uc3m | Universidad Carlos III de Madrid

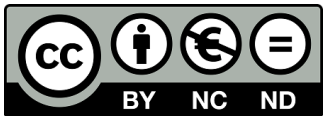
Grado en Ingeniería Informática

2020-2021

Apuntes

Ingeniería del Software

Jorge Rodríguez Fraile[1]



Esta obra se encuentra sujeta a la licencia Creative Commons
Reconocimiento - No Comercial - Sin Obra Derivada

[1]Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. INFORMACIÓN

1.1. Profesores

Magistrales: JESÚS MANUEL POZA CARRASCO (jepozac@inf.uc3m.es)

Prácticas: Maria Luisa Arjonilla marjonil@inf.uc3m.es

1.2. Recursos

Why Software Is Eating the World - Andreessen Horowitz

Images and Opinions

ArticleS.UncleBob.PrinciplesOfOod

https://loufranco.com/wp-content/uploads/2012/11/cheatsheet.pdf

Libros de Martin Fowler:

- NoSQL Distilled
- Domain Specific Languages
- Refactoring
- P of EAA
- UML Distilled
- Refactoring Ruby Ed.
- Analysis Patterns
- Planning XP

2. TEMA 1 - REQUISITOS

https://www.barchart.com/stocks/indices/sp-sector/information-technology

https://a16z.com/2011/08/20/why-software-is-eating-the-world/

https://www.youtube.com/watch?v=8WVoJ6JNLO8&feature=youtu.be&ab_channel=RankingTheWorld

2.1. Introducción

Ingeniería del software: Aplicación sistemática de conocimientos, métodos y experiencias científicas y tecnológicas al diseño, implementación, pruebas y documentación de software. Es una disciplina ubicua, está presente en todo y en todos los tiempos.

Ingeniería de requisitos: Desarrollo sistemático de los requisitos a través de un proceso iterativo (se repite de forma circular y va mejorando) y cooperativo en el que se analiza el problema, se documenta el resultado en diversos formatos de representación, y se comprueba la exactitud de la comprensión alcanzada.

En ambas disciplinas el producto es software, son más artesanales que ingenieriles y es necesario describir y documentar lo que se va a producir.

2.2. Requisitos

Lista ordenada de capacidades, funciones, aspectos que debe cumplir el software que vamos a desarrollar.

- Condición o capacidad=funcionalidad que el usuario necesita, que debe poseer el sistema o aplicación.
- Estos tienen función contractual en las compañías, para poder exigir alcanzar lo acordado.

Especificación: Lista de requisitos

Hay que tener en cuenta a todos los interesados o stakeholders, aquellos que utilizan el sistema. Aunque a veces estos no saben lo que quieren y por eso se puede volver un proceso complicado.

Hay estudios que demuestran que la especificación de requisitos es un factor muy importante para que un proyecto no fracase.

- La nula o mala especificación provoca grandes pérdidas de dinero.
- Aunque tampoco garantiza el éxito.

2.2.1. Obtención, descripción y escritura de requisitos

Tipos:

- Requisitos de capacidad (funcionales): Son los de cara al usuario. Funciones y operaciones requeridas para resolver un problema o alcanzar un objetivo.
- Requisitos de restricción (no funcionales): Son los relativos a las características técnicas, que hacen que el sistema funcione aceptablemente. Restricciones impuestas sobre la manera en que el problema es resuelto o el objetivo alcanzado.

V

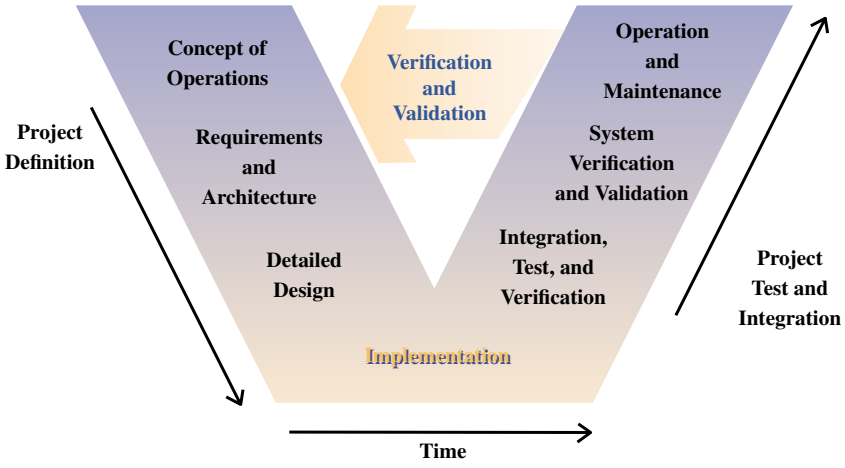


Fig. 2.1: V Ingenieria de Software

Plan para obtener requisitos:

- Identificar: Conocer a los stakeholders.
- Entrevistar: A los stakeholders para recoger requisitos.
- Escribir: Los requisitos se documentan.
- Revisar: Analizamos los requisitos, si pasa la revisión seguimos, si no volvemos a identificar.

Técnicas para la obtención y descripción de requisitos.

- Textuales: Accesibles a un cliente sin formación específica.
  - En prosa común y corriente.
  - Texto estructurado, casos de uso o tabla de roles de usuario y servicio.
- Gráficas: Requieren un cierto grado de formación técnica.
  - Con cuidado, no se tienen que convertir en diseño.
    - Diagramas de flujos de datos.
    - Diagramas de actividad.
    - diagramas de estado.
  - Interfaces de usuario y prototipos: No confundir con diseño.

Técnicas de elicitación:

- Historias de usuario: Manera cómoda de obtener especificaciones de los usuarios. Lo escriben ellos mismos.
  - Como ___ quiero ____ para ____
- Prototipos: Permite extraer requisitos, ensayar soluciones y eliminar partes arriesgadas. Se realizan en fases tempranas, para que el usuario lo vea y exprese lo que desea y saber si se va por buen camino. No tiene que ser funcional más visual, y no tarda demasiado.
  - Mock-ups (maquetas): Modelo o prototipo.
- Diagramas de Casos de uso: Muestran las funciones y la relación entre los actores y dichas funciones de un sistema.

