

## Metodología del Desarrollo del Software - Apunte Final UML - RUP

### Introducción: ¿Por qué modelamos?

La principal causa es para comprender mejor el sistema que se está desarrollando.

- Modelar permite comprender mejor el sistema que se está construyendo, muchas veces descubriendo oportunidades para la simplificación y la reutilización.
- Modelar nos permite desarrollar sistemas desde diversos puntos de vista a través de distintos modelos.
- Modelar permite documentar decisiones que adoptamos a lo largo del diseño del sistema.
- Modelar proporciona plantillas que nos ayudaran a la construcción de un sistema.
- Modelar nos permite comunicar la estructura deseada y el comportamiento de nuestro sistema. Tanto a integrantes del desarrollo como a los clientes.
- Por último construimos modelos para controlar el riesgo.

¿Por qué modelar sistemas? ¿No sería más fácil sentarnos y simplemente codificar hasta generar un "buen" software?

NO, los modelos ayudan a visualizar como es o queremos que sea un sistema. Al sentarnos a codificar hasta que "salga" algo, se está gastando una gran cantidad de potencia mental del equipo de desarrollo en resolver problemas que fácilmente podrían haber sido descubiertas con solo pensar antes de programar.

Se construyen modelos de un sistema porque no se puede comprender un sistema en su totalidad (hablamos de sistemas grandes y complicados o en sistemas que crecen y se vuelven usualmente de esa manera).

Cuando se piensa en modelar se reduce la cantidad de datos a comprender sin descartar su esencia (retiramos los excesos de la realidad); por eso partimos y comprendemos el problema, centrándonos cada vez en una sola parte del mismo.

Es el viejo dicho de: **Divide y Vencerás**, aplicada al desarrollo de software.

Algo importante que se tiene que mencionar es que los mejores modelos están ligados a la "realidad".

Cualquier sistema informático no trivial se aborda mejor a través de un pequeño conjunto de modelos casi independientes con múltiples puntos de vista para asegurar el éxito del sistema.

Además de eso, los modelos sirven como un tipo de documentación, con el cual, algún otro programador que retome el proyecto para *refactoring* comprenderá fácilmente la forma en que este está estructurado.

### ¿Qué es un modelo?

Es una representación simplificada de la realidad.

### ¿Qué es hacer análisis?

Es un proceso que nos ayuda a mapear en nuestros términos (lenguaje del desarrollador) lo que estaba en lenguaje natural.

### ¿Por qué diseñamos?

- El diseño es el proceso de determinar cual de muchas posibles soluciones es la mejor para lograr lo que se necesita hacer, respetando las restricciones tecnológicas y de presupuesto del proyecto. El diseño escoge un cómo específico para aplicarlo al qué.
- El diseño consiste en decidir la manera en que debe construirse el sistema para satisfacer los requerimientos de los usuarios.

### ¿Alta cohesión y bajo acoplamiento?

Dentro de un modelo, un modulo tiene alta cohesión si todos sus elementos están fuertemente relacionados y son agrupados por una razón lógica, esto significa que todos cooperan para alcanzar un objetivo común que es alcanzar la función del modulo. La cohesión es una propiedad interna de cada modulo, por el contrario el acoplamiento caracteriza las relaciones de un modulo con otros. El acoplamiento mide la interdependencia de dos módulos. Si dos módulos dependen fuertemente uno del otro tienen un alto acoplamiento lo que los vuelve difíciles de analizar, comprender, modificar, testear o reusar en forma separada.

Una estructura modular con alta cohesión y bajo acoplamiento permite ver los módulos como cajas negras.

## ¿Qué es una metodología de desarrollo de software?

Es un proceso organizado para la producción de software. Especifica el ciclo de vida a utilizar, indicando además qué personas deben desempeñar cada rol en el desarrollo de las actividades.

Consiste en una serie de pasos sistemáticos para que los diferentes grupos que participan en un desarrollo posean una buena comunicación.

El **modelo de ciclo de vida** para el desarrollo de software, nos dice el orden en que se realizaran las diferentes etapas (espiral/cascada) y lo que hay que obtener en cada una de ellas a lo largo del desarrollo del proyecto.

Un **proceso de desarrollo de software** es un conjunto de actividades necesarias para transformar los requerimientos del usuario en un sistema software.

Una **etapa** son los diferentes pasos a seguir durante el desarrollo del software. Determinado orden de estos pasos se los conoce como ciclo de vida. Ejemplo captura de requerimientos, análisis del problema, diseño, implementación, testing, mantenimiento.

Un **rol**, es el papel que debe desempeñar una persona en el desarrollo del software, como por ejemplo analista, diseñador, programador, tester, etc.

Un **modelo/diagrama**, es un lenguaje visual que se aplica en cada etapa del ciclo de vida. Permite para visualizar, especificar, construir y documentar.

El tipo de metodología va a depender del sistema que se quiere implementar. Hoy en día existen distintas metodologías que fueron surgiendo de acuerdo a la evolución de los lenguajes de programación:

- Orientados a DATO/FUNCION (ASML, A System Modeling Language).
- Orientados a OBJETOS (UML/RUP, Unified Modeling Language/ Rational Unified Process).
- FORMALES (B/Z/Object Z).
- Métodos Ágiles.

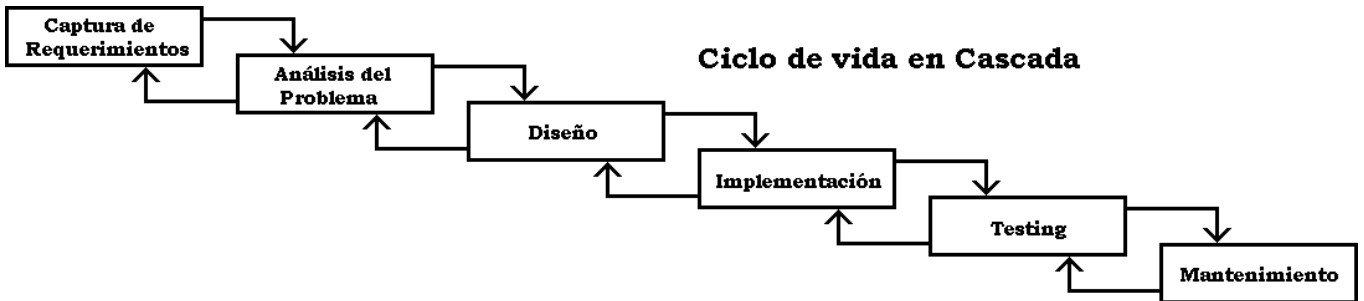
Para desarrollar proyectos grandes y complejos es necesario seguir una metodología. Si un proyecto involucra más de tres personas, nos sería útil para una buena comunicación y buen desempeño a lo largo de todo el proyecto.

