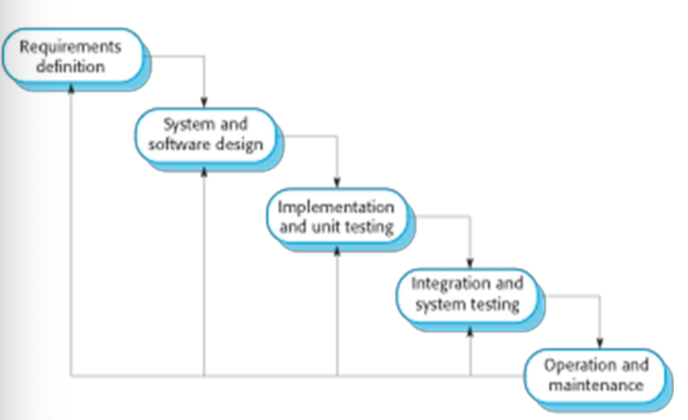
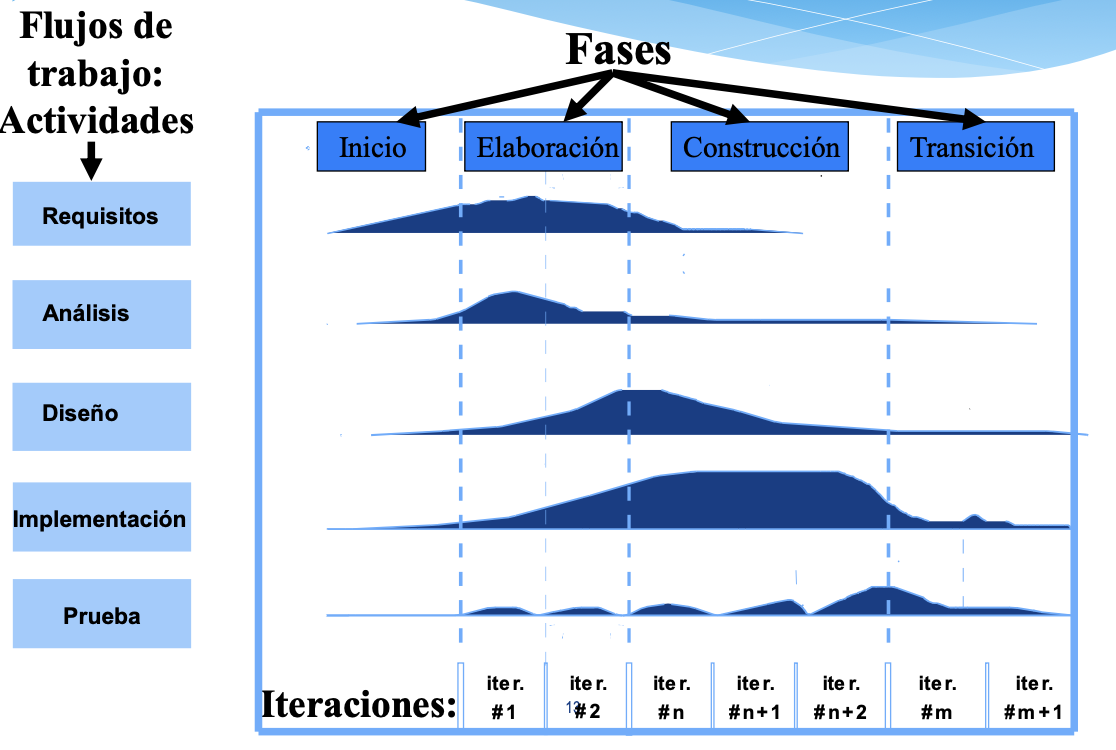
# Unidad 1

## **Fases y etapas del ciclo de vida de un software**



## Criterios a considerar para el desarrollo de sw

* Nivel de flexibilidad de los requerimientos
* Considerar la escala y el alcance del desarrollo
* Que conviene más teniendo en cuenta lo que exige el cliente
* Definir los usuarios finales claramente
* Considerar donde se ubica el equipo de desarrollo

## Modelado del negocio

Es la perspectiva de cómo se mueven los procesos en el lugar en donde se construirá una solución de software (institución, empresa o dominio).

Los diagramas de proceso del negocio, son representaciones de los procesos que se dan en la organización y que son de interés para el sistema.

Es un diagrama de flujo extendido, en donde además de las operaciones, se detallan las personas que intervienen en un proceso.

## Importancia del objetivo

**Los objetivos deben estar planteados desde la perspectiva del negocio.** Esto nos permite ubicarnos desde la perspectiva del cliente y del usuario y no solamente como desarrollador.