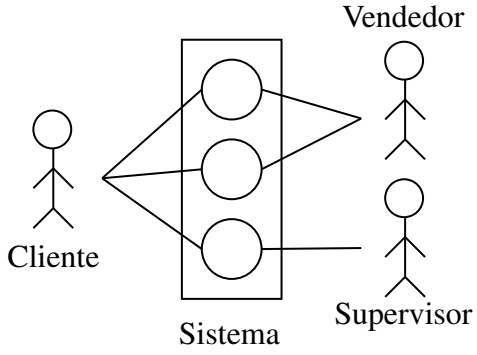


Fig. 2.2: Caso de Uso

Ciclo de vida:

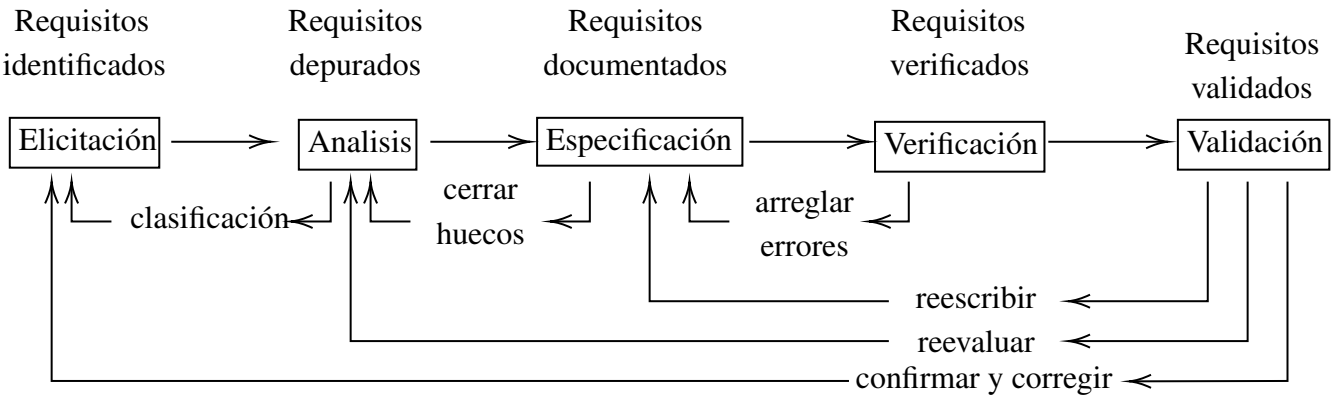


Fig. 2.3: Ciclo de vida de los requisitos

- Verificación: Comprobar que funciona tal y como se espera.
  - Do the thing right, hacer las cosas
- Validación: El usuario/stakeholder está de acuerdo/acepta.
  - Do the right thing, hacer la tarea correcta.

Especificación inteligente de requisitos:

- e**S**pecifico: Claro y simple.
- **M**edible: Se puede cuantificar y evaluar.
- **A**lineado: Con la estrategia o con el fin del sistema.
- **R**ealista: Puede conseguirse con un numero de recursos lógicos.
- limitado en **T**iempo: Establece un periodo de tiempo.

2.2.2. Como escribir buenos requisitos. Propiedades de los requisitos

Una buena especificación es completa, consistente entre requisitos y correcta definición de requisitos.

- Completa: Que no falte ningún requisito, cubra todas las necesidades y requisitos del usuario.
- Consistente: No haya conflicto entre requisitos, ya que hay tantos que surgen sin querer.
- Correcta: Cada requisito por individual sea de la manera que debe ser, se entienda, sea adecuado gramaticalmente y ortográficamente.
- Además:
    - Modificable: Puedan variar con las revisiones.
    - Verificable: Hay una condición que puedo probar y verificar.
    - Trazable: Se puede seguir el proceso llevado, demostrando que todo está presente.
    - Claros/No ambiguos: No da lugar a varias interpretaciones.

Estructura de la especificación:

- Puede haber centenares de requisitos.
- Escribir los requisitos para no olvidarlos y además pueden ser firmados (pieza clave en contratos), fuente del diseño y se verifica el software con ellos.
- La organización es vital.

Especificaciones completas:

- Muchas reuniones con clientes, afectados, expertos, etc. Cruzar las revisiones de las distintas reuniones, para conectar todos los requisitos.
- Revisiones por pares.
- Emplear check-lists de proyectos previos que vamos creando con la experiencia, que nos ayuden a no olvidar nada.
- Reutilizar requisitos de proyectos previos similares.
    - Grano grueso: Componentes reutilizables
    - Grano fino: buscadores avanzados.

Especificación consistente:

- Tras escribir los requisitos, buscar contradicciones y redundancias, que se solapen o dependan unos de otros.
- Se usan revisiones en grupo y técnicas automáticas de herramientas de apoyo.
    - Comparación de grafos semánticos.
    - Detección de unidades inconsistentes.

Poner un glosario con términos del dominio y posibles acrónimos utilizados.

La especificación de requisitos no es una novela:

- Estilo narrativo objetivo, plano. Sin valoraciones, adjetivos o adverbios.
- Textos simples y claros
- Gramáticas fijas y simples
- Voz activa
- Evitar terminología no definida.
- Requisitos fáciles de medir.
- Ambigüedad cero.
- No dar conocimiento por sentado
    - Deben ser autocontenidos.
- Orden lógico y bien estructurado.
- Cada requisito debe ser atómico.
- Escribirse con herramientas especializadas para:
    - Identificarlos unívocamente.
    - Atomizarlos
    - Organizarlos, categorizarlos y relacionarlos.
    - Reutilizarlos por separado o en conjunto.
    - Medir su calidad individual o global.

Indicadores de calidad:

- Tamaño del requisito: El justo, 1 frase o como mucho 2. Medido en caracteres, palabras o párrafos.
- Legibilidad: La máxima posible, evitar referirnos a requisitos anteriores.
- Tiempo verbal: Imperativa en lugar de condicional.
- Modo verbal: Voz activa.
- Sentencias opcionales y especulativas: Evitar "quizá", "usualmente"...
- Expresiones ambiguas: Evitar expresiones, poner cifras numéricas y medibles.
- Sentencias subjetivas: Evitar incluirse a uno mismo. "yo creo...", "En mi opinión"
- Sentencias implícitas: Evitar abuso de los pronombres, siempre que se pueda.
- Conectores: Su abuso indica que se incluye más de una necesidad o exceso de detalle.
- Negaciones: más de una palabra negativa en la misma frase es difícil de entender.
- Sentencias incompletas: Evitar sentencias de tipo: "etcétera", "entre otros", "..."
- Términos propios de diseño o de flujo: Evitar términos que denotan diseño.
- Numero de términos del dominio: Exceso de términos del dominio del contexto puede indicar que se están mezclando diferentes necesidades en el mismo requisito.
- Numero de verbos principales: Verbos los justos y necesarios.
- Acrónimos y abreviaturas: Solo si están definidos en el documento.
- Estructura gramatical: Usuario/Acción/Objeto/Cualificador.

