Análisis y diseño de aplicaciones I

UT3 Diseño y UML

1

Diseño

- El Diseño es tanto el **proceso de** definir:
  - Arquitectura
  - Componentes
  - Interfases
  - Otras características del sistema / componente
- Y el **resultado** de ese proceso

2

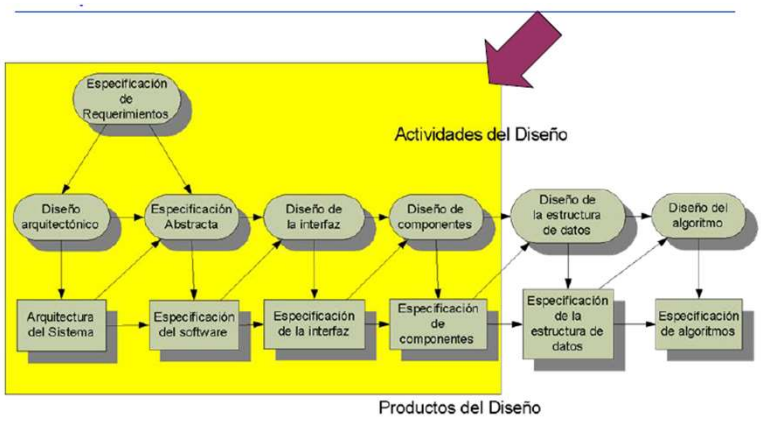
Definiciones

- Sistema
  - Una **entidad lógica**, que tiene un conjunto de responsabilidades u **objetivos** y consiste de hardware, software o ambos
- Subsistema
  - Un sistema que es **parte de un sistema mayor**,
  - Tiene una interface bien definida
- Componente
  - **Cualquier pieza de software o hardware** que tenga definido un rol claro
  - Un componente puede ser aislado, permitiendo que se lo reemplace con un componente diferente que tenga una funcionalidad equivalente
  - Muchos componentes están diseñados para ser **reutilizables**
  - Recíprocamente, otros realizan funciones con un propósito especial
- Módulo
  - Un componente que es definido a nivel de lenguaje de programación.
  - **Pakage / Namespace**

3

Tarea de Aplicación 1
Clasificación inicial de sistemas.

4



Somerville, I- Software Engineering, Cap. 4

5

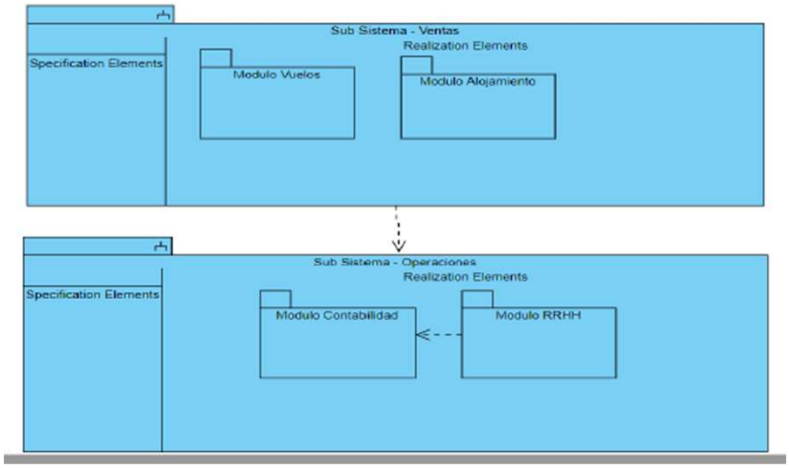
El proceso de diseño del Software

- **Diseño arquitectónico:** Se define la arquitectura (generalmente se utiliza un patrón de arquitectura capas, mvc, monolítico, microservicios, SOA, etc).
- **Especificación abstracta:** Se especifican los subsistemas. Cada **subsistema** realiza **un servicio importante**
  - Contiene objetos altamente acoplados
  - Relativamente independiente de otro subsistemas
  - Generalmente se descompone en módulos aunque se puede descomponer en subsistemas más pequeños
- **Diseño de la interfaz:** Se describen las interfaces de los subsistemas.
- **Diseño de los componentes:** Se descompone cada subsistema en componentes.