* Deben definir **por que** se construye el sistema para tal fin.
* Que **beneficios** obtiene el negocio en la construcción del sistema.
* No debe describir funcionalidades

## Casos de Uso

* Especifica **requerimientos funcionales** del Sistema.
* Se utiliza a nivel de Sistema o Negocio.
* **Dirigen el proceso de desarrollo.**
* El diagrama trata de identificar:
* Actores:
  + Representa un rol que interactúa con el sistema y ejecuta las funcionalidades.
  + El nombre del actor debe describir el rol que desempeña.
  + Existen Actores Principales y Actores Secundarios.
* Casos de uso:
  + Describen la funcionalidad y requerimientos del sistema.
  + Deben ser descriptos con una acción en infinitivo al principio.
  + Deben ser equilibrados entre el tamaño y su descripción.
  + Evitar CU muy minimalista o abarcativos.
  + Un requerimiento o funcionalidad se puede descomponer y luego relacionar para formar un todo.
* Relaciones:
  + **Include:** debe ejecutarse siempre y ayudan a complementar la función de quién lo llama.
  + **Extend:** agrega funcionalidad a otro caso de uso, que puede o no ejecutarse siempre, es optativo.
  + **Generalización:** permite especificar comportamiento partiendo de la funcionalidad base.

## Modelado

### ¿Qué es un Modelo?

Un modelo es una simplificación (representación a bajo costo) de la realidad. **Es una abstracción del sistema.**

### ¿Qué lenguaje emplear para modelar software?

Lenguaje visual, fácil de interpretar y procesar.

### **¿Cuál es el objetivo de modelar?**

* Nos ayudan a visualizar un sistema como es, o cómo queremos que sea.
* Nos permiten especificar la estructura o el comportamiento de un sistema.
* Nos dan una plantilla que nos guía en la construcción del sistema.
* Documentan las decisiones que hemos tomado.
* Nos facilitan la comunicación con el cliente.

### Aportes del Modelado

* Se facilita la comunicación entre el equipo al existir un lenguaje común.
* Se dispone de documentación que trasciende al proyecto.
* Hay estructuras que trascienden lo representable en un lenguaje de programación.