Los siete pecados del especificador:

- Ruido: Información irrelevante.
- Silencio: Aspectos no cubiertos.
- Ambigüedad: Admiten varias interpretaciones.
- Referencia futura: Definida más adelante, pero si se admiten referencias cruzadas entre requisitos.
- Contradicción; Dos o más formas incompatibles.

- Sobre especificación: Elementos que no corresponden al problema sino a una posible solución.
- Pensamiento ilusorio: Definir el problema de tal forma que imposibilita una solución realista.

Errores típicos:

- Modo condicional: Evitarlo, usar presente del indicativo o futuro inmediato.
- Detalles de diseño: Evite detalles demasiado técnicos o del diseño.
- Opcionalidad: Exprese la opcionalidad mediante un atributo del requisito, indicarlo directamente.
- Atomicidad: Dar demasiado detalle, mejor pone un requisito por cada necesidad.
- Acrónimos y vaguedad: Acrónimos cuando estén comúnmente aceptados y utilizar medidas físicas, no términos como rápido o amigable.
- Puntuación y legibilidad: Uso apropiado de puntuación. El número de sílabas por palabra y palabras por frase es un indicador de legibilidad.
- Pronombres: El exceso de pronombres puede hacer un requisito difícil de entender.
- Pseudocódigo: Evitar el uso de pseudocódigo y los requisitos extensos.
- Numero de términos: Exceso de términos diferentes en el mismo requisito puede indicar mezcla de necesidades o demasiado detalle.
- Subjetividad, exceso de negaciones.
- Falta de precisión: Evite expresiones vagas.

2.2.3. Tipos de requisitos. Organización de requisitos. Matrices de trazabilidad

Tipos de requisitos del software:

- Niveles:
  - Requisitos del usuario: Son aquellos que nos los dicen los usuarios.
  - Requisitos del Software: Son las descripciones formales de los requisitos de usuario, documentadas. Agregan detalles a la especificación.
  - Diferencias:

	Requisitos del Usuario	**Requisitos del Software**
objetivos	planteamiento del problema captura de requisitos	refinamiento del problema análisis de requisitos
fuente	usuario/cliente	usuario/cliente y análisis
responsable	usuario/cliente	analista
audiencia	usuario/cliente (y desarrollador)	desarrollador (y usuario/cliente)
énfasis	perspectiva del producto características de los usuarios entorno operacional captura mediante prototipos	conocimiento de expertos modelo, métodos formales organización, no dejar cabos sueltos consistencia y completitud

- Clasificación de requisitos:
  - Requisitos funcionales: Que tiene que hacer.
  - Requisito no funcionales: Como lo hace, características técnicas y restricciones del cómo.
    - Consumo de recursos: Memoria, capacidad...
    - Rendimiento: Velocidad, tiempo de respuesta...
    - Fiabilidad y disponibilidad.
    - Manejo de errores: Del entorno e internos.
    - Requisitos de interfaz.
    - Restricciones.
    - Seguridad del sistema o de las personas.

Métodos para organizar los requisitos del software:

- Según el modelo del sistema:
  - Según el modelo de casos de uso.
  - Según el modelo conceptual.
- Ciclo de vida de los requisitos.
- Uso de herramientas para analizar y organizar requisitos.

Matrices de trazabilidad: Para ir siguiendo que no se nos pase ningún requisito de usuario o documentar para que sea de software.

- Relaciona un requisito de usuario con aquellos de software que lo contienen, y también controlamos lo contrario, no poner algo que no nos piden.

RU	**RS**	**Unidades de implementación**
1	1, 2	1
2	3	1, 3
3	2, 4, 5	2

- También puede ser de requisitos y clases, marcando en donde se encuadra cada uno. Así se ve si todos tienen clase y no haya clases vacías, o requisitos sin asignar.

	Clase 1	**Clase 2**
r1		X
r2	X	

Matriz de referencias cruzadas:

- Controla:
  - Conflictos: fallos entre requisitos.
  - Acoplamiento: dependencias de las que tenemos que tener cuidado, pero no son fallos.
  - Redundancia: Decir lo mismo dos veces, se puede reformular, pero no es fallo.

	R1	R2	R3	R4	R5	R6	R6
R1			+		x	x	
R2							
R3	+			+		+	
R4			+		x		o
R5	x			x			
R6	x		+				
R7				o			

Conflicto (x)	Acoplamiento (+)	Redundancia (o)	Independiente
R1 y R5 R1 y R6 R4 y R5	R1 y R3 R3 y R4 R3 y R6	R4 y R7 (->R7xR5, R7+R3)	R2

Ejemplo plantilla de requisito

ID	007
Título	Usuarios Visitantes
Tipo	Funcional
Descripción	Los usuarios Visitantes no son usuarios registrados del sistema: pueden acceder libremente a las funciones del rol Visitante sin necesidad de registrarse ni de iniciar sesión, y sin límite de sesiones simultáneas
Pruebas	1. Comprobar que para todas las funciones del rol Visitante el sistema no requiere inicio de sesión. 2. Comprobar que el número de sesiones simultáneas de usuarios Visitantes es limitado.

Propiedades y atributos deseables de los requisitos del software.

- Completitud, organización por tipos de requisitos. Matriz de trazabilidad.
- Consistencia, matriz de referencias cruzadas.
- Corrección, tamaño adecuado, claridad, capacidad de comprobación y condiciones de error.

3. TEMA 2 - MODELADO

Modelo: Abstracción o simplificación de una realidad compleja. Representar realidades. Contar historias.

- Tipos:
  - Modelos formales: Funcionan más como un mensaje. Lenguaje universal, precisión, rigor.
  - Modelos informales: Sin lenguaje común.
- El lenguaje es vehículo del pensamiento.
- Elemento esencial del proceso de desarrollo de software.
- Requiere un lenguaje adecuado.

Modelar es pensar con diagramas.

Propiedades deseables:

- Comprensible: Se pueda entender fácilmente.
- Preciso: Representación fiel del sistema.
- Predictivo: Se puede utilizar para obtener conclusiones sobre el sistema.
- Barato: más económico que construir y estudiar el sistema.

