Seleccionar los conceptos relevantes para el problema 3. Representarlos, como clases, en el diagrama de conceptos

Estrategias para identificar conceptos: Establecer una lista de categorías de conceptos y rellenarla a partir de la información que se dispone. Encontrar los términos que se correspondan con sustantivos o frases nominales; éstos van a ser los candidatos a conceptos.

Identificar e incorporar asociaciones Una asociación es una conexión significativa y relevante entre conceptos

1. Identificar las posibles asociaciones
2. Representarlas en el diagrama y seleccionar las que sean válidas
3. Asignarles nombre
4. Identificar la multiplicidad

Incorporar generalizaciones 1. Identificar posibles generalizaciones. A partir de las descripción del problema y de las clases conceptuales identificadas, encontrar clases conceptuales con elementos comunes. Definir las relaciones de superclase y subclase. 2. Validar las estructuras encontradas. Una subclase potencial debería estar de acuerdo con la regla del 100 % (conformidad con la definición de la superclase) y regla es-un (conformidad con la pertenencia al conjunto que define la superclase). 3. Representarlas en el modelo conceptual.

Agregar atributos 1. Identificar atributos desde casos de uso y lista de requisitos, así como otras fuentes de información. 2. Representarlos en el diagrama, en los conceptos o en las relaciones que correspondan. Tipos de atributos válidos: - Primitivos o valores puros de datos (entero, real, carácter, booleano, cadena. . .) - No primitivos (nombre de persona, número de teléfono, hora, fecha, dirección. . .)

Estructurar el modelo Paquete: división del modelo agrupando conceptos que tienen una fuerte relación entre sí (facilita el modelado y la posterior representación mediante diagramas).

Para estructurar el diagrama de conceptos o modelo de dominio: - Elementos que están en el mismo área de interés - Están juntos en una jerarquía de clases - Participan en los mismos casos de uso - Están fuertemente asociados

Obtención del modelo de comportamiento Estudio adicional del dominio del problema en el que se añaden los requisitos funcionales al modelo del análisis (qué hace el sistema sin explicar cómo lo hace). El diagrama de secuencia del sistema es un diagrama de UML en el que se muestra cómo los

eventos generados por los actores provocan la ejecución de una operación por el sistema, siendo visto este como una caja negra. Pasos a seguir, para todos los casos de uso: 1. Identificar los actores que inician las operaciones. 2. Asignar un nombre a todo el sistema. 3. Identificar y nombrar las operaciones principales del sistema a partir de las descripciones de los casos de uso. 4. Determinar los parámetros de las operaciones. 5. Incluir las operaciones en la clase que identifica a todo el sistema. 6. Hacer una descripción informal de la funcionalidad de cada operación.

Contratos Documento que describe lo que una operación se propone lograr, sin decir cómo se conseguirá. Contenido del contrato: - Nombre (nombre de la operación y sus parámetros) - Responsabilidad (descripción informal de las responsabilidades que debe cumplir la operación) - Tipo (concepto, clase o interfaz responsable de la operación) - Notas (notas de diseño, algoritmos. . .) - Excepciones (casos excepcionales) - Salida (mensajes o datos que proporciona) - Precondiciones (suposición acerca del estado del sistema o de los objetos del modelo conceptual antes de ejecutar la operación) - Poscondiciones (estado del sistema o de los objetos del modelo conceptual después de la ejecución de la operación)

Directrices para la elaboración de un contrato: - El nombre de la operación viene del diagrama de secuencia del sistema correspondiente - Comenzar con las responsabilidades, describiendo informalmente el propósito de la operación, continuar con las poscondiciones y finalizar con las demás secciones, especialmente con las precondiciones y excepciones. - Las poscondiciones deben describir los cambios de estado de un sistema, no sus acciones, estos son: creación y destrucción de objetos, creación y destrucción de enlaces, modificación de atributos. - Las poscondiciones deben expresarse mediante una frase verbal en pretérito.

Para especificar las poscondiciones hay que identificar en el diagrama de conceptos los objetos que intervienen en la operación.

1.3 Tema 3. Diseño e implementación

1.3.1 3.1 Fundamentos del diseño software

El diseño es el proceso de aplicar distintas técnicas y principios con el principio de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física. El diseño software es el proceso de aplicar distintas técnicas y principios del diseño con el propósito de traducir el modelo de análisis a una representación del software (modelo de diseño) que pueda codificarse. El diseño se descompone en dos subprocesos: 1. Diseño arquitectónico. Se describe cómo descomponer el sistema y organizarlo en los diferentes componentes (arquitectura del software). 2. Diseño detallado. Se describe el comportamiento específico de cada uno de los componentes del software.