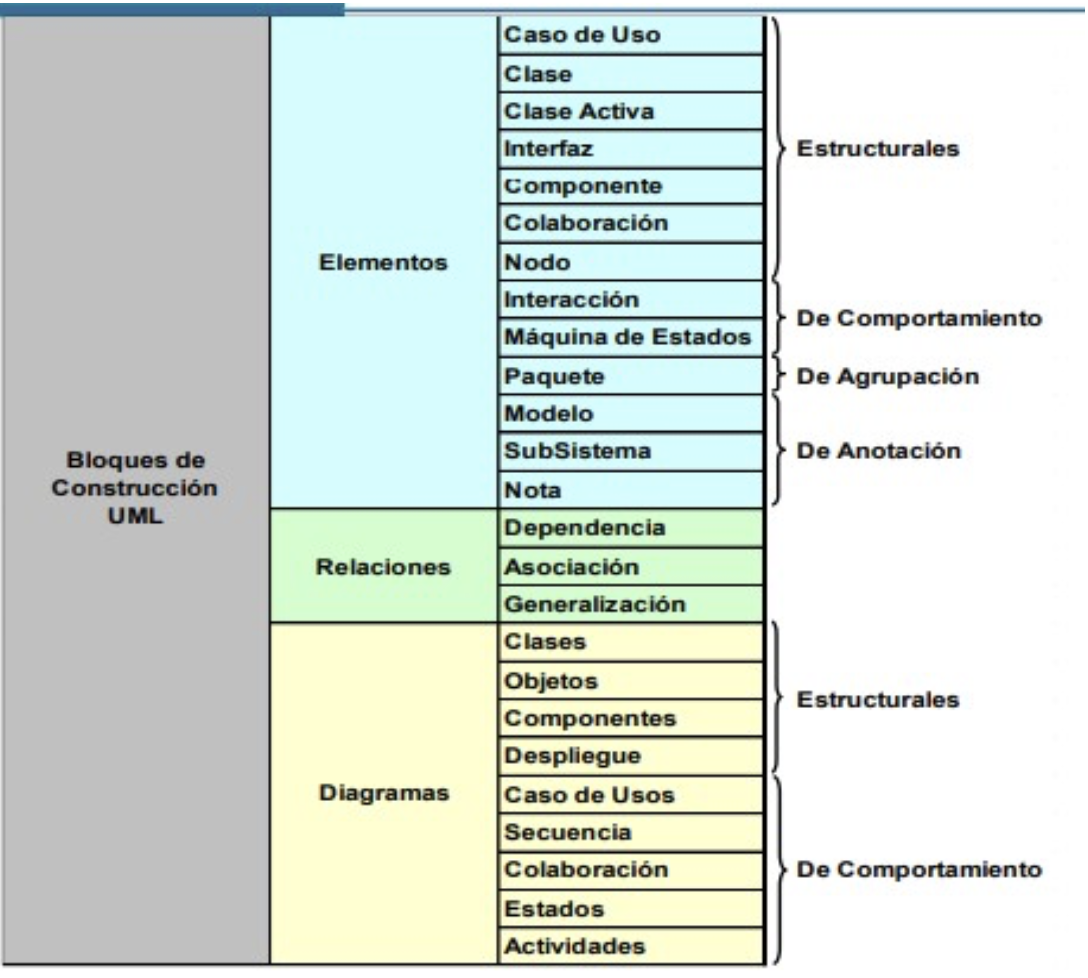
### Conceptos del Modelado

* **Sistema software:** Descrito por un conjunto de modelos
* **Modelo:** Simplificación para comprender mejor un sistema
* **Diagrama:** Representación gráfica de un modelo (UML usa grafos)
* **Vista:** Subconjunto de diagramas de un modelo que analiza un aspecto concreto

## UML

Es un lenguaje de modelado, de propósito general, usado para la visualización, especificación, construcción y documentación de sistemas orientados a objetos.

Combina las nociones provenientes del Modelado Orientado a Objetos, Modelado de Datos, Modelado de Componentes y Modelado de Flujos de Trabajo.



# **Unidad 2**

## **El proceso unificado del desarrollo de software**

Es un proceso **orientado a objetos**, guiado por **casos de uso**, centrado en la **arquitectura** y con un ciclo de vida **iterativo** **e** **incremental**.

* Guiado por **CASOS DE USO**
* Centrado en la **ARQUITECTURA**
* Ciclo de vida **ITERATIVO E INCREMENTAL**

Debe ofrecer un marco de trabajo genérico que unifique la **parte dinámica**, la **parte estática**, el **ciclo de vida** y la **interfaz**.

Utiliza **UML** en su notación, herramientas como "**visio**" o "**rational** **rose**" y sigue el "**proceso** **unificado** **de** **desarrollo** **de** **Rational**"

El CV del Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo constituye una nueva versión del sistema.

Un ciclo de vida se divide en fases, cada fase se divide en iteraciones, en cada iteración se realizan flujos de trabajo.

Cada ciclo consta de cuatro **fases**:

* **Inicio:** Define el alcance del Proyecto, describe producto final y análisis del negocio, identifica los riesgos más importantes, establece la planificación inicial del Proyecto y decide si se continúa.
* **Elaboración:** Planifica el Proyecto, elabora una arquitectura base.
* **Construcción:** Construir el Sistema y desarrollar el producto.
* **Transición:** Transición a los usuarios, es decir, proporciona sistema a los usuarios.

**Producto del Proceso Unificado**

* No es sólo código ejecutable.
* Son los modelos o representación del software.
* Debe ajustarse a todas las personas implicadas.

**Iteraciones**

Cada fase se divide en iteraciones donde cada iteración es un mini proyecto (en cascada) que ejecuta flujos de trabajo que producen un incremento en producto tal y como estaba.

Se reduce el riesgo ya que se puede perder sólo lo realizado en esa Iteración.

**Flujo de Trabajo**

* **CAPTURA DE REQUISITOS:** identificar requisitos del sistema, construir un modelo del mismo, modelo de casos de uso, modelo del dominio (o negocio).
* **ANÁLISIS:** especificar requisitos, construir modelo del análisis.
* **DISEÑO:** encontrar la forma del sistema (solución), construir modelo del diseño.
* **IMPLEMENTACIÓN:** codificar el diseño (solución), construir modelo de implementación.
* **PRUEBAS:** verificar la implementación, construir modelo de pruebas.

En la ***Fase de Iniciación***, se hace Un MCU simplificado con los CU más críticos, Borrador con los subsistemas principales y se Identifican los riesgos principales y priorizarlos, planifica la elaboración y presupuesto aproximado.

En la ***Fase de Elaboración***, se especifican en detalle los CU más críticos, se diseña la arquitectura.

En la ***Fase de Construcción***, La arquitectura se completa para construir un sistema bien cimentado, La visión evoluciona hasta convertirse en un producto preparado para los usuarios, Es donde se gastan la mayoría de los recursos, La arquitectura del sistema es estable. Sin embargo, se pueden realizar cambios mínimos a la misma.