Sistema: Colección de elementos organizados para cumplir una finalidad.

Modelo: Abstracción de un sistema, una simplificación para comprenderlo mejor.

Diagrama: Representación gráfica de un conjunto de elementos interconectados.

Componentes de un lenguaje:

- Fonología.
- Semántica.
- Gramática.
- Pragmática.

UML - Unified Modeling Language: Es un lenguaje de modelado.

Tipos de Diagramas:

- Los más importantes:
  - Casos de Uso: Muestra la relación de los actores con el sistema y lo que este le ofrece al user. Sirve para especificar la comunicación.
  - De clases: Muestra las clases de nuestro sistema como sustantivos, con sus atributos y operaciones, y conectan con los relacionados.
  - De componentes: Representa como un sistema de software es dividido en componentes y muestra las dependencias entre los componentes.
- Más tipos:
  - De despliegue: Muestra la distribución de componentes del sistema de información con respecto a la partición física. Dirigido a diseñadores.
  - De actividades: Diagrama de flujo que muestra actividades ejecutadas por un sistema.
  - De objetos.
  - De secuencia.
  - De colaboración.
  - De tiempo.
  - De interacción.

Ingeniería directa e inversa

- Ingeniería directa: Construir sistemas a partir de lo que los modelos especifican.
- Ingeniería inversa: Representar en modelos sistemas existentes.

Objetos y clases

- Objeto: Entidad concreta con identidad, estado y comportamiento.
  - Un objeto es una instanciación de una clase.
  - Tipos:
    - Objetos físicos.
    - Objetos lógicos.
    - Objetos históricos.

- Atributos

  Propiedad compartida por los objetos de una clase, cada atributo tiene un valor.

  Atributo derivado: Propiedad redundante que puede ser calculada a partir de otras.

  visibilidad nombre multiplicidad: Tipo = valorInicial {propiedades}

- Operaciones

  Función o transformación que puede aplicarse a los objetos de una clase, pueden ser invocadas por otros objetos, o el mismo objeto.

  Método: Especificación procedimental de una operación.

  visibilidad nombre (param: Tipo = valDef,...): TipoRet {propiedades}

- Clase: Conjunto de entidades abstractas con estructura y comportamientos comunes.
  - La clase se usa como plantilla para construir objetos.
- Análisis: Especificación, vista externa, caja negra.
  - Clases, atributos y operaciones corresponden a conceptos del dominio.
  - Es habitual usar una notación simplificada al máximo.
- Diseño: Implementación, vista interna, caja blanca.
  - Clases, atributos y operaciones corresponden a fragmentos de código.
  - Nuevos artefactos y soluciones que dependen del lenguaje y la plataforma de implementación.

Enlace: Conexión entre objetos.

Asociación: Especificación de un conjunto de enlaces, representan la estructura y el comportamiento del sistema.

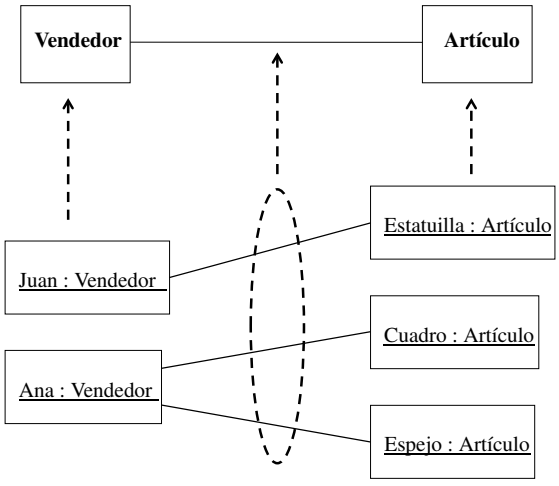


Fig. 3.1: Asociación

- Nombre de asociación y Dirección del nombre.

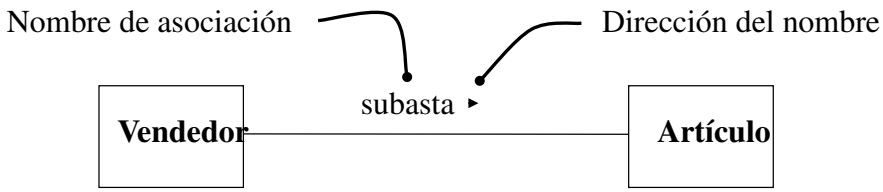


Fig. 3.2: Nombre y Dirección de asociación

- Nombres de rol

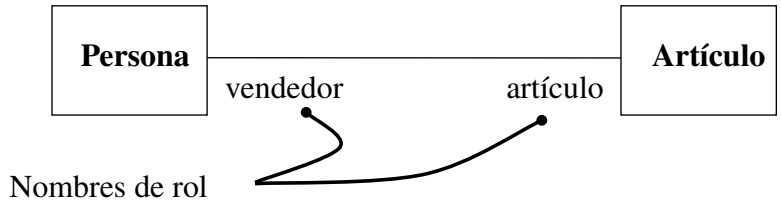


Fig. 3.3: Rol asociación

- Restricciones y notas
- Multiplicidad de la asociación

  Asociación binaria, la multiplicidad de un extremo de asociación específica el número de instancias destino que pueden estar enlazadas con una única instancia origen a través de la asociación.

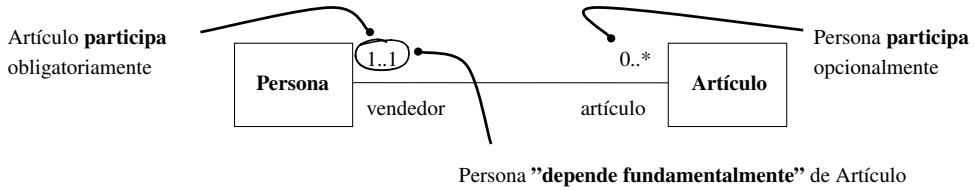


Fig. 3.4: Multiplicidad de la asociación

Valores tipicos:

- 0..1 cero o uno
- 1..1 uno y solo uno
- 0..* desde cero hasta muchos
- 1..* desde uno hasta muchos

Otros valores:

- rangos enteros: (2..*), (0..3), etc
- lista de rangos serparados por comas: (1, 3, 5..10, 20..*), (0, 2, 4, 8), etc

- Navegabilidad de la asociación.

  Especifica la capacidad que tiene una instancia de la clase origen de acceder a las instancias de la clase destino por medio de las instancias de la asociación que las conectan.