Características de la metodología orientada a objetos:

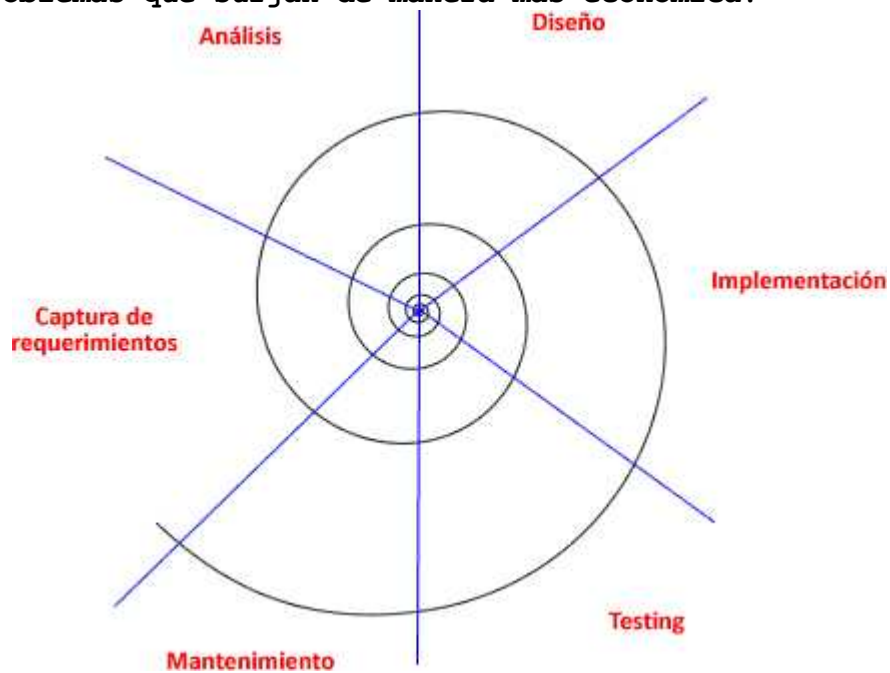
- Énfasis en la abstracción de datos.
- Funciones y datos encapsulados en entidades fuertemente relacionadas.
- Facilidades de mantenimiento y extensión.
- Mapeo directo a entidades del mundo real.

Modelos de ciclo de vida:

**Modelo en cascada:** En un principio fue mucha utilidad, pero el problema es que para pasar de una etapa a la otra había que terminar la primera, produciendo un gran inconveniente si algún cambio era requerido.



**Modelo en espiral:** Está basado en el modelo en cascada. Su desarrollo es incremental. Procesa pequeñas partes de cada etapa, lo que nos permite solucionar problemas que surjan de manera más económica.



Etapas que comprenden los modelos de ciclo de vida:

- **Captura de requerimientos:** el propósito es dejar bien en claro cuál es la necesidad del cliente y los requerimientos.
- **Análisis:** es la captura de requerimientos en el lenguaje del desarrollador. Llevar el lenguaje natural del cliente a un lenguaje formal.
- **Diseño:** se modela una solución del sistema, teniendo en cuenta el ambiente de implementación a utilizar, por ejemplo, si el sistema es centralizado o distribuido, la base de datos a utilizar, lenguaje de programación, performance deseada, etc.
- **Implementación:** se implementa el sistema en el lenguaje elegido.
- **Testing:** se prueba que el sistema funcione correctamente (verificación) y que, además, sea el sistema correcto (validación).
- **Mantenimiento:** es la etapa más difícil de desarrollo del sistema, actualiza y modifica el sistema si surgen nuevos requerimientos.

A la hora de elegir un modelo de ciclo de vida, influyen factores como el tiempo, tamaño del proyecto, comunicación con el cliente, comunicación con el grupo de desarrollo, etc.

## Unified Modeling Language (UML)

### ¿Qué es UML?

UML es un lenguaje de modelado estándar para escribir planos de software. Puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software. Proporciona un conjunto de diagramas que me permite modelar los diferentes aspectos del sistema y su principal objetivos es la comunicación de información.

Características principales:

- Es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.
- El vocabulario y las reglas de un lenguaje como UML indican cómo crear y leer modelos bien formados, pero no dice que modelos se deben crear ni cuando se deberían crear. Esta tarea corresponde al proceso de desarrollo del software.
- Detrás de cada símbolo en la notación de UML hay una semántica bien definida, de esta manera un desarrollador puede escribir un modelo en UML, y otro desarrollador o incluso otra herramienta, puede interpretar ese modelo sin ambigüedad.
- UML está pensado principalmente para sistemas con gran cantidad de software.
- No está limitado al modelado de software.