En la ***Fase de Transición***, el producto se encuentra en fase beta. En esta fase se encuentran actividades como la venta, formación de los usuarios, ofrecimiento de ayuda en línea y corrección de defectos descubiertos tras la implantación.

# **Unidad 3**

## **¿Qué es un Requerimiento?**

Un requerimiento de software define las **funciones**, **capacidades** o **atributos** de cualquier sistema de software.

Un requerimiento de software define tanto los **servicios** que un sistema de software debe ofrecer como sus **limitaciones**.

**Representan:**

* Factores de calidad del sistema.
* Lo que se requiere implementar.
* Una descripción de cómo el sistema deberá comportarse (dominio, restricciones, operaciones, atributos y propiedades)
* Un problema por resolver.

## Características de los requerimientos

* Estar por escrito
* Descritos como una característica del sistema a entregar.
* Necesarios
* No ambiguos
* Completos
* Correctos
* Consistentes
* Verificables
* Viables
* Priorizables
* Precisos

## Tipos de requerimientos

* **Funcionales:** describen servicios o funciones.
* **No-funcionales:** son un límite en el sistema o en el proceso de desarrollo. Pueden ser de producto, organizacionales o externos.
* **De Dominio:** se obtienen de el dominio de la aplicación del sistema y reflejan sus características.

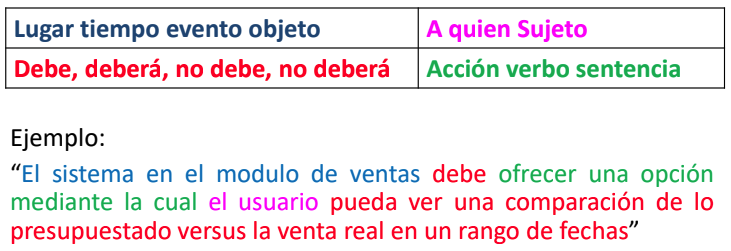
## Identificación de requerimientos

Tareas que facilitan la identificación de requerimientos:

* Analice los documentos “fuentes” (resultados de las reuniones con los usuarios).
* Extraiga las sentencias que contienen las palabras: DEBERA, DEBE.
* Identifique sentencias que impliquen requerimientos.
* Identifique sentencias de tipo VERBOS, ACCIONES y un RESULTADO.
* Haga una lista de FUNCIONALES y NO FUNCIONALES.

## Redacción de un requerimiento

***Todos los requerimientos deben tener el mismo estilo de redacción.***



## Problemas con los Requerimientos

* El proceso de requerimientos no es fácil.
* No es simplemente “tomar nota” de las necesidades del cliente.
* Es un proceso de comunicación.
* Es un proceso de negociación.
* No hay acuerdos en los requerimientos.
* Los requerimientos no se han priorizado.
* Requerimientos incompletos.
* Requerimientos contradictorios.
* Requerimientos ambiguos.
* Requerimientos de desarrollador. (Presunciones de diseño)

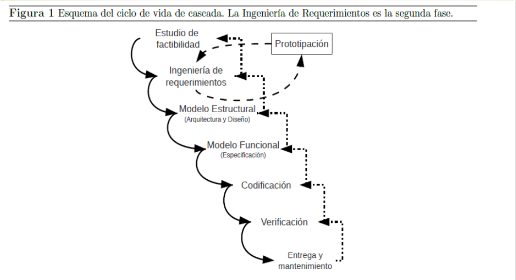
***¿Cómo solucionar estos problemas?***

*Llevar un proceso disciplinado que busque determinar y solucionar los diferentes problemas de los requerimientos, y un sistema de especificaciones que posibilite comunicar y negociar los requerimientos eficientemente con los usuarios.*

# **Unidad 4**

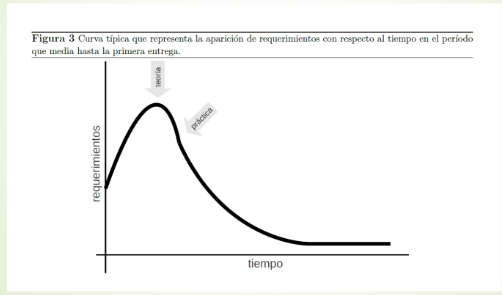
## Ingeniería de Requerimientos

Es el proceso para establecer los servicios que el sistema deberá proveer y las restricciones bajo las cuales deberá operar y ser desarrollado.