Características: - El diseño implica una propuesta de solución al problema especificado en el análisis. - Es una actividad creativa que se apoya en la experiencia del diseñador. - Está apoyado por principios, técnicas, herramientas. . . - Es un proceso clave para la calidad del producto software. - Es la base para el resto de etapas del desarrollo. - Es un proceso de refinamiento. - El diseño va a garantizar que un programa funcione correctamente. Los principios del diseño ayudan a responder las siguientes preguntas: - ¿Qué criterios se usan para dividir el software en sus componentes individuales? - ¿Cómo

se extraen los detalles de una función o estructura de datos a partir de la representación conceptual del software? - ¿Cuáles son los criterios que definen la calidad técnica de un diseño software? Los principios son: - Abstracción - División de problemas y modularidad - Ocultamiento de información - Independencia funcional

Abstracción Mecanismo que permite determinar qué es relevante y qué no lo es en un nivel de detalle determinado, ayudando a obtener la modularidad adecuada para ese nivel de detalle. Tipos de abstracciones: - Abstracción de datos. Define un objeto compuesto por un conjunto de datos. - Abstracción de control. Define el sistema de control sin describir la información sobre su funcionamiento interno. - Abstracción procedimental. Se refiere a la secuencia de pasos que conforman un proceso determinado. #### División de datos y modularidad La división de problemas sugiere que cualquier problema complejo puede manejarse con más facilidad si se subdivide en elementos susceptibles de resolverse u optimizarse de manera independiente. La modularidad es la manifestación más común de la división de problemas. El software se divide en componentes con nombres distintos y abordables por separado, denominados módulos, que se integran para satisfacer los requisitos del problema. Ventajas de la modularidad: - Son más fáciles de entender y documentar que todo el subsistema o sistema. - Facilitan los cambios. - Reducen la complejidad. - Proporcionan implementaciones más sencillas. - Posibilitan el desarrollo en paralelo. - Permiten la prueba independiente (prueba de unidad). - Facilitan el encapsulamiento. #### Ocultamiento de información Los módulos deben especificarse y diseñarse de forma que la información (algoritmos y datos) contenida en un módulo sea inaccesible para los que no necesiten de ella. Ventajas: - Reduce la probabilidad de efectos colaterales. - Limita el impacto global de las decisiones de diseño locales. - Enfatiza la comunicación a través de interfaces controladas. - Disminuye el uso de datos globales. - Potencia la modularidad. - Produce software de calidad. #### Implementación funcional El software debe diseñarse de manera que cada módulo resuelva un subconjunto específico de requisitos y tenga una interfaz sencilla cuando se vea desde otras partes de la estructura del programa. La independencia se evalúa con dos criterios: - Cohesión. Un módulo cohesivo ejecuta una sola tarea, por lo que requiere interactuar poco con otros componentes en otras partes del programa. Idealmente se hace una sola cosa. La alta cohesión proporciona módulos fáciles de entender, reutilizar y mantener. - Acoplamiento. Indica la interconexión entre módulos en una estructura de software y depende, básicamente, de la complejidad de la interfaz entre módulos. El bajo acoplamiento proporciona módulos fáciles de entender y con menos efectos colaterales.

Las herramientas de diseño son instrumentos que ayuda a representar los modelos de diseño software. Algunas de las más usuales son: - Diagramas UML: clase, interacción, paquetes, despliegue. . . - Cartas de estructura - Tablas de decisión - Diagramas de flujo de control - Diagramas de Nassi-Shneiderman - Lenguajes de diseño de programas

Los métodos de diseño proporcionan las herramientas, técnicas y pasos a seguir para obtener diseños de forma sistemática. Características: - Principios en los que se basa - Mecanismos de traducción del modelo de análisis al modelo de diseño - Herramientas que permiten representar los componentes funcionales y estructurales - Heurísticas que permiten refinar el diseño - Criterios para evaluar la calidad del diseño Principales métodos de diseño: - Diseño estructurado de sistemas - Desarrollo de sitemas Jackson - Entidad-relación-atributo - Técnicas de modelado de objetos - Método Boock (diseño orientado a objetos) - Métodos orientados a objetos.