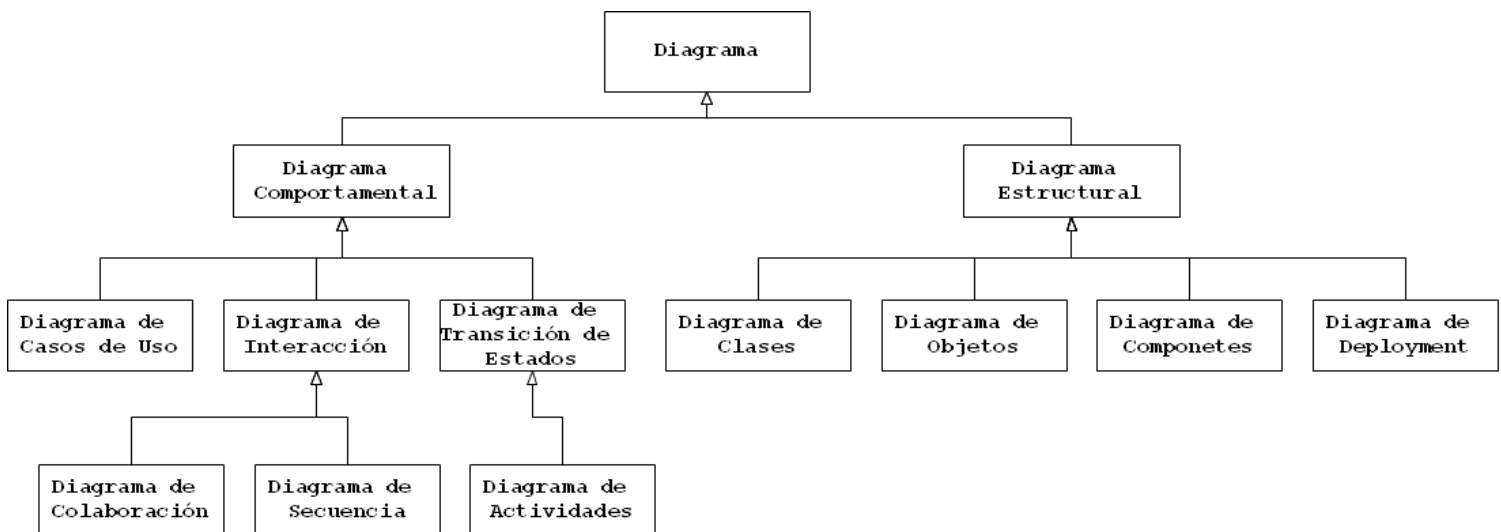
Diagramas soportados por UML:

UML Versión 1.2 (actual 2.0)	
Dinámica/Comportamental	Estática/Estructural
D. Casos de Uso	D. Clases
D. Actividades	D. Objetos
D. Transición de Estados	D. Componentes
D. Interacción <ul style="list-style-type: none"> <li>➤ D. Colaboración</li> <li>➤ D. Secuencia</li> </ul>	D. Deployment (despliegue)

- **Diagrama de Casos de Uso:** son importantes para visualizar, especificar y documentar el comportamiento de un sistema, un subsistema o una clase. Modela las necesidades a satisfacer y los límites del sistema.
- **Diagrama de Actividades:** es un tipo especial de diagrama de transición de estados que muestra el flujo de actividades que se tiene que desarrollar dentro de un sistema para un fin "X".
- **Diagrama de Transición de Estados:** muestra una maquina de estados que consta de estados, eventos, transiciones y actividades. Son especialmente importantes en el modelado del comportamiento de una interfaz, una clase o una colaboración.
  - Un estado es una situación durante la vida de un objeto.
  - Un evento es un estímulo que provoca un cambio de estados.
  - Una actividad es una ejecución computacional no atómica.
  - Una acción es una ejecución computacional atómica.

- **Diagrama de Interacción:** es un diagrama que muestra una interacción entre objetos que se mandan mensajes y sus relaciones.
- **Diagrama de Colaboración:** es un tipo de diagrama de Interacción que resalta la organización estructural de los objetos que envían y reciben mensajes.
- **Diagrama de Secuencia:** es un tipo de diagrama de Interacción que resalta el orden temporal de los mensajes enviados entre los distintos objetos.
- **Diagrama de Clases:** muestra un conjunto de clases colaboración e interfaces así también como sus relaciones. Son los más utilizados en sistemas orientados a objetos y son la base para los diagramas de objetos, componentes y de despliegue. Representan elementos lógicos de un sistema.
- **Diagrama de Objetos:** muestra un conjunto de objetos y sus relaciones. Representan una instancia del diagrama de clases. Es sacarle una foto al sistema en tiempo de ejecución.
- **Diagrama de Componentes:** Muestra la organización y dependencia de un conjunto de componentes. Modela los aspectos físicos de los sistemas orientados a objetos esto implica ejecutables, bibliotecas, tablas, archivos, documentos, etc.
- **Diagrama de Despliegue:** muestra la configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. Modela la topología del hardware en la que se ejecuta el sistema.

En la siguiente ilustración, se mapea la herencia de diagramas soportados por UML:



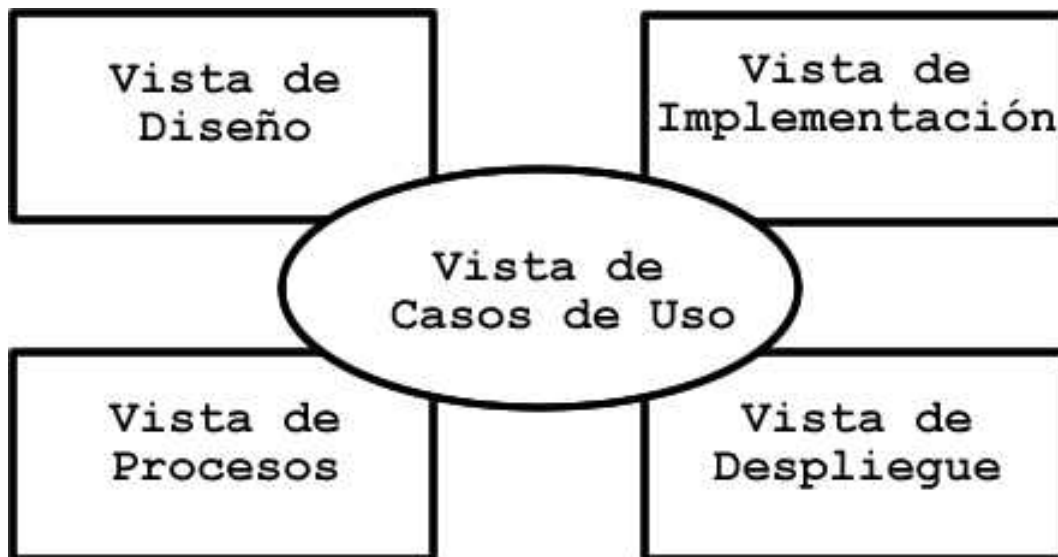
La visualización, especificación, construcción y documentación de un sistema con gran cantidad de software requiere que el sistema sea visto desde varias perspectivas.