  Acceder=nombrar, designar o referenciar el objeto para leer o modificar atributos, invocar una operación.

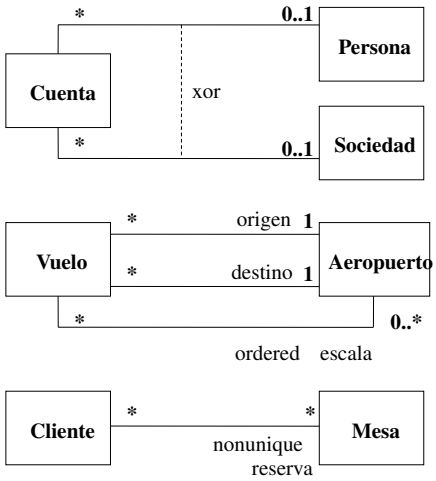
  No confundir dirección del nombre con navegabilidad.

- Un atributo es equivalente a una asociación unidireccional.

- Toda asociación tiene doble significado: Aspecto estático y aspecto dinámico.

  - Son preferibles los nombres estáticos, reservando los nombres dinámicos para nombres de operaciones.
  - Una misma asociación permite la invocación de muchas operaciones.

- Restricciones en asociaciones



or exclusivo entre asociaciones

ordenación de los elementos

UML 1.x: una asociación no puede contener tuplas repetidas
UML 2.x: la restricción nonunique en un extremo permite la repetición

Fig. 3.5: Restricciones en asociaciones

- Asociaciones actor-sistema y clase-clase

  - Un mismo concepto puede ser modelado a la vez como actor y como clase:
    - Actor: representa entidades externas al sistema.
    - Clase: representa entidades modeladas dentro del sistema.

Asociaciones reflexivas: Es aquella en la que los dos extremos de la asociación están unidos a la misma clase.

- Conectan dos instancias distintas de la misma clase o incluso una instancia consigo mismo.
- Los nombres de rol son obligatorios.
- No es simétrica.

Asociación n-aria

- Asociación entre N clases, los enlaces conectan N instancias.
  - No permite: dirección del nombre, agregación.
  - Si permite: navegabilidad, clase asociación.
- Multiplicidad engañosa.
  - número permitido de instancias para cualquier posible combinación de instancias de las otras n-1 clases
  - la multiplicidad mínima suele ser 0
  - efecto "rebote del uno": la multiplicidad mínima 1 (o superior) en un extremo implica que debe existir un enlace (o más) para cualquier posible combinación de instancias de los otros extremos
- Se puede sustituir la asociación n-aria por una clase simple, cuyas instancias representan enlaces.
- Se pierden las multiplicidades originales.
- Clase-asociación
  - Tiene todas las propiedades de una clase y de una asociación:
    - Atributos, operaciones y asociaciones con otras clases.
    - Conexión entre clases que especifica enlaces entre ellas.
    - Multiplicidad, navegabilidad, agregación...
  - Tiene nombre único.

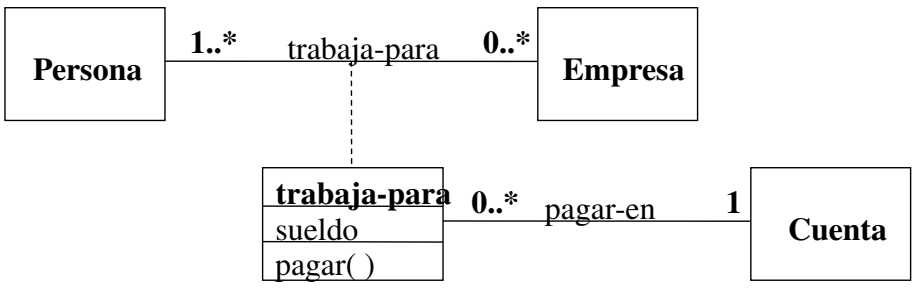


Fig. 3.6: Clase-asociación

- Se puede transformar en clase intermedia, cuyas instancias representan enlaces.
- Las multiplicidades originales se cruzan.

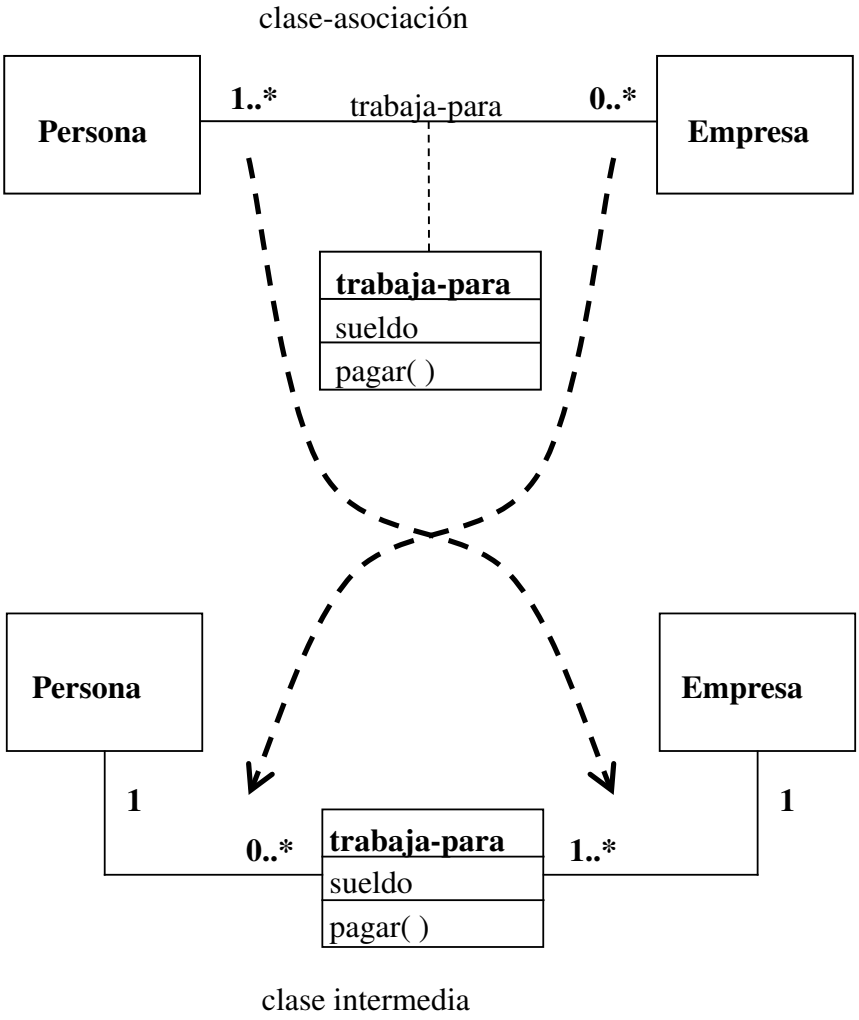


Fig. 3.7: Multiplicidad Clase-asociación

Generalización y clasificación

- Principio de sustitución:
  - Extensión: todos los objetos de la subclase son también de la superclase.
  - Intensión: la definición de la superclase es aplicable a la subclase.
- Generalización: clase-clase.
- Clasificación: objeto-clase.

