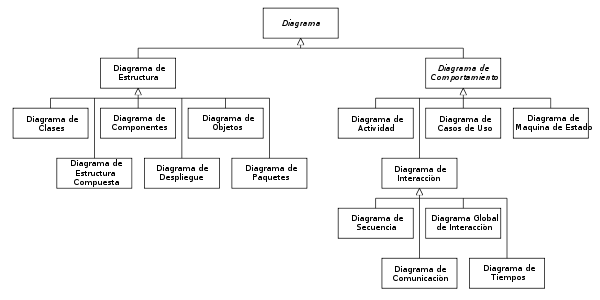
UML (Unified Modelling Language, lenguaje unificado de modelado): lenguaje de modelado de sistemas de software

* Visualizar, especificar, construir y documentar un sistema
* Describe “planos” de sistema, puede ser procesos, funciones del sistema, expresiones de lenguajes de programación, esquemas de bases de datos, etc
* No especifica metodologías o procesos a usar
* Desde 2004 está aprobado como estándar por la ISO
* Antes de UML 1.x
  + Object-Modeling Technique (OMT) para análisis orientado a objetos (James Rumbaugh), Método Booch para diseño orientado a objetos (Grady Booch), método de ingeniería de software orientado a objetos (Ivar Jacobson)
  + Rational Software Corporation les pidió en 1996 que hicieran un lenguaje unificado de modelo. Se presentó un borrador en 1997 y fue adoptado por la Object Management Group (OMG) ese mismo año
* UML 1.x
  + Se integraron conceptos de otros métodos orientados a objetos (MOO) para hacerlo compatible con todos los MOO. Se tuvieron en cuenta otros métodos de la época, para tener amplia cobertura en el dominio de los sistemas en tiempo real
  + Útil desde procesos sencillos a sistemas concurrentes y distribuidos
* UML 2.x
  + Correcciones de defectos y errores
* UML 2.5
  + Dos tipos de diagramas: diagramas estructurales y diagramas de comportamiento. Representan aspectos de las interacciones



* + Diagramas estructurales: estructura estática del sistema y sus partes en diferentes niveles de abstracción
    - **Diagramas de clase:** clases en un sistema, con sus atributos, operaciones y relaciones entre clases. Tienen nombre en la parte superior, atributos en el centro y operaciones en la parte inferior. Las distintas relaciones son distintas flechas
    - **Diagrama de componentes:** relación estructural de los componentes de un sistema de sw. Útil con sistemas complejos con muchos componentes. Los componentes se comunican entre sí mediante interfaces. Las interfaces se enlazan con conectores
    - **Diagrama de despliegue:** hardware del sistema y el software del hardware. Útil cuando la solución se despliega en equipos con diferentes configuraciones
    - **Diagrama de objetos (instancias):** relaciones entre objetos usando ejemplos del mundo real, cómo se verá un sistema en un momento dado. Explican relaciones complejas entre objetos
    - **Diagrama de paquetes:** dependencias entre paquetes que componen un modelo. Muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre agrupaciones. Como un paquete está pensado como un directorio, estos diagramas muestran la descomposición de la jerarquía lógica del sistema
    - **Diagrama de perfiles (UML 2):** no se usa mucho
    - **Diagrama de estructura compuesta (UML 2):** estructura interna de una clase
  + Diagrama de comportamiento: comportamiento dinámico de los objetos en el sistema
    - **Diagrama de actividades:** flujos de trabajo (empresarial u operativo). Se usan como alternativas a los diagramas de máquina de estado
    - **Diagrama de casos de uso:** actores involucrados en un sistema, funciones que necesitan y cómo interactúan estas diferentes funciones
    - **Diagrama de máquina de estados:** describir el comportamiento de objetos que actúan diferente según su estado
  + Diagrama de interacción
    - **Diagrama de secuencia:** cómo interactúan los objetos entre sí y el orden en que se dan las interacciones en un escenario en particular. Los procesos se representan verticalmente y las interacciones son flechas
    - **Diagrama de comunicación:** similar a diagramas de secuencia, el foco está en los mensajes pasados entre objetos
    - **Diagrama de tiempos (sincronización) (UML 2):** comportamiento de objetos en un marco de tiempo dado. Si es un objeto, el diagrama es directo. Si hay varios objetos, se puede usar para mostrar interacciones de objetos en ese período de tiempo
    - **Diagrama global (general) de interacciones (UML 2):** muestran una secuencia de diagramas de interacción y el orden en que suceden

Formas de usar UML

* Sketch
  + Ayuda a comunicar aspectos del sistema
  + Informal y dinámico (ejemplo: en un pizarrón)
  + No se trata de la completitud, sino comunicación
  + En forward engineering, hay detalles que se resuelven en código
  + En reverse engineering, se usa el sketch para explicar cómo funciona alguna parte del sistema
* Blueprint
  + Se trata de completitud, un diseñador lo arma para que un programador lo codifique. Debería tener todas las decisiones de diseño, el programador no debería pensar y hacerlo directamente en código
  + En reverse engineering, dan información detallada del código
  + Necesitan herramientas, CASE (computer-aided software engineering)
    - Forward-engineering tools hacen diagramas y tienen un repositorio para guardar la información
    - Reverse-engineering tools interpretan código fuente, lo suben al repositorio y hacen un diagrama
* Programming language
* Forward engineering: hacer UML antes del código
* Reverse engineering: hacer UML luego del código para entenderlo