La arquitectura de un sistema es quizás el artefacto más importante que puede emplearse para manejar estos diferentes puntos de vista y controlar el desarrollo iterativo e incremental de un sistema a lo largo de su ciclo de vida.

La arquitectura toma decisiones significativas sobre:

- La organización de un sistema de software.
- La selección de elementos estructurales y sus interfaces a través de los cuales se constituye el sistema.
- Su comportamiento, como se especifica en las colaboraciones entre esos elementos.
- La composición de esos elementos estructurales y de comportamientos en subsistemas progresivamente más grandes.
- El estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos y sus interfaces, sus colaboraciones y su composición.

Como se ilustra en la siguiente figura, la arquitectura de un sistema con gran cantidad de software puede describirse mejor a través de cinco vistas interrelacionadas. Cada vista es una proyección de la organización y la estructura del sistema, centrada en un aspecto particular de ese sistema.



Vista de Casos de Uso: Describe el comportamiento del sistema tal y como es percibido por los usuarios finales.

Vista de Diseño: Soporta principalmente los requerimientos funcionales del sistema.

Vista de Procesos: Comprende principalmente el funcionamiento, capacidad de crecimiento y rendimiento del sistema.

Vista de Implementación: Se preocupa de los componentes y archivos que deben ensamblarse para producir un sistema en ejecución.

Vista de Despliegue: Se preocupa de la distribución, entrega e instalación de las partes que constituyen el sistema físico.

Cada una de estas 5 vistas puede existir por sí mismas, de forma que diferentes usuarios puedan centrarse en las cuestiones de la arquitectura del sistema que más les interesen.

## Rational Unified Process (RUP)

### ¿Qué es RUP?

RUP es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

RUP soporta diagramas UML para validar y verificar los sistemas que se estén desarrollando.

Características principales:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
- Pretende implementar las mejores prácticas en Ingeniería de Software.
- Desarrollo iterativo e incremental.
- Administración de requisitos.
- Uso de arquitectura basada en componentes.
- Control de cambios.
- Modelado visual del software.
- Verificación de la calidad del software.
- Diseñado para ser flexible y extensible.

El RUP es un producto de Rational (IBM). Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

**Dirigido por los casos de uso** significa que los casos de uso se utilizan como artefacto básico para establecer el comportamiento deseado del sistema, para validar y verificar la arquitectura del sistema, para las pruebas y para la comunicación entre las personas involucradas en el proyecto.

**Centrado en la arquitectura** significa que la arquitectura del sistema se utiliza como un artefacto básico para conceptualizar, construir, gestionar y hacer evolucionar el sistema en desarrollo.

Un **proceso iterativo** es aquel que involucra la gestión de un flujo de ejecutables del sistema.

Un **proceso incremental** es aquel que involucra la continua integración de la arquitectura del sistema para producir esos ejecutables, donde cada ejecutable incorpora mejoras incrementales sobre los otros.

Una iteración es un conjunto bien definido de actividades, con un plan y unos criterios de evaluación bien establecidos, que acaba en una versión, bien interna o externa.

Ciclo de vida:

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

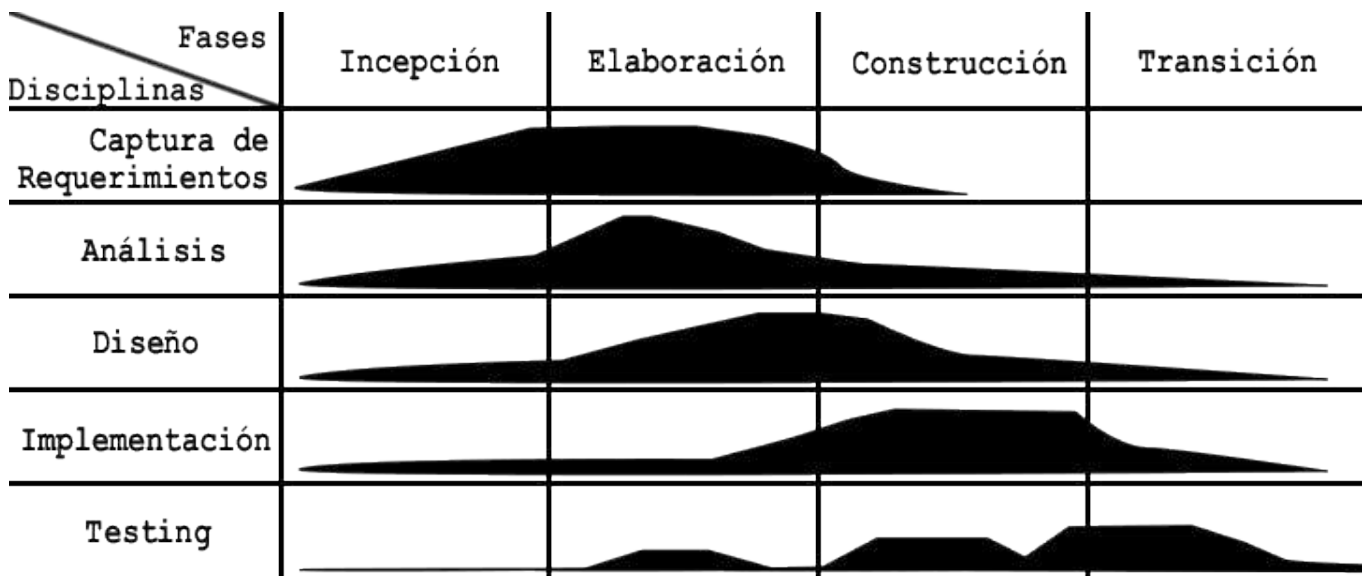
RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades.