Generalización y especialización

- Generalizar es identificar las propiedades comunes (atributos, asociaciones, operaciones) de varias clases y representarlas en una clase más general denominada superclase.
- Especializar es capturar las propiedades específicas de un conjunto de objetos dentro de una clase dada, que aún no han sido distinguidas en ella, y representarlas en una nueva clase denominada subclase.
- Es una relación pura entre clases:
  - No tiene instancias, ni multiplicidad.
  - La subclase hereda todas las propiedades de la superclase.
  - Las propiedades heredadas de la superclase no se representan en la subclase (a menos que sean operaciones redefinidas).
  - Toda generalización induce una dependencia subclase Æ superclase

Subclase vs. Atributo

- Atributo: Para propiedades cambiantes o rangos de valores muy altos.
- Subclase: Para propiedades fijas con valores enumerados, especificación.
- Permite modelar un cambio de propiedad como una reclasificación del objeto.

Agregación

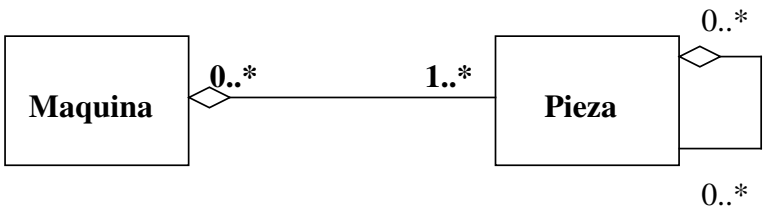


Fig. 3.8: Agregación

- Es un tipo especial de asociación que representa una relación todo-parte, transitiva y asimétrica.
    - impone ninguna restricción especial sobre la multiplicidad

Composición

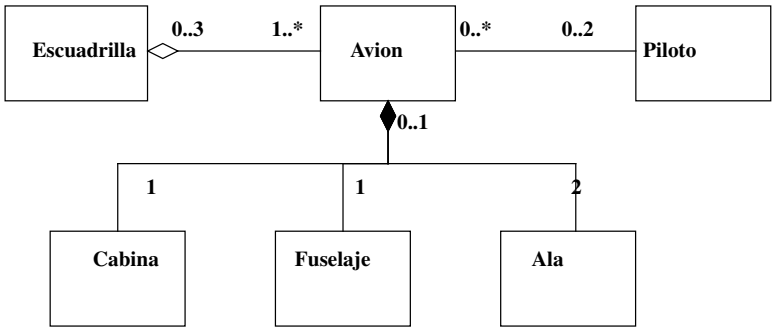


Fig. 3.9: Composición

- Es un tipo especial de agregación no compartida
    - la multiplicidad solo puede ser 0..1 o 1..1
    - el todo es responsable de la existencia y almacenamiento de las partes
    - propagación de las operaciones de copiado y borrado

Diagrama de clases: Captura y especifica y vocabulario del sistema:

- elementos: clases, atributos, operaciones...
- relaciones: asociaciones, generalizaciones, composición, agregación...
- estructura del sistema, fundamento de su comportamiento.

Sugerencias para mejorar la comunicación:

- nombres adecuados: clases, atributos, operaciones, asociaciones, roles
- distribución espacial de los elementos
- evitar cruces de líneas
- distinto nivel de detalle según el propósito y nivel de abstracción

Diagrama de objetos

- ilustra la estructura del sistema mediante situaciones particulares.
- Muestra objetos, valores de atributos y enlaces.

4. TEMA 3 - ARQUITECTURA

La definición que mejor lo transmite:

- La arquitectura del software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y con el entorno, y los principios que orientan su diseño y evolución.

Es como vamos a organizar los componentes/bloques del sistema, la relación entre ellos y también su relación con el entorno.

Modelo 4+1: Las vistas son como dimensiones, nos dan una aproximación distinta.

- Vista lógica (conceptual).
  - Diagramas de clases, modelado, y asociaciones, relaciones entre clases.
  - El propósito es especificar la arquitectura de información del sistema que se desea construir, mediante un conjunto de diagramas de clases adecuadamente explicados.
  - El modelo de información, o modelo conceptual, debe estar justificado a partir de los requisitos. No tiene sentido que en él aparezcan clases, atributos, operaciones y otros elementos que no hayan aparecido anteriormente en los requisitos. Igualmente, no tiene sentido que en los requisitos se mencionen conceptos importantes que no aparezcan reflejados de ninguna manera en el modelo conceptual.
  - El vocabulario del modelo conceptual define todos los términos significativos y específicos del problema que parecen en los requisitos. Es un puente importante que vincula los requisitos con el modelo conceptual.
- Vista de desarrollo (implementación).
  - Organización del software.
  - Habla de los bloques/componentes del sistema y sus relaciones.
  - Interesados: Programadores.
  - Diagramas de componentes y relaciones de uso.
  - Define la descomposición del sistema en subsistemas y componentes, y se especifican las dependencias entre los distintos componentes que hayan resultado de la descomposición

- Los componentes se refieren a:
  - Elementos estático y estructurales del sistema que representa una funcionalidad concreta o un conjunto de ellas utilizados en la implementación: funciones, librerías, etc.
  - Conjunto de datos usados en la implementación: ficheros, bases de datos, etc.
- La arquitectura de desarrollo se representa a través de los diagramas de componentes.
- Componentes vs. Clases
  - Se parecen a las clases en que tienen nombres, realizan interfaces, pueden participar en relaciones
  - Pero se diferencian en que las clases son abstracciones lógicas (se instancian como objetos) y los componentes son fragmentos físicos.
- Componentes e interfaces
  - Definen su comportamiento en base a interfaces requeridas y ofertadas.
  - Unos componentes implementan las interfaces y otros acceden a los servicios proporcionados por esas interfaces.
- Estas relaciones se pueden mostrar con dos tipos de notación: icónica o expandida.