6

El proceso de diseño del Software

- Ejemplo Agencia de viajes
- Definimos los componentes dentro de los subistemas



7

Principios del Diseño

- **Dividir y Conquistar:** Tratar con algo grande de entrada es normalmente mucho mas difícil que si se hace con cosas mas pequeñas
- **Incrementar la Cohesión:** Todo lo que sea posible. Un subsistema o módulo tiene un alto grado de cohesión si mantiene juntas las cosas que están relacionadas y afuera las restantes.
- **Reducir el Acoplamiento:** Todo lo que sea posible. El acoplamiento ocurre cuando existe interdependencia entre un módulo y otro
- Mantener el **nivel de abstracción tan alto** como sea posible. Asegúrese que el diseño **oculte** o difiera consideraciones de detalle, reduciendo por tanto la complejidad

8

Principios del Diseño

- **Incrementar la Reusabilidad** donde sea posible. Diseñe los variados aspectos de su sistema de tal forma que pueda ser usado nuevamente en otros contextos. Reutilización de los diseños y códigos existentes cuando sea posible.
- **Diseñe para ser flexible**. Activamente anticipe los cambios que el diseño pueda tener en el futuro, y prepararse para ello
- **Anticipe la obsolescencia.** Planifique los cambios en la tecnología o el entorno de tal manera que el software continúe ejecutándose o sea fácilmente cambiado. Por ej. Diseñar pantallas responsive (meeeeh).

9

Cohesión

- Un subsistema o módulo tiene un alto grado de cohesión si mantiene "unidas" cosas que están relacionadas entre ellas y mantiene fuera el resto,
  - Un módulo cohesivo lleva a cabo una sola tarea dentro un procedimiento de software
- **Objetivo**
  - Diseñar servicios robustos y altamente cohesionados cuyos elementos estén fuerte y genuinamente relacionados entre sí
- **Ventajas**
  - Favorece la compresión y el cambio de los sistemas
- El objetivo es entonces **maximizarla cohesión**

10

Acoplamiento

- "Acoplamiento es la medida de la fortaleza de la asociación establecida por una conexión entre módulos dentro de una estructura de software"
  - Depende de la complejidad de interconexión entre los módulos, el punto donde se realiza una entrada o referencia a un módulo y los datos que se pasan a través del interfaz.
  - Es una medida de la interconexión entre módulos dentro de una estructura del software
  - Un bajo acoplamiento indica un sistema bien dividido y puede conseguirse mediante la eliminación o reducción de relaciones innecesarias.

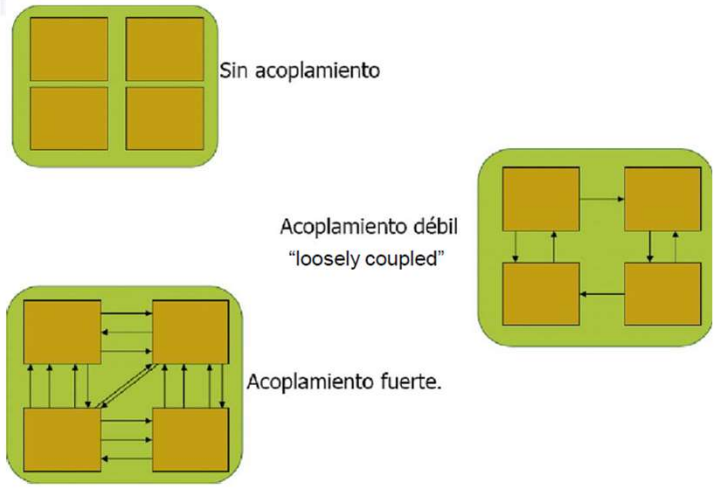
11

Acoplamiento

- Se produce una situación de acoplamiento cuando un elemento de un diseño depende de alguna forma de otro elemento del diseño.
  - **El objetivo es conseguir un acoplamiento lo más bajo posible**.
  - Conseguir que cada componente sea tan independiente como sea posible

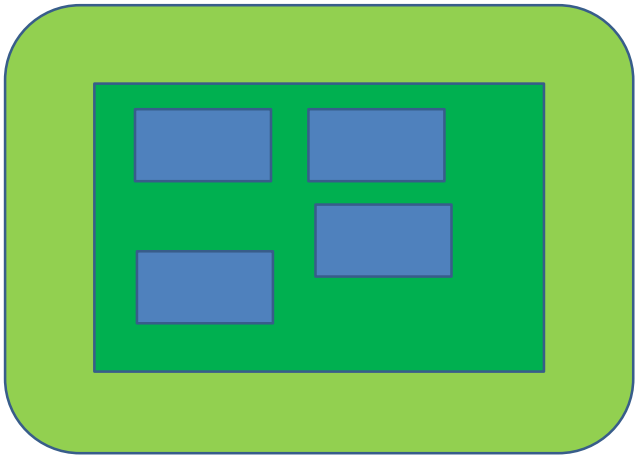
12

UCU


Sin acoplamiento
Acoplamiento débil
"loosely coupled"
Acoplamiento fuerte.