UP presenta 2 dimensiones:

- Tiempo
- Componentes

Básicamente me dice qué tengo que hacer en cada etapa.

En la Figura muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.



Es importante aclarar que en cada etapa, las iteraciones se basan en el modelo de ciclo de vida en cascada

### Descripción de cada fase:

#### **Incepción**

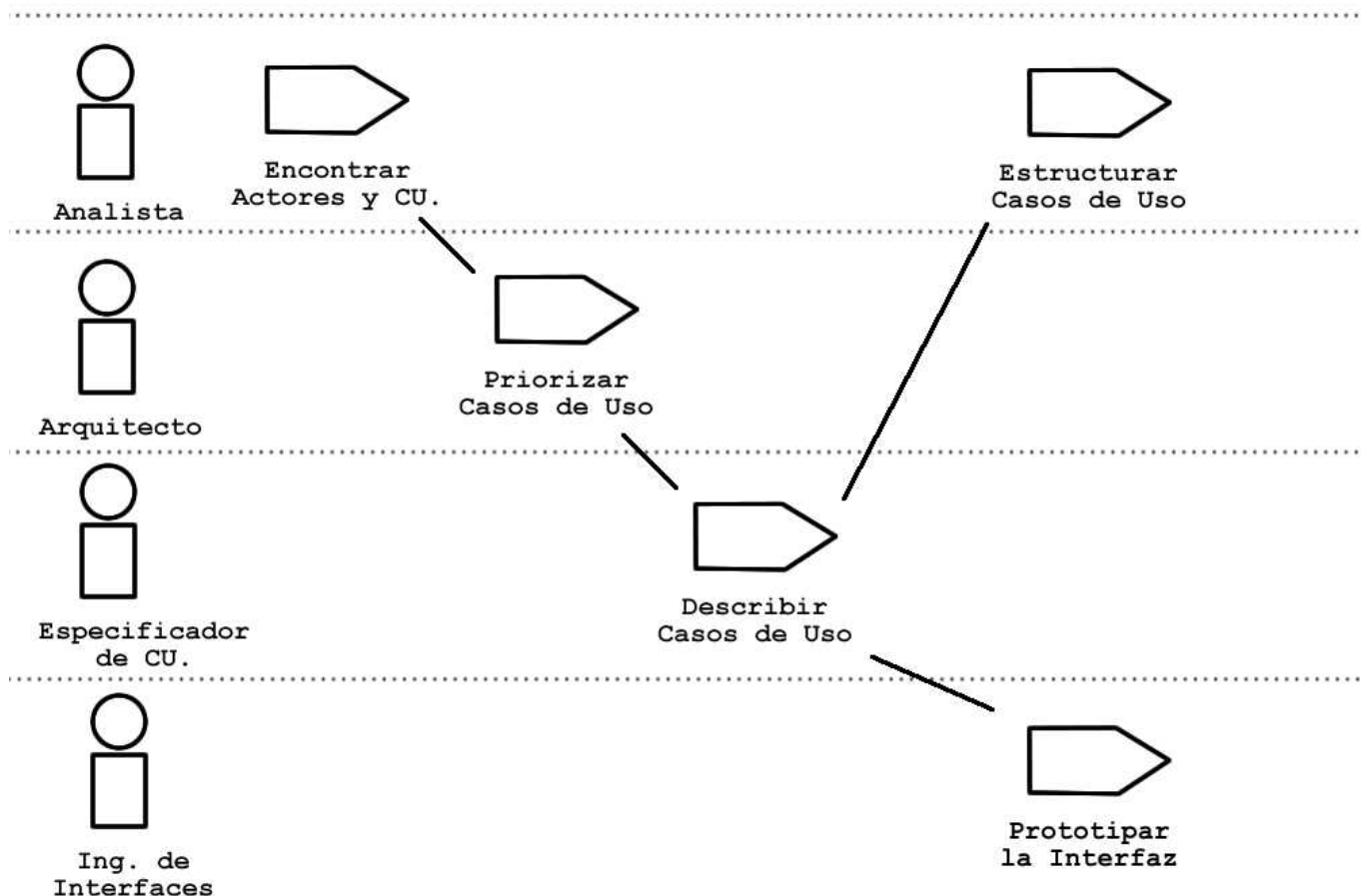
- La fase inicial o de Incepción se centra en la captura de requerimientos.
- Durante la fase de Incepción se establecen los criterios de éxito y los riesgos, se estiman los recursos necesarios y se delimita el alcance del sistema.

Los principales Artefactos involucrados en la Captura de Requerimientos son: El modelos de Casos de Uso; La descripción de los Actores; La descripción de la Arquitectura; El Glosario de términos y El Prototipo de Interfaz a Usuario.

Los principales Trabajadores involucrados en la Captura de Requerimientos son: El Analista de Sistemas; El Especificador de Casos de Uso; El Diseñador de la Interfaz de Usuario y El Arquitecto.

Las principales Actividades involucradas en la Captura de Requerimientos son: Identificar Actores y Casos de Uso; Priorizar los Casos de Uso; Detallar Casos de Uso; Prototipar la Interfaz con el Usuario y Estructurar el Modelo de Casos de Uso.

### Captura de Requerimientos



## Elaboración

- La fase de Elaboración se centra en el análisis y diseño.
- Durante la fase de Elaboración se analiza el dominio del problema, se establece una fundación arquitectónica del sistema y se tratan los elementos de más alto riesgo para el proyecto.

Los principales Artefactos involucrados en el Análisis son: Las Clases de Análisis; Las Realizaciones de Casos de Uso a nivel de Análisis; Los Paquetes de Análisis y La descripción de la Arquitectura.