- Vista de proceso (ejecución). No lo veremos.
- Vista física (despliegue). No lo veremos.
- Vista de casos de uso. La hemos visto, es del modelado conceptual.

Diseñar una arquitectura: pasos

- Buscar los patrones más apropiados que den el soporte requerido para alcanzar los atributos de calidad deseados
  - Basarse en Arquitecturas de Referencia reconocidas por tanto por la academia como por la industria
    - Implementaciones conocidas, de amplia difusión y uso
    - Buena documentación
  - Reconocer el tamaño de la aplicación objetivo
    - Aplicaciones pequeñas: Pocos patrones requeridos
    - Aplicaciones grandes: Mezcla de varios patrones

- Debe justificarse teniendo en cuenta:
  - Los criterios: simplicidad, extensibilidad, modificabilidad, eficiencia.
  - Los requisitos no funcionales.
  - Otros que se consideren relevantes: habilidades técnicas, costes.

- Identificación de módulos: subsistemas y componentes
  - La clasificación de requisitos por áreas temáticas puede también proporcionar pistas importantes para esta descomposición.
  - Asignación de componentes
    - Identificar los componentes principales
    - Identificar como los componentes se ajustan a los patrones
    - Identificar dependencias entre ellos
    - Identificar las interfaces y los servicios que cada componente soporta

Como lograr una buena descomposición modular:

- Buen diseño: Bajo acoplamiento y Alta cohesión.
- Acoplamiento: El número de líneas que hay entre dos subsistemas distintos.
- Cohesión: La relación entre los componentes de un subsistema. Si hay relaciones, está muy cohesionado.
- Ejemplo

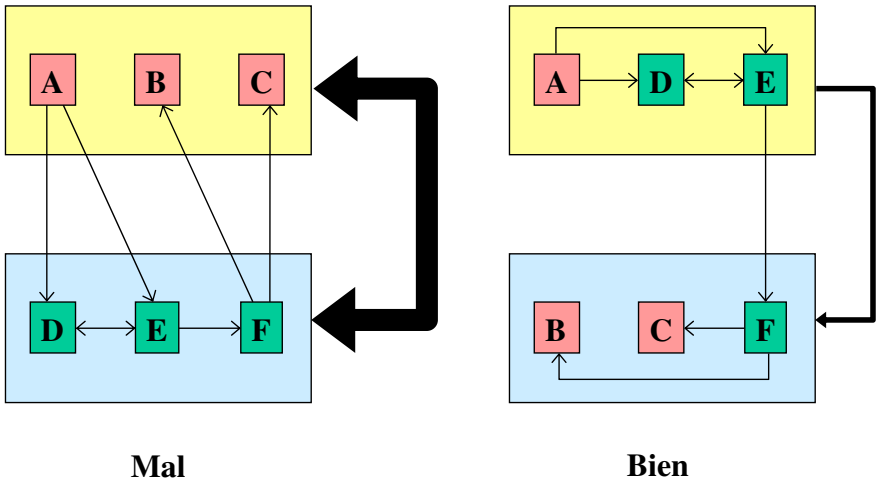


Fig. 4.1: Descomposición modular

Criterios para la selección de una arquitectura:

- Criterios clásicos:
  - Extensibilidad: Que puede evolucionar, se facilita la adición de nuevas características.
    - Hace más complejo el diseño.
    - Mayor grado de abstracción.
    - Distinguir entre opcionales y deseables es muy útil.
  - Modificabilidad: facilitar el cambio de requisitos.
  - Simplicidad: hacer fácil de entender, hacer fácil de implementar.
  - Eficiencia: Lograr alta velocidad o pequeño tamaño.
- Otros criterios: Reúso, Escalabilidad, Coste, Requisitos no funcionales...

Componentes:

- Definición:
  - Partes modulares de un sistema (pieza ejecutable de software)
  - Independientes entre sí.
  - Autocontenidos, encapsulan las estructuras que contienen.
  - Los elementos encapsulados en un componente se comunican con otros elementos de otros componentes mediante interfaces.
  - Interfaces: En un diagrama de componentes, documentan las relaciones y dependencias en una arquitectura de software.
  - Modelan los sistemas de cara a la implementación.
- Componentes: En UML es una definición muy amplia, se utiliza para denominar varias partes del sistema, como bases de datos, paquetes, archivos y bibliotecas. También componentes especializados relacionados con los ámbitos y procesos.

Se usan diagramas de componentes por:

- Visión general del sistema y documenta la organización de los componentes del sistema y sus relaciones y dependencias mutuas.
- Visión orientada a la ejecución, es decir, dan al desarrollador información.
- Especificación de arquitecturas de software y al división de sistemas en subsistemas.

- Facilitan la gestión del desarrollo de programas.
- Reutilización: Permiten visualizar de forma clara que bloques modulares se pueden utilizar varias veces en varios puntos de una arquitectura.

Los componentes se representan con «component» y la cajita con dos rayas.

Las relaciones son de dependencia, apunta del que depende. De tal manera que los cambios del destino afectan al origen de la relación.

Representación de elementos de diagramas de componentes.

- Usaremos: Componente, Paquete (subsistema), Interfaz ofrecida, Interfaz requerida y Relación.

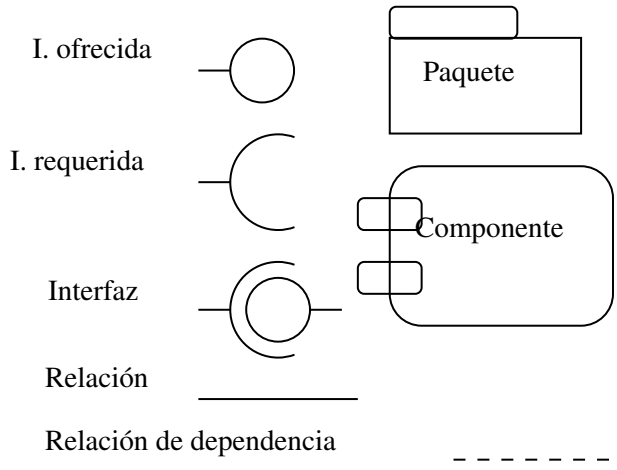


Fig. 4.2: Elementos diagrama de componentes

Patrones

- Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información.
- Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto.
- Los patrones de diseño de software ayudan a diseñar una aplicación. Un patrón es la solución replicable a cierto problema.
- En una arquitectura de aplicaciones habrá:
  - servicios de frontend: se refiere a la experiencia del usuario con la aplicación
  - servicios de backend: implica proporcionar acceso a los datos, los servicios de negocio y otros sistemas actuales que permiten el funcionamiento de la aplicación.