El modelo del diseño a nivel general está formado por varios subsistemas de diseño conjunto con las interfaces que requieren o proporcionan estos subsistemas. Cada subsistema de diseño puede contener diferentes tipos de elementos de modelado de diseño, principalmente realización de casos de uso-diseño y clases de diseño.

Se puede pensar en el modelo de diseño como una elaboración del modelo de análsis con detalles añadidos y soluciones técnicas específicas. Estrategia: 1. Convertir el modelo de análisis en un modelo de diseño. 2. Convertir el modelo de análisis en un modelo de diseño y usar una herramienta para recuperar una vista de análisis. 3. Congelar el modelo de análisis y hacer una copia en un modelo de diseño. 4. Mantener los dos modelos separados.

1.3.2 3.2 Diseño de la arquitectura

El diseño de la arquitectura es un proceso creativo que se interesa por entender cómo se debe organizar un sistema y cómo se tiene que diseñar la estructura global del sistema. Características: - Es la primera etapa en el proceso de diseño del software. - Es el enlace entre el diseño y la ingeniería de requisitos. - Proporciona un modelo arquitectónico que describe cómo se organiza el sistema en un conjunto de componentes (subsistemas) de comunicación. - Es la influencia dominante para los requisitos no funcionales. Importancia de la arquitectura - Facilita la comprensión de la estrcutra global del sistema. - Permite trabajar en los componentes de forma independiente. - Facilita las posibles extensiones del sistema. - Facilita la reutilización de los distintos componentes. Decisiones estructurales - Cómo se va a dividir el sistma en componentes - Cómo deben interactuar los componentes - Cuál va a ser la interfaz de cada componente. - Qué estilo arquitectónico se va a utilizar. El estilo arquitectónico depende de los requisitos no funcionales: - Rendimiento. La arquitectura se diseña para localizar las operaciones críticas dentro de un pequeño número de componentes desplegados en la misma computadora. - Seguridad. La arquitectura se diseña con una estructura, en capas, con los activos más críticos en las capas más internas, y con un alto nivel de validación de seguridad para esas capas. - Protección. La arquitectura se diseña para que las operaciones relacionadas con la protección se ubiquen en un componente individual o en un pequeño número de componentes. - Disponibilidad. La arquitectura se diseña para incluir componentes redundantes. - Facilidad de mantenimiento. La arquitectura se diseña usando componentes auto contenidos de grano fino que pueda cambiar con facilidad.

Herramientas de representación: - Diagrama de paquetes. Describe el sistema en torno a agrupaciones lógicas y proporciona una primera estructura del sistema. - Diagrama de componentes. Representa una estructuración concreta del sistema a partir de los componentes software (sistemas) y su interrelación (interfaces). - Diagrama de despliegue. Especifica el hardware físico sobre el que se ejecutará el sistema software y cómo cada subsistema se despliega en ese hardware. Los nodos representan tipos de recursos computacionales sobre los que se pueden desplegar los artefactos para su ejecución. Los artefactos representan especificaciones de elementos concretos del mundo real. - Diagramas de componentes. Un componente es la unidad software que ofrece una serie de servicios a través de una o varias interfaces.

Estilos arquitectónicos Proporcionan un conjunto de subsistemas predefinidos, especificando sus responsabilidades e incluyendo reglas y guías para organizar las relaciones entre ellos. No proporcionan la arquitectura del sistema, sino una guía de como obtenerla.