13

UCU



14

Dependencia entre Componentes

- El acoplamiento depende de varios factores:
  - Referencias hechas de un componente a otro (invocaciones)
  - Cantidad de datos pasados de un componente a otro (parámetros)
  - El grado de control que un componente tiene sobre el otro (ej. Composición entre clases)

15

Clases de Acoplamiento

- Fuertemente acoplados
  - Si los módulos utilizan variables compartidas o intercambian información de control
- Débilmente acoplados
  - Garantizar que los detalles de la representación de datos están dentro de un componente.
  - Interfaz con otros componentes mediante lista de parámetros.
  - Información compartida limitada a aquellos componentes que necesitan acceder a la información
  - Evitar compartir información de forma global.

16

Journey map

17

Journey map

- **Descubrimiento** (Awareness)
  - Esta primera fase es cuando ***un consumidor descubre el producto***. En este estado, las características del producto o servicio no son tan importantes.
  - Estamos en una fase de información del usuario, y no se intenta incitar al usuario a comprar, ***simplemente se le informa*** de que hay un producto para una necesidad que este puede tener.
- **Consideración** (Consideration)
  - La fase de consideración es el momento en el que ***el consumidor quiere realizar una compra***, y considera diferentes opciones para llevarla a cabo. Esta es la fase de valoración.
  - En esta fase, es cuando el usuario debe conocer la marca y ser consciente de la existencia de esta para tenerla en cuenta para realizar una posible compra. Para ello, aquí sí que hay que ***informar a los consumidores de las características del producto, así como los puntos fuertes***.
  - A diferencia de la publicidad informativa de la fase de descubrimiento, en este momento hay lanzar un mensaje más directo sobre las ventajas de nuestro producto con el objetivo de incentivarlo a comprar.

18

Journey map

- **Compra** (Purchase)
  - La fase de compra es cuando el ***usuario ya ha tomado la decisión de compra y decide llevarla a cabo***. En este proceso, tener un canal online que no ralentice o frene este proceso es importante, así como un personal en tienda cualificado para que la experiencia de compra en tienda sea buena.
- **Retención** (Retention)
  - Esta es la primera fase dentro del servicio post-venta donde se busca ***mantener la satisfacción del cliente***.
    El objetivo de esta fase es mantener la relación con el cliente y propiciar que este repita compras y se fidelice.
- **Recomendación** (Advocacy)
  - Tras una experiencia de compra satisfactoria, es posible que los ***clientes nos ayuden a mejorar la propia imagen*** de marca de la empresa e impactar en fases anteriores para otros usuarios.
    Las redes sociales, ***las valoraciones y el boca a boca*** son las partes más importantes de la fase de recomendación.

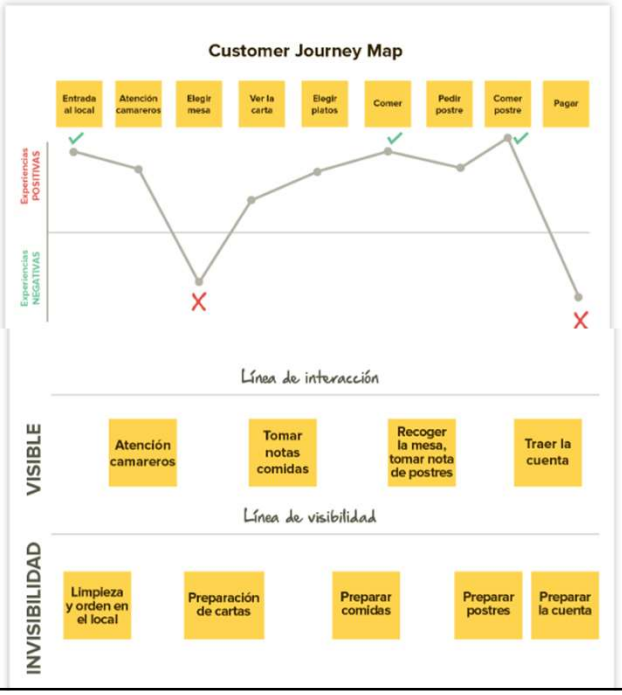
19

Construcción

- En primer lugar, se dibuja un gráfico en el que el Eje X muestra las **fases por la que pasa el cliente** a lo largo del tiempo y en el Eje Y, se define **cómo siente las experiencias**, desde la más negativa, en rojo, hasta la más positiva, en verde
  - **Puntos positivos**: Entrada al local, comida y comer postre.
  - **Puntos negativos**: Elección de la mesa y pagar.
  - ***Stops* o puntos críticos**: El primer vistazo al local, la carta y la comida.