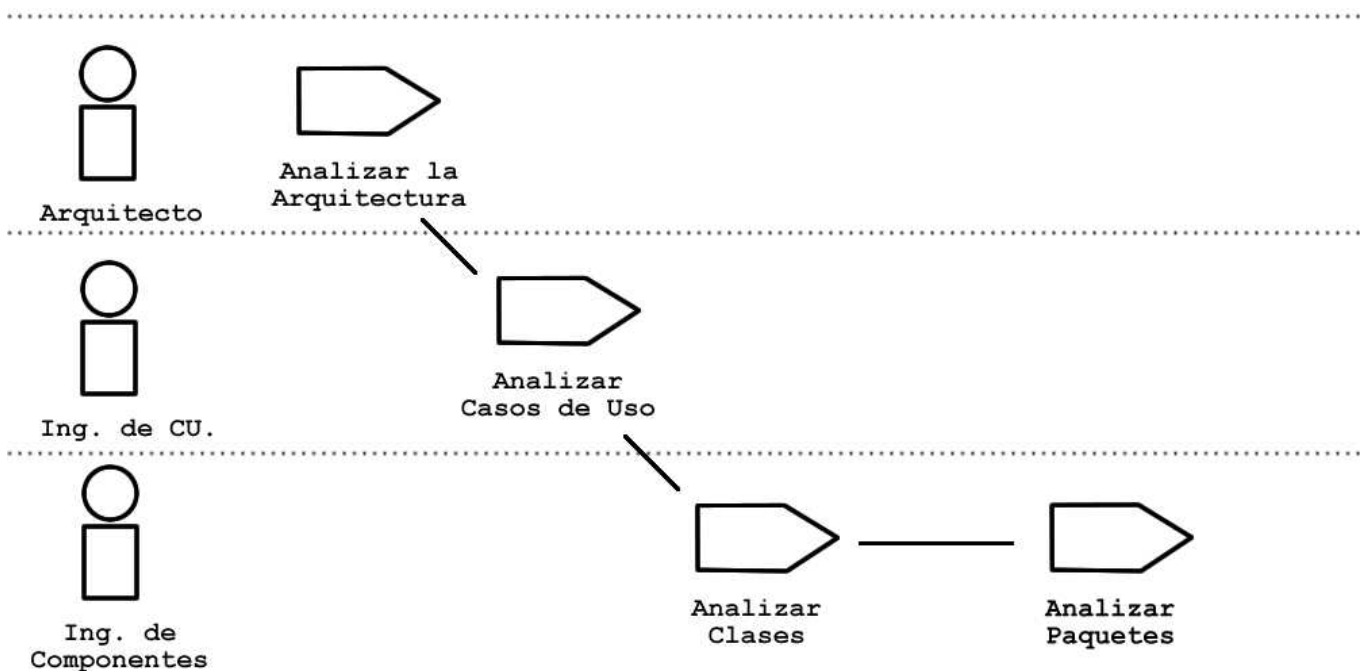
Los principales Artefactos involucrados en el Diseño son: El Modelo de Diseño; Las Clases de Diseño; Las Realizaciones de Casos de Uso a nivel de Diseño; Los Subsistemas a nivel de Diseño; Las Especificaciones de Interfaces; El modelo de Deployment y la descripción de la Arquitectura.

Los principales Trabajadores involucrados en el Análisis y Diseño son: El Arquitecto; El Ingeniero de Casos de Uso y El Ingeniero de Componentes.

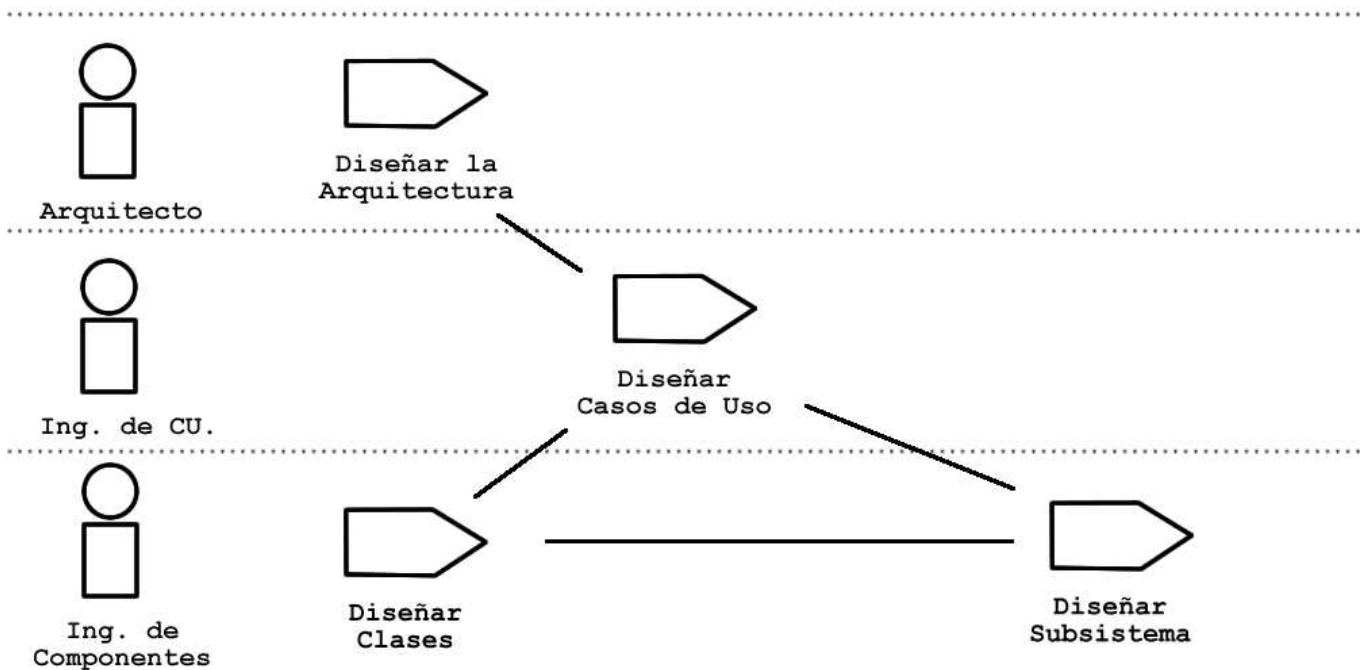
Las principales Actividades involucradas en el Análisis son: El Análisis Arquitectónico; El Análisis de Casos de Uso; El Análisis de Paquetes y El Análisis de Clases.