Arquitectura Modelo-Vista-Controlador Separa la presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí: - Modelo. Maneja los datos del sistema y las operaciones asociadas a esos datos. - Vista. Define y gestiona cómo se representan los datos al usuario. - Controlador. Dirige la interacción del usuario y pasa estas interacciones a vista y modelo. Principio de diseño: - Cada sistema puede diseñarse independientemente. - Se aumenta la cohesión de los subsistemas si la vista y el controlador se unen en una capa llamada interfaz de usuario. - Se reduce el acoplamiento puesto que la comunicación entre los subsistemas es mínima. Cuándo se usa: - Cuando existen múltiples formas de interactuar con los datos. - Cuando se desconocen los requisitos futuros para la interacción y la presentación. Ventajas: - Permite que los datos cambien de manera independiente de su representación (y viceversa). - Soporta diferentes representaciones de los mismos datos. - Los cambios en una representación se muestran en todos ellos. Desventajas: - Código adicional y complejidad de código cuando el modelo de datos y las interacciones son simples. ##### Arquitectura en capas Organiza el sistema en capas con funcionalidad relacionada con cada capa- Una capa da servicios a la capa de encima y las capas de nivel inferior representan servicios núcleo que es probable que se utilicen a lo largo de todo el sistema. Principios de diseño: - Las capas se pueden diseñar, construir y probar independientemente. - Una capa bien diseñada presenta alta cohesión. - Una capa bien diseñada no tiene conocimiento de las capas superiores (ocultamiento de información). - Las capas deben estar desacopladas. Todas las dependencias en un sentido. Todas las dependencias en las interfaces. - Las capas inferiores se deben diseñar para representar servicios de bajo nivel (bajo acoplamiento). Cuándo se usa: - Al construir nuevas facilidades encima de los sistemas existentes. - Cuando el desarrollo se dispersa a través de varios equipos de trabajo. - Cuando existe un requisito de seguridad multinivel. Ventajas: - Permite la sustitución de capas completas siempre que se conserve la interfaz. - En cada capa se pueden incluir facilidades redundantes. Desventajas: - Es difícil ofrecer una separación limpia entre capas. - El rendimiento es un problema debido a los múltiples niveles de interpretación. ##### Arquitectura de repositorio Todos los datos en un sistema se gestionan en un repositorio central, accesible a todos los componentes. Los componentes no interactúan directamente, sino solo a través del repositorio. Principios de diseño: - Se pueden diseñar sistemas independientes, aunque deben conocer el esquema de repositorio. - Cada subsistema tiene definida una funcionalidad específica (alta cohesión). - El acoplamiento de los subsistemas con el respositorio es alto. Cuándo se usa: - Cuando se tiene un sistema donde los grande volúmenes de información generados se deben almacenar durante mucho tiempo. - En sistemas en los que la inclusión de datos en el repositorio active una acción o herramienta. Ventajas: - Los componentes pueden ser independientes, no necesitan conocer la existencia de otros. - Los cambios en un componente se pueden propagar hacia todos los componentes. - La totalidad de datos se puede gestionar de manera consistente. Desventajas: - Los problemas en el repositorio afectan a todo el sistema. - Existencia de ineficiencias al organizar toda la comunicación a través del repositorio. - Quizás sea difícil distribuir el repositorio en varias computadoras. ##### Arquitectura cliente-servidor La funcionalidad del sistema se organiza en servicios, y cada servicio lo entrega un servidor diferente. Los clientes son usuarios de dichos servicios y para usarlos ingresan a los servidores. Principios de diseño: - Permite diseñar, implementar y probar servidores y clientes independientemente. - Mejora la cohesión de los susbsistemas al proporcionar servicios específicos a los clientes. - Reduce el acoplamiento al establecer un canal de comunicación para el intercambio de mensajes. - Aumenta la abstracción al tener subsistemas distribuidos en nodos separados. Cuándo se usa: - Cuando desde varias ubicaciones se tiene que acceder a una base de datos compartida. - Cuando la carga de un sistema es variable. Ventajas: - Los servidores se pueden distribuir a través de una red. - Se pueden añadir fácilmente servidores o clientes extra. - Se pueden escribir clientes