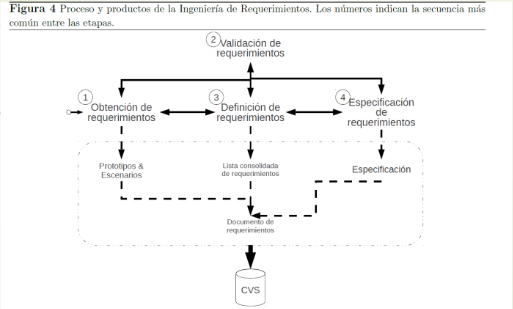


## Relación entre los requerimientos y el resto de las etapas del proceso

* El diseño y la especificación del sistema se obtienen a partir de los requerimientos.
* El diseño y la especificación se utilizan para implementar el programa.
* Finalmente, los requerimientos son el documento esencial para llevar adelante el testing de aceptación.
* Cuanto mayor calidad tengan los requerimientos mayores son las posibilidades de que el proyecto sea un éxito.
* Se podrá iniciar la fase de Arquitectura y Diseño (o la etapa de Definición De requerimientos) cuando se desacelera la cantidad de requerimientos validados que pide el cliente.
* Esto ocurre cuando cuando el cociente entre la aparición de requerimientos y el tiempo es bajo durante un periodo de tiempo razonable.



## **Proceso de la Ingeniería de requerimientos**



## **Obtención de requerimientos**

En esta etapa se obtienen los requerimientos del sistema a través de la observación de sistemas existentes y del entorno donde se instalará el sistema.

* Reuniones con los interesados.
* Generación de prototipos.
* Definición de escenarios.

## Validación de requerimientos

Validación de requerimientos. Los requerimientos solo pueden ser validados por el cliente. Por lo tanto, una vez que se ha obtenido un requerimiento (preguntando al cliente) se le debe preguntar si ese es el requerimiento que expresó.

## Proceso de Validación de Requerimiento

**Verificaciones de validez:** Aquí surgen funciones adicionales o diferentes a las planteadas.

**Verificaciones de consistencia:** En el documento de requerimiento no debe haber restricciones o descripciones contradictorias de la misma función del sistema.

**Verificaciones de completitud:** El documento de requerimiento debe incluir requerimientos que definan todas las funciones y restricciones propuestas por el usuario.

**Verificaciones de realismo:** De acuerdo a la tecnología existente, los requerimientos deben verificarse para asegurar que se pueden implementar, también se debe tener en cuenta el presupuesto y la agenda.

**Verificabilidad:** Esto se logra si por cada requerimiento se puede escribir un conjunto de pruebas que demuestren que el sistema a entregar cumple con los requerimientos especificados.

## Técnicas de Validación de Requerimiento

**Revisión de Requerimientos:** Los requerimientos son analizados sistemáticamente por un conjunto de revisores.

**Construcción de prototipos:** Los usuarios pueden experimentar con este modelo para ver si cumple con las necesidades reales. (Elegir las personas que probaran el protoDpo ́ Desarrollar los escenarios a probar ́ Ejecutar los escenarios ́ Documentar los problemas)

**Generación de casos de prueba:** Los requerimientos deben poder probarse. Se diseñan las pruebas antes de la codificación de los requerimientos. (Planificar las pruebas ́ Diseñar los procedimientos de prueba. ́ Diseñar los casos de prueba. ́ Ejecutar las pruebas ́ Documentar resultados)

## Definición de requerimientos

En esta etapa la información recolectada en la etapa anterior se vuelca en un documento consolidado, organizado y estructurado.

**Priorización de requerimiento:**

* Requerimientos que deben ser absolutamente satisfechos.
* Requerimientos que son muy deseables pero no indispensables.
* Requerimientos que son posibles, pero que podrán eliminarse.

## Calidad de los requerimientos

* La calidad de un requerimiento se refiere en general a que tan perdurable en el tiempo será tal y como está descrito en este momento.
* Un requerimiento es de mala calidad si el cliente no puede expresarlo claramente o directamente duda o desconoce que es lo que realmente necesita o quiere.
* La calidad de un requerimiento está directamente relacionada con el grado de validación que aún necesita.