Las principales Actividades involucradas en el Diseño son: Diseñar la Arquitectura; Diseñar los Casos de Uso y Diseñar las Clases.

## Análisis



## Diseño



### Construcción

- La fase de Construcción se centra en la implementación y el testing.
- Durante la fase de Construcción se desarrolla iterativa e incrementalmente un producto completo para ser entregado en la fase actual a la comunidad de usuarios.

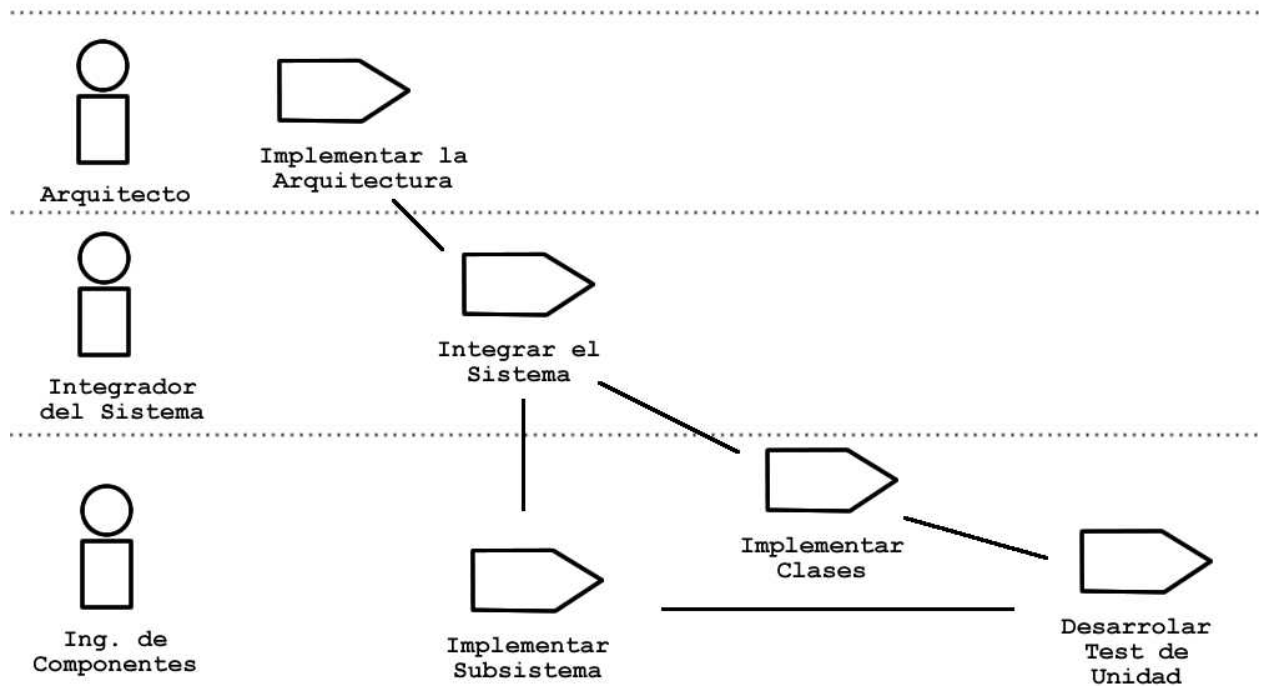
Los principales Artefactos involucrados en la Implementación son: El modelo de Implementación; Los Componentes; Los Subsistemas a nivel de Implementación; Las Especificaciones de Interfaces; La descripción de la Arquitectura y el Plan de Construcción e Integración.

Los principales Trabajadores involucrados en la Implementación son: El Arquitecto; El Ingeniero de Componentes y El Integrador a nivel de Sistema.

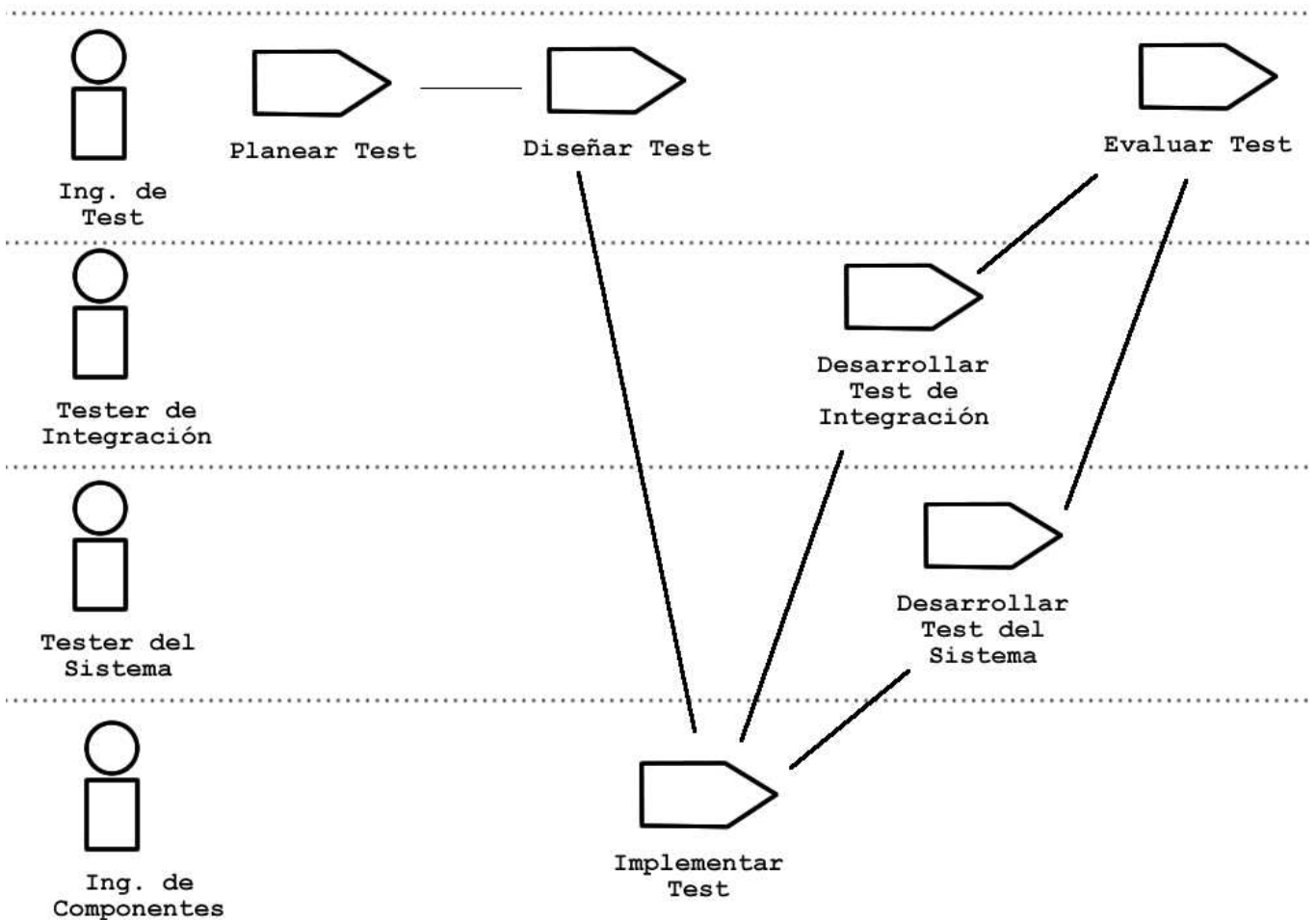
Los principales Trabajadores involucrados en el Testing son: El Ingeniero de Test; Los Ingenieros de Componentes; Los realizadores de los Test de Integración y Los realizadores de los Test a nivel de Sistema.