20

Cons



21

Tarea de Aplicación 2

Acoplamiento y cohesión

22

Agenda

- Contexto… Qué es UML?
- Diagrama de CU
- Diagramas de clase
- Diagramas de secuencia
- Diagramas de actividad
- Diagramas de paquetes/componentes/despliegue

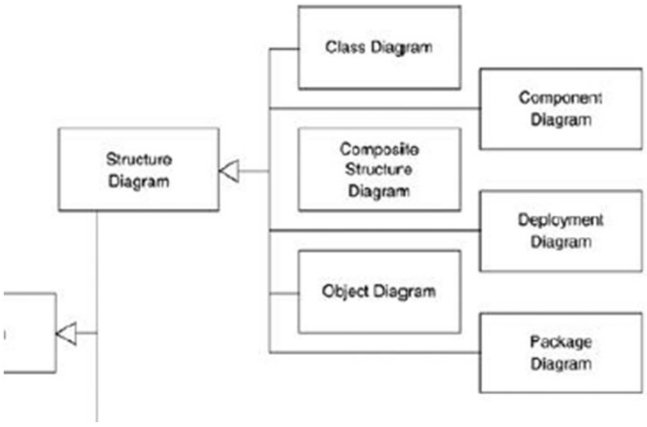
23

Contexto

- El **Lenguaje Unificado de Modelado** (UML) es una familia de notaciones gráficas respaldada por un único metamodelo que ayuda a describir y diseñar sistemas de software, en particular aquellos construidos utilizando el estilo orientado a objetos (OO).
- El UML es un término que puede significar cosas diferentes para diferentes personas, dependiendo de su experiencia y percepción de lo que hace que un proceso de ingeniería de software sea efectivo.
- A pesar de que los lenguajes de modelado gráfico han existido en la industria del software por mucho tiempo, hay mucho debate sobre su papel y el papel del UML en particular.
- El UML es un estándar relativamente abierto controlado por el Object Management Group (OMG), un consorcio abierto de empresas que se formó para construir estándares que apoyaran la interoperabilidad de los sistemas orientados a objetos. Desde su aparición en 1997, el UML ha unificado muchos lenguajes de modelado gráfico orientados a objetos y ha sido de gran ayuda para los desarrolladores de software.

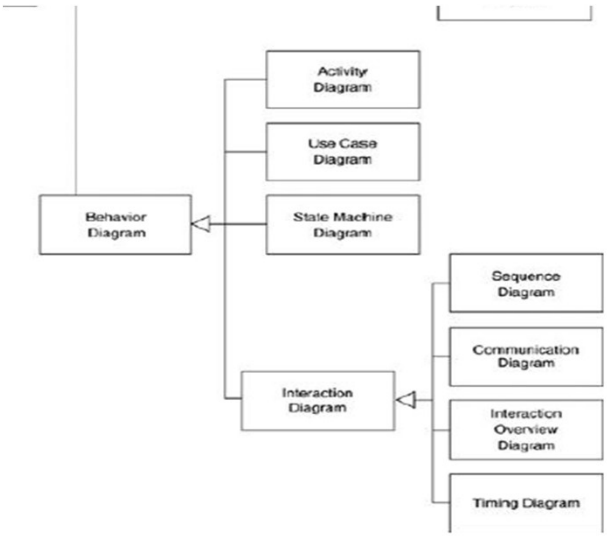
24

Tipos de diagramas



25

Tipos de diagramas



26

Diagrama de casos de uso

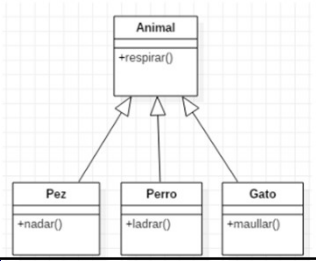
- Los casos de uso proporcionan una narrativa de cómo los actores utilizan el sistema.
  - **Representar los requisitos funcionales**.
  - Representar los **actores** que se comunican con el sistema. Normalmente los actores del sistema son los **usuarios** y **otros sistemas externos** que se relacionan con el sistema. En el caso de los usuarios hay que entender el actor como un "perfil", pudiendo existir varios usuarios que actúan como el mismo actor.
  - **Representar las relaciones** entre requisitos funcionales y actores.