para nuevas plataformas sin modificar a los servidores. Desventajas: - Es susceptible de fallos del servidor. - El rendimiento es impredecible porque depende de la red. - Problemas administrativos cuando los servidores sean propiedad de distintas organizaciones. #### Actividades del diseño arquitectónico Identificar los objetivos del sistema: - Identificar las cualidades deseables del sistema. - Obtener a partir de los requisitos no funcionales. - Seleccionar un pequeño conjunto de objetivos del diseño que el sistema debe satisfacer necesariamente. - Adquirir compromisos, puesto que muchos objetivos de diseño son contrapuestos. Determinar la arquitectura software: - Seleccionar el estilo arquitectónico que mejor se adapte. - Identificar subsistemas en el dominio del problema. - AÑadir subsistemas predefinidos de acuerdo al estilo arquitectónico seleccionado. Modelar la arquitectura software: - Diseñar la arquitectura en un diagrama de paquetes. - Elaborar el diagrama de componentes de la arquitectura. - Realizar el diagrama de despliegue. Diseñar la arquitectura en un diagrama de paquetes. Los doagramas de paquetes permite modelar una primera estructuración del sistema en base a uno o más paquetes. - Cada paquete agrupa a un conjunto de clases relacionadas semánticamente. - Los paquetes peuden guiar en la identificaićon de los subsistemas y/o componentes reutilizables. Al identificar los paquetes hay que tener en cuenta lo siguientes aspectos: - Identificar paquetes cohesivos y con poca interacción con otros paquetes de acuerdo con la arquitectura planteada. - Diseñar paquetes que se puedan reutlilzar en otros proyectos. - Integrar subsistemas de proyectos anteriores. - Evitar las dependencias cíclicas entre paquetes. Elaborar el diagrama de componentes. Los diagramas de componentes permiten estructurar el sistema de subsistemas en base a componentes que pueden reemplazarse. A partir del diagrama de paquetes determinar qué partes pueden definir componentes. Para ello habrá que: - Definir interfaces sumistradas de cada componentes para determinar las operaciones que pueden ser usadas por otros componentes. - Especificar las interfaces necesarias para cada componentes para poder desarrollar su funcionalidad. - Construir el componente de forma que su contenido interno sea independiente del exterior, salvo a través de las interfaces suministradas y necesarias. Refinar la descomposición en subsistemas. - Refinar los subsistemas hasta satisfacer los objetivos de diseño. - Establecer los siguientes aspectos genéricos: - Correspondencia hardware-software - Administración de datos persistentes - Especificación de una política de control de accesos - Diseño del flujo de control - Diseño de la concurrencia y sincronización - Definición de las condiciones del entorno ### 3.3 Diseño e implementación de los casos de uso #### Patrones de diseño para asignar responsabilidades Responsabilidad. Contrato u obligación que debe tener un objeto en su comportamiento: - Hacer - Hacer algo en uno mismo - Iniciar una acción en otros objetos - Controlar y coordinar actividades en otros objetos - Conocer - Estar enterado de los datos privados encapsulados - Estar enterado de la existencia de objetos conexos - Estar enterado de las cosas que se pueden derivar o calcular Patrón de diseño. Descripción de un problema con su solución en un determinado contexto. Son una forma de reutilizar el conocimiento y la experiencia de otros diseñadores. Partes esenciales de un patrón: - Nombre. Que sea una referencia significativa al patrón. - Problema. Una descripción del problema que enuncie cúando se puede aplicar el patrón. - Solución. Una descripción de la solución de diseño, sus relaciones y responsabilidades. Es una plantilla para que unas oslución se instale en diferentes formas. - Consecuencias. Los resultados (buenos y malos) y las negociaciones de aplicar el patrón. #### Patrones GRAPS General Responsability Assignment Software Patterns. Describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones. Características: - No expresan nuevos principios de la ingeniería del software. - Codifican conocimiento, expresiones y principios ya existentes. - Son un ejemplo de la fuerza de abstracción porque dan nombre a una idea compleja. Apoyan la incorporación de conceptos a nuestro sistema congnitivo y memoria. Facilitan la comunicación.

Experto en información Problema: ¿Cuál es el princpiio general para asignar responsabilidades a objetos? Solución: Asignar responsabilidad a la clase que contine la información necesaria para llevarla a cabo. Consecuencias: Malas: en ocasiones va en contra de los principios de acoplamiento o cohesión. Buenas: mantiene el oculpamiento de información y distribuye el comportamiento.

Creador Problema: ¿Quién debería ser el responsable de la creación de una instancia de alguna clase? Solución: Asignar a la clase B la responsabilidad de crear la instancia A cuando: - B agrega objetos de A - B contiene objetos de A - B registra objetos de A - B utiliza objetos de A - B tiene los datos de inicialización de A Consecuencias: Malas: No es conveniente su uso cuando se construye a partir de instancias existentes. Buenas: produce bajo acoplamiento.

Bajo acoplamiento Problema: ¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento de la reutilización? Solución: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo. Consecuencias: Malas: llevado a extremo puede ocasionar diseños pobres; en un conjunto de clases debe haber un nivel de acoplamiento moderado y adecuado. Buenas: no afectan los cambios en otros componentes. Fáciles de entender de manera aislada. Conveniente para reutilización.

Formas comunes de acoplamiento: - El tipo X tiene un atributo que referencia a una isntancia del tipo Y o al propio tipo Y. - El objeto de Tipo X invoca a los servicios de un objeto de Tipo Y. - El tipo X tiene un método que referencia a una instancia de Tipo Y, o al propio Tipo Y, de algún modo. - El tipo X es una subclase, directa o indirecta, del tipo Y. - El tipo Y es una interfaz y el tipo X implementa esa interfaz.