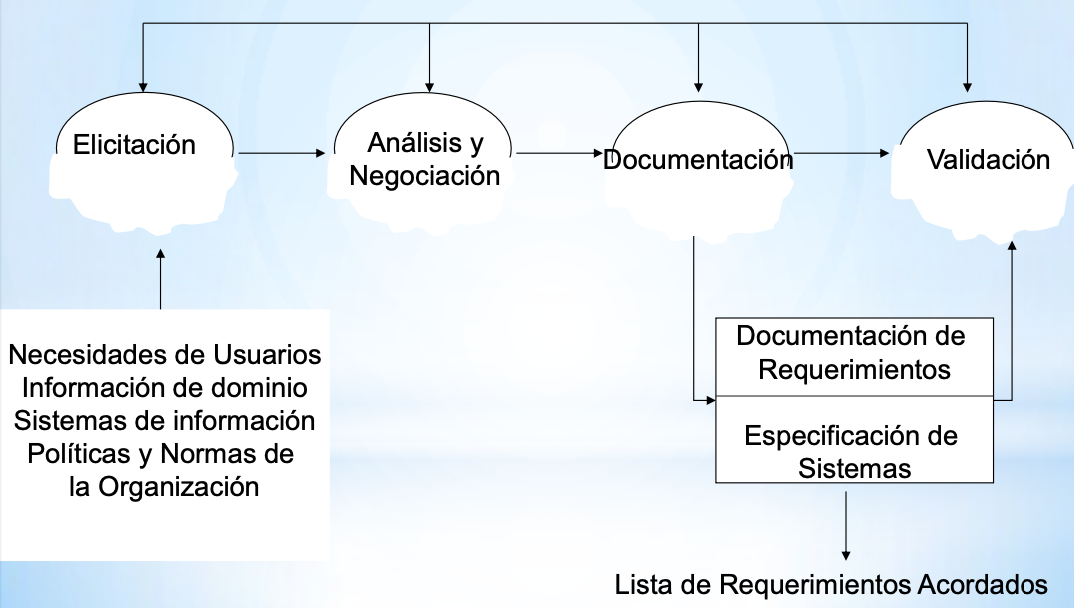
## Especificación de requerimientos

En pocas palabras en esta etapa la Lista consolidada de requerimientos se escribe desde la perspectiva del desarrollador. Esta etapa puede incluirse en la fases denominadas Modelo Estructural o Modelo Funcional en el modelo de ciclo de vida de cascada

## **Productos del proceso de la ingeniería de requerimientos**

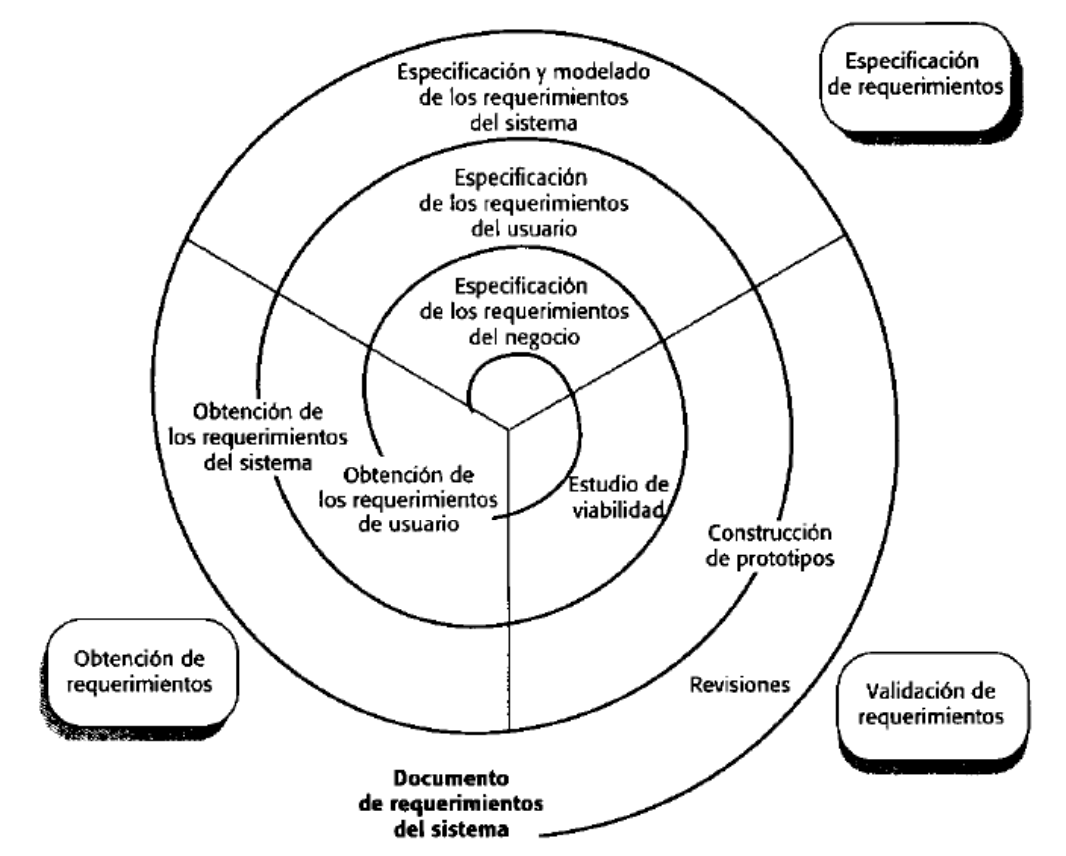
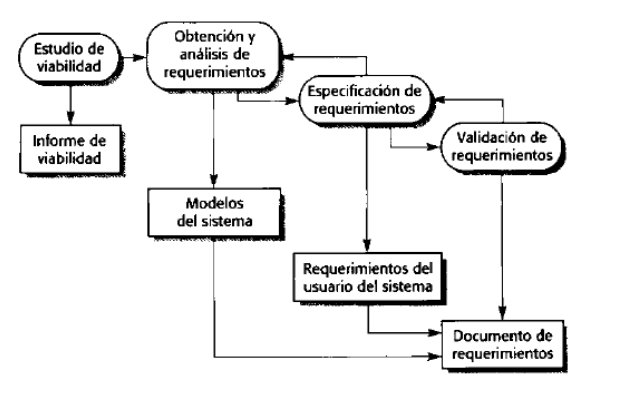
* **Prototipos:** Debe ser la forma de interacción casi excluyente entre los ingenieros de requerimientos y el cliente durante esta fase. Nos referimos a prototipo desechables. Un prototipo desechable es un programa que imita al menos algunos de los requerimientos funcionales pero luego de que son validados, el programa se descarta.
* **Escenarios:** Se utilizan escenarios de atributos de calidad como técnica efectiva para determinar con suficiente precisión cuál es el significado que el usuario le asigna a las cualidades del sistema. Ya que es muy complejo o costoso y muchas veces imposible prototipar atributos de calidad de un sistema.
* **Lista consolidada de requerimientos:** Es un documento que consolida, organiza y estructura la información recolectada durante la etapa de obtención de requerimientos. El vocabulario y las notaciones que se utilicen para escribir esta lista deben ser estrictamente las del cliente o las del dominio de aplicación No deberá incluir términos técnicos ni frases que hablen de conceptos propios de la carrera.