Las principales Actividades involucradas en la Implementación son: Implementar la Arquitectura; Integrar el Sistema; Implementar Subsistemas; Implementar Clases y Realizar Test de Unidades individuales.

## Implementación



## Testing



## Transición

- En esta etapa se traspassa el producto a los usuarios, lo que incluye manufacturar, entregar, entrenar, dar soporte y mantener el producto hasta que los usuarios estén satisfechos.

## Testing

Los *Testing's* (pruebas de software) son los procesos que permiten verificar, validar y revelar la calidad de un producto software.

- **VERIFICAR:** ¿Estamos construyendo el sistema correctamente?
- **VALIDAR:** ¿Estamos construyendo el sistema correcto?

Las pruebas de software se integran dentro de las diferentes fases del Ciclo del software dentro de la Ingeniería de software. Así se ejecuta un programa y mediante técnicas experimentales se trata de descubrir que errores tiene.

Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema.

Las pruebas de software, testing o beta testing es un proceso usado para identificar posibles fallos de implementación, calidad, o usabilidad de un programa de ordenador o video juego. Básicamente es una fase en el desarrollo de software consistente en probar las aplicaciones construidas. "El testing puede probar la presencia de errores pero no la ausencia de ellos".

## Niveles de testing

**Test de unidad:** verifica una y solo una unidad (clase, bloque).

**Test de integracion:** verifica que un conjunto de unidades trabajen juntas correctamente.

**Test del sistema:** verifica que todo el sistema funcione como corresponde.

## Tipos de testing

**Test de regresión:** verifica que lo que ya funcionaba siga funcionando normalmente con nuevas incorporaciones.

**Test de operación:** clásico, ver que el sistema funcione correctamente en situaciones normales.

**Test de escala total:** ver todos los límites del sistema. Probar el programa en situaciones límites. Ej., si el máximo permitido es de 50 usuarios, pruebo que pasa con 50 usuarios.

**Test de performance o capacidad:** mide la habilidad de procesamiento del sistema.

**Test de sobrecarga:** sobrecargar todo para tratar de que falle el sistema.

**Test negativo:** es ir más allá de los límites. Tratar de cargar 51 usuarios en el sistema.

**Test ergonómico:** me permite evaluar la interfaz de usuario. Ej., si el sistema debe estar capacitado para no videntes o para personas mayores con

miedo.

**Test de documentación:** ver si se corresponde el código con la documentación.

**Test de aceptación:** el más deseado, el cliente acepta que el sistema está bien y lo paga.

## UML y RUP: ¿Cómo trabajan en armonía?

En principio sería útil aclarar que UML y RUP son dos cosas distintas. Mientras que UML es sólo un lenguaje visual y de modelado, RUP es un proceso de desarrollo de software.

RUP se basa en el desarrollo iterativo e incremental. Y este proceso es relativamente complicado si no lleva una adecuada documentación. Para hacer más fácil el trabajo, dividen a las actividades en disciplinas y dentro de éstas proponen el uso de modelos. Cada disciplina atacará cierta actividad o tarea desde un punto de vista.

Es aquí donde entra UML. Como UML ofrece un amplio conjunto de diagramas para representar las ideas desde diferentes, y complementarios, punto de vista, RUP aprovecha esta ventaja y adoptan a UML como una herramienta más para realizar, diseñar y documentar el desarrollo de sistemas.

En síntesis, RUP propone usar UML para llevar la documentación del sistema, facilitar la etapa del diseño y posterior construcción o desarrollo, transmitir ideas y ayudar al equipo a comunicarlas.

Ahora bien, UML tiene mayor sentido cuando se está hablando de un análisis, diseño y programación bajo el paradigma OO (Orientado a Objetos), aunque uno puede, si así lo desea, extrapolar el concepto de un diagrama para transmitir una idea fuera del paradigma OO. Como por ej., el diagrama de actividad que en ocasiones se lo emplea para representar el flujo de información y los procesos de un área o departamento de una empresa.

Recordar que como proceso, RUP no impone el uso del paradigma de programación. Si bien el concepto UP nació para facilitar los proyectos que hacían uso de la orientación a objetos, nada impide seguir otro paradigma.

Los modelos o procesos de desarrollo sólo se limitan a ofrecer un marco de trabajo y una forma de estructurar las actividades.