27

Diagrama de clase

- Los diagramas de clases UML son ampliamente utilizados y describen los **tipos de objetos** en un sistema, así como las **relaciones** estáticas entre ellos.
- Incluyen **propiedades, operaciones y restricciones** aplicables a las conexiones entre objetos.

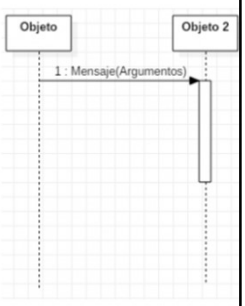
- Los diagramas de clase **pueden tender a volverse incoherentes** a medida que se expanden y crecen. Es mejor evitar la creación de diagramas grandes y **dividirlos** en otros más pequeños que se puedan vincular entre sí más adelante.
- Usando la notación de clase simple, puedes crear rápidamente **una visión general de alto nivel** de su sistema.
- Cuantas más líneas se superpongan en sus diagramas de clase, más abarrotado se vuelve…
- Usa **colores** para agrupar módulos comunes. Diferentes colores en diferentes clases ayudan al lector a diferenciar entre los diversos grupos.



28

Diagrama de secuencia

- Su objetivo es representar **el intercambio de mensajes entre los distintos XXX** del sistema para cumplir con una funcionalidad. Define, por tanto, el comportamiento dinámico del sistema de información.
- También se suele construir para comprender mejor el diagrama de clases, ya que el diagrama de secuencia muestra como objetos de esas clases interactúan haciendo intercambio de mensajes.



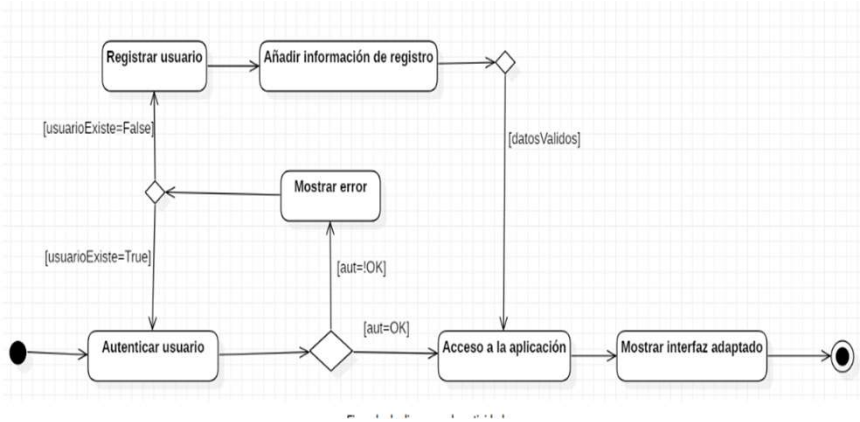
29

Diagrama de actividad

- Los diagramas de actividades muestran una secuencia de acciones, un **flujo de trabajo que va desde un punto inicial hasta un punto final**.
- La finalidad de este diagrama es modelar el **workflow** de una actividad a otra, pero sin tener en cuenta el paso de mensajes entre ellas.
- También es utilizado para modelar las actividades, que podemos asemejar a requisitos funcionales de negocio, por lo que este diagrama tendrá una influencia mayor a la hora de comprender el negocio o sus funcionalidades que en la propia implementación. Hay que tener en cuenta que este diagrama ofrece una visión a alto nivel.
  - Documentar los **requisitos** de negocio.
  - Modelar el **flujo de trabajo** entre actividades o/y entre subsistemas.
  - Comprender a **alto nivel** las funcionalidades del sistema de información.