## **Modelos del proceso de IR**

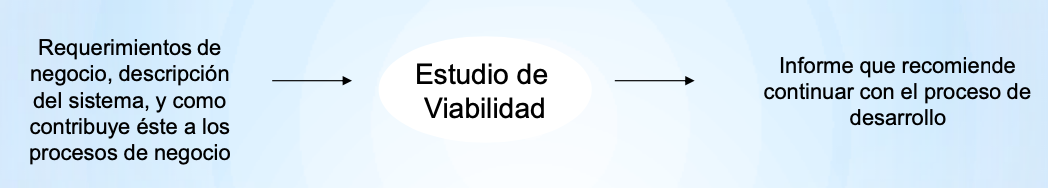


* **Elicitación de Requerimientos:** Significa descubrir requerimientos del sistema utilizando entrevistas con los stakeholders, documentación del sistema, conocimiento del dominio y estudios de mercado.
* **Análisis y Negociación:** Los requerimientos son analizados en detalle y los stakeholders se ponen de acuerdo para determinar los requerimientos válidos.
* **Documentación de Requerimientos:** Se documentan los requerimientos acordados utilizando lenguaje natural y diagramas que faciliten la comprensión de todos los stakeholders.
* **Validación de Requerimientos:** Se realiza una cuidadosa revisión de los requerimientos evaluando consistencia y completitud. Este proceso tiene la intención de detectar errores antes de que los requerimientos se tomen como base para el desarrollo

## El proceso de IR/ Modelo en espiral



## **Estudio de Viabilidad**



## Requerimientos según la IEEE

Según la IEEE Std. 610, los requerimientos son una condición o capacidad que necesita el usuario para resolver un problema o alcanzar un objetivo. Condición o capacidad que debe satisfacer o poseer un sistema o un componente de un sistema para satisfacer un contrato, un estándar, una especificación u otro documento formalmente impuesto.

Los **requerimientos funcionales** de un sistema describen lo que el sistema debe hacer. Describen en detalle la función de éste, sus entradas y salidas, excepciones, etc.

Los **requerimientos no funcionales** no se refiere a las funciones del sistema sino a las propiedades emergentes de éste tales como la fiabilidad, tiempo de respuesta, capacidad de almacenamiento. Estos se dividen en 3 tipos, los cuales son “Del Producto”, “Organizacionales” y ”Externos”.