- Monolito
    - Los monolitos son otro tipo de arquitectura asociado con los sistemas heredados.
    - Antes las aplicaciones se escribían como una sola unidad de código, en la que todos los elementos compartían los mismos recursos y espacio de memoria.
    - Son pilas de aplicaciones únicas que contienen todas las funciones dentro de cada aplicación. Tienen conexión directa, tanto en la interacción entre los servicios como en la manera en que se desarrollan y distribuyen.
    - Esto implica que al actualizar o ampliar un solo aspecto de una aplicación monolítica, habrá una repercusión en toda esa aplicación y en la infraestructura subyacente.
- Cliente-servidor
    - Donde el software reparte su carga de cómputo en dos partes independientes, pero sin reparto claro de funciones.
- Modelo Vista Controlador
    - Es el conocido MVC, que divide una aplicación interactiva en tres partes (modelo, vista, controlador) encargadas de contener la funcionalidad, mostrar la información al usuario y manejar su entrada. Este patrón de arquitectura de software separa los datos y la lógica de negocio de una aplicación de su representación.
        - El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). En los frameworks actuales normalmente representa una entidad del diagrama entidad-relación.
        - La Vista: presenta el modelo (información y lógica de negocio) en un formato adecuado para que un usuario pueda interactuar (usualmente la interfaz de usuario).
        - El controlador: es el intermediario entre la vista y el modelo, su función consiste en controlar el flujo de datos, responder a eventos (usualmente provocados por los usuarios) e invocar peticiones al modelo.
    - Maestro esclavo
        - Suele utilizarse para replicaciones en la base de datos (la maestra es la fuente autorizada y las esclavas se sincronizan con ella). Estas dos partes distribuyen el trabajo y calculan el resultado final de toda la actividad que realizan dichos esclavos. Este patrón es una arquitectura fundamental

que los desarrolladores utilizan cuando tienen dos o más procesos que necesitan ejecutarse de forma simultánea.

- Igual a igual (pares)
  - Todos los elementos individuales se les denomina ‘pares’, que pueden funcionar tanto como ‘cliente’, como ‘servidor’. Además, pueden ir cambiando su rol con el paso del tiempo.
- En tuberías
  - Arquitecturas de flujo de datos. Esta arquitectura se aplica cuando los datos de entrada son transformados a través de una serie de componentes computacionales o manipulativos en los datos de salida. Típicamente usada en procesamiento de señales y transformación de flujos de datos. Los componentes reciben el nombre de ‘filtros’ conectados entre sí por ‘tuberías’ que transmiten los datos.
- En pizarra
- Es una arquitectura centradas de datos. Algunas veces llamado Blackboard. En el centro de esta arquitectura se encuentra un almacén de datos (por ejemplo, un documento o una base de datos) al que otros componentes acceden con frecuencia para actualizar, añadir, borrar o bien modificar los datos del almacén. Muy utilizada en sistemas expertos, visión artificial, interpretación sensorial.

Interfaz: Manera de comunicarse entre los componentes.

- Conjunto de operaciones que requiere u ofrece un componente.
- Manera en la que un componente se comunica con el exterior u ofrecer nuestros servicios el sistema.
- No necesariamente el flujo de información ofrecida ha recorrido.

- Toda manera de intercambiar información del sistema. Ejem: Botones, pilotitos, enviar datos...
  - Un componente puede ofrecer más de una interfaz.
  - Una interfaz puede ser ofrecida por más de un componente.
- Encapsulamiento: Separación de la interfaz y la implementación.
  - Una clase/componente puede realizar una o varias interfaces.
  - Una interfaz puede ser realizada por una o varias clases.
- Conjunto de operaciones que ofrecen un servicio coherente:
  - no contiene la implementación de las operaciones.
  - una interfaz no puede tener atributos ni asociaciones.
  - analogía a una clase abstracta con todas sus operaciones abstractas: no puede tener instancias directas.

Diseño por contratos: Acuerdo entre partes.

- Condiciones a cumplir entre las interfaces de componentes.
- Se busca que los componentes y sus relaciones sean seguras/robustas.
- Busca la robustez y corrección.
- Precondición: Condiciones que debe cumplir las interfaces requeridas. Inquilino.
- Postcondición: Condición que debe cumplir las interfaces ofrecidas. Propietario.
- Contrato: Documento que describe qué es lo que se espera de un sistema, enfatizando más en el análisis que en el diseño, más en el qué que en el cómo. Lo más común es en forma pre y post. Describe el comportamiento esperado del sistema en cada operación.
- Programación defensiva a errores vs Diseño por contratos:
  - Programación defensiva a errores: Se distribuyen los componentes en distintos equipos de programación sin coordinación entre ellos, lo que provoca que haya redundancias que en conjunto haya más errores, es más difícil de entender en conjunto. Cada uno se protege del siguiente, hay más redundancia y devuelve más difícil.
  - Diseño por contratos: La aproximación de diseño por contrato es mucho mejor que la programación defensiva, se ponen pre y post condiciones en cada parte y de esta manera están más coordinados. Están más acotadas las funciones y los límites entre componentes.

- Cliente y proveedor acuerdan un contrato.
- Semántica:
    - Precondiciones: Cliente debe asegurarse de que se cumple. Proveedor comprueba.
    - Postcondiciones: Proveedor debe asegurarse de que se cumple. Lo que el cliente espera del servidor.
    - Invariantes: Aserción que deben ser ciertas en todo momento.
        - Las operaciones pueden violarlos temporalmente y de modo controlado durante su ejecución.
        - La violación de un invariante es un error grave.
- Herencia:
    - Superclase subcontrata a las subclases para las instancias directas de la subclase.
    - Las instancias de la subclase son también instancias de la superclase.
    - Las invariantes pueden ser más restrictivo en la subclase, pero no menos restrictivo.
    - Regla o principio de subcontratación: la operación redefinida en una subclase debe...
        - debilitar o mantener la precondición.
        - reforzar o mantener la postcondición.

5. PRÁCTICA

Acceso en Drive a las diapositivas Acceso en Drive a las diapositivas