30

Diagrama de actividad



31

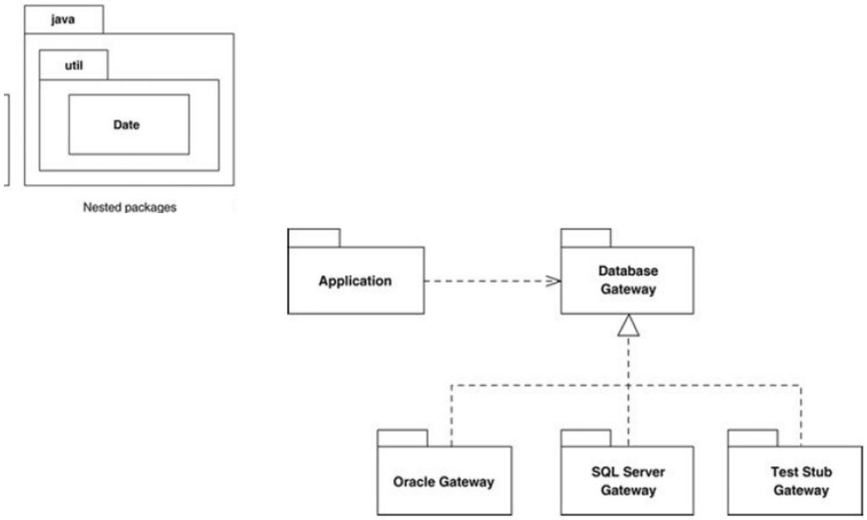
Diagrama de paquetes

- Los diagramas de clases sirven para representar la estructura más básica de un sistema.
- A medida que estos empiezan a crecer, necesitamos una forma más práctica de visualizar las relaciones y dependencias.
- Los diagramas de paquetes nos permiten agrupar por algún concepto de con un mayor nivel de abstracción.

32

Diagrama de paquetes

UCU

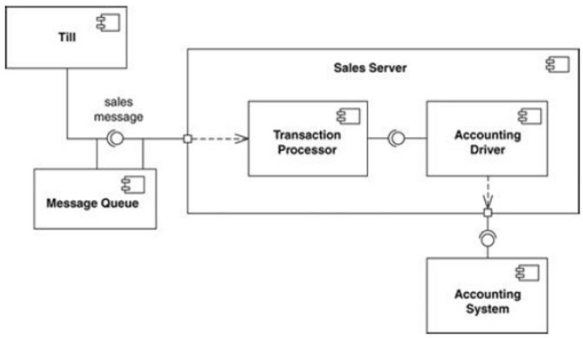


33

Diagrama de componentes

UCU

- Podemos verlo como una agrupación de módulos, por algún criterio.
- Sirve para documentar las relaciones entre el sistema a través de las interfaces.



34

Diagrama de deploy

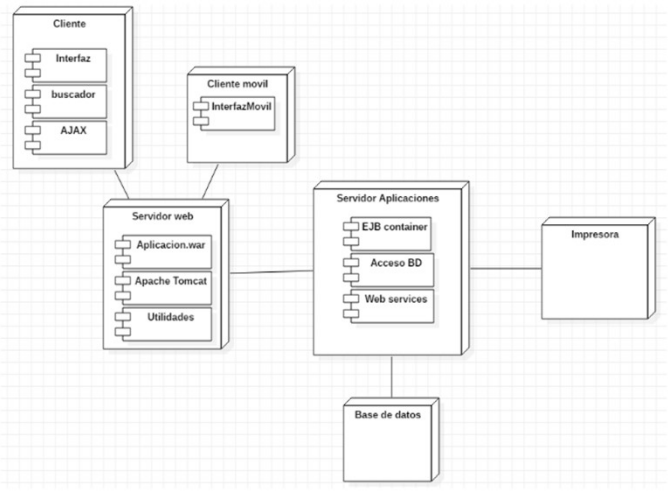
UCU

- Suele ser utilizado junto con el diagrama de componentes (incluso a veces con el diagrama de paquetes) de forma que, juntos, dan una visión general de como estará desplegado el sistema de información.
- El diagrama de componentes muestra que componentes existen y como se relacionan mientras que el diagrama de despliegue es **utilizado para ver como se sitúan estos componentes lógicos en los distintos nodos físicos**.
- Como prácticamente todos los diagramas de UML, puede ser utilizado para representar aspectos generales o muy específicos, siendo utilizado de forma más común para aspectos generales.

35

Diagrama de deploy

UCU



36

Tarea de Aplicación 3

Tipos de diagramas

UCU

37

Bibliografía

UCU

- Ingenieria del Software 7ma. Ed. - Ian Sommerville.pdf
- UML Distilled - Martin Fowler
- https://diagramasuml.com/
- https://blog.fromdoppler.com/customer-journey-map-como-crear-uno/
- https://www.tcgroupsolutions.com/blog/las-5-etapas-del-customer-journey/

38