Grados de cohesión funcional: - Muy baja cohesión. Una única clase es responsable de muchas cosas en áreas funcionales diferentes. - Baja cohesión. Una única clase tiene la responsabilidad de una tarea compleja en un área funcional. - Alta cohesión. Una única clase tiene una responsabilidad moderada en un área funcional y colabora con otras clases para llevar a cabo las tareas. - Moderada cohesión. Una única clase tiene responsabilidades ligeras y únicas en unas pocas áreas diferentes que están lógicamente relacionadas con el concepto de la clase, pero no entre ellas.

Controlador Problema: ¿Quién debe ser responsable de gestionar un evento de entrada al sistema? Solución: Asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que represente: - El sistema global, dispositivos o subsistemas. - El escenario de caso de uso en el que tiene lugar el evento del sistema. Consecuencias. Malas: controladores saturados. Buenas: se asegura que la lógica de la aplicación no se maneje en la interfaz. Aumento de la reutilziación y bajo nivel de acoplamiento. Posibilidad de razonar sobre el estado de los casos de uso.

Elaboración del modelo de interacción de objetos Directrices generales: - Las bases principales para obtener los diagramas de interacción son los contratos y el modelo conceptual - El modelo conceptual sirve de guía para saber qué objetos pueden interaccionar en una operación. - Todo lo especificado en el contrato, especialmente las poscondiciones, excepciones y salidas, tienen que satisfacerse en el correspondiente diagrama de comunicación. - Para la elaboración de cada diagrama de comunicación se aplican los patrones de diseño. Pasos a seguir 1. Elaborar los diagramas de interacción. Para cada operación especificada en los diagramas de secuencia. - Tener presente el diagrama de conceptos y el contrato de la operación. - Representar las relaciones del controlador con los objetos que intervienen en la interacción. - Asignar responsabilidades a objetos. - Establecer tipos de enlaces entre objetos. 2. Inicialización del sistema 3. Establecer relaciones entre el modelo y la interfaz de usuario

1.3.3 3.4 Diseño de la estructura de objetos

Modelado de la estructura de objetos Diagrama de clases del diseño. Describe gráficamente las especificaciones de las clases e interfaces software, y las relaciones entre estas, en una aplicación. Representa la solución a un problema. Puede contener los siguientes elementos: - Clases con sus atributos y operaciones - Interfaces con sus operaciones y constantes - Relaciones entre clases, entre interfaces o entre clases e interfaces - Información sobre el tipo de los atributos y parámetros - Navegabilidad de las asociaciones Herramienta para su representación : diagrama de clases UML. Pasos a seguir: 1. Identificar y representar las clases 2. Identificar y añadir las operaciones 3. Añadir tipos de atributos y parámetros 4. Identificar y representar las asociaciones y su navegabilidad 5. Identificar y representar las relaciones de dependencia 6. Incluir relaciones de generalización

Identificar y representar las clases Todos los objetos en los diagramas de interacción tendrán su correspondiente clase en el diagrama de clases del diseño. Las clases identificadas tomarán sus atributos del modelo conceptual y de los diagramas de interacción. **Identificar y añadir operaciones** Todos los envíos de mensajes deben tener su operación en la clase correspondiente **Añadir tipos de atributos y parámetros Identificar y representar las asociaciones y su navegabilidad** Todos los enlaces estereotipados con <> deben tener su correspondiente asociación. La navegabilidad la da la dirección del envío de mensaje y la multiplicidad la existencia de multiobjetos. Todos los enlaces estereotipados con <>, <

> o <> estarán en el diagrama de clases de diseño como una dependencia. **Incluir relaciones de generalización** Las generalizaciones que hay en el modelo conceptual también pueden aparecer en el diagrama de clases del diseño. Porceder de la siguiente forma: - En el diagrama de clases del diseño obtenido hasta ahora, observar: - Clases con nombres que identifiquen las distintas clasificaciones de un conjunto de objetos. - Clases con los mismos atributos. - Clases con la misma asociación con una clase. - Clases con operaciones con el mismo nombre o parecido (teniendo en cuenta la similitud del diagrama de comunicación correspondiente) - Si se da alguna o varias de estas situaciones establecer una generalización entre las clases, llevando a la superclase los atributos, operaciones y asociaciones comunes.