Los **requerimientos del usuario** deben describir los requerimientos funcionales y no funcionales de tal forma que sean comprensibles por los usuarios del sistema sin conocimientos técnicos detallados. Especifican el comportamiento externo del sistema y deben evitar las características de diseño del sistema.

Los **requerimientos del sistema** son versiones extendidas de los requerimientos del usuario y son utilizados por los ingenieros de software como punto de partida para el diseño de sistema; deben describir el comportamiento externo del sistema y sus restricciones operativas.

## **Administración de requerimientos**

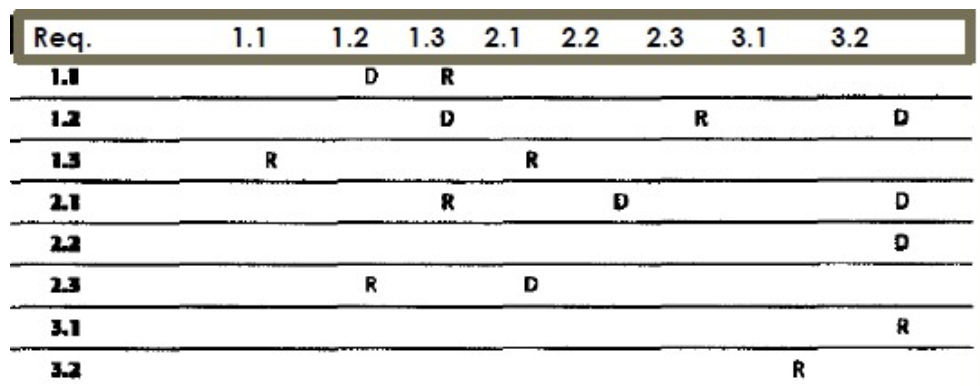
Es el proceso de comprender y controlar los cambios en los requerimientos. Hay que establecer un proceso formal para implementar las propuestas de cambios y vincular estos a los requerimientos del sistema. Debería empezar en cuanto este disponible una versión preliminar del documento de requerimientos.

**¿Porque administramos requerimientos?**

* Los requerimientos para sistemas de sw son siempre cambiantes
* Los requerimientos de sw son incompletos porque el problema no puede definirse completamente
* La comprensión del problema por parte de los stakeholders está cambiando constantemente a lo largo del proceso de desarrollo.

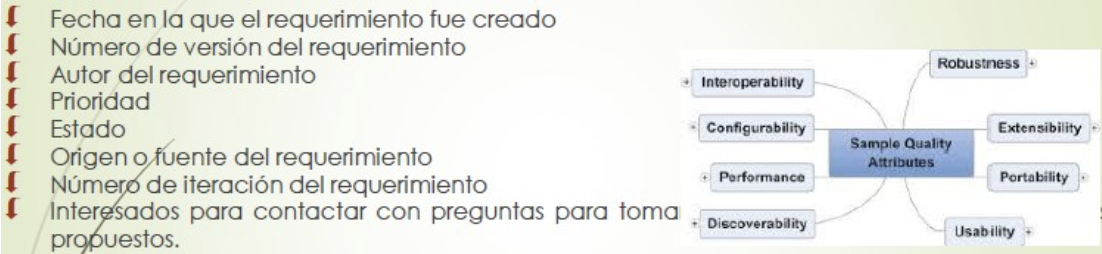
## **Matriz de rastreo (Trazabilidad)**

Relaciona los requerimientos, los stakeholders y módulos de diseño. En esta matriz cada requerimiento se representa en una fila y en una columna de la matriz. Cuando existen dependencias entre diferentes requerimientos, se registran en la celda de intersección fila/columna.



* D = depende del requerimiento (fila depende de columna)
* R = existe alguna otra relacion mas débil entre los requerimientos.

## Atributos de los Requerimientos



# Unidad 6

## **Diseño**

El diseño de software agrupa el **conjunto de principios, conceptos y prácticas** que llevan al desarrollo de un sistema o **producto de alta calidad**. El diseño es una representación significativa de ingeniería de algo que vamos a construir.

## Características de un buen diseño

* **Resistencia:** un programa no debe tener ningún error que impida su funcionamiento.
* **Funcionalidad:** un programa debe ser apropiado para los fines que persigue.
* **Belleza:** la experiencia de usar el programa debe ser placentera.

## Etapas o niveles de detalle del diseño

Comienza con el modelo de requerimientos. Se trabaja para transformar este modelo y obtener cuatro niveles de detalle de diseño:

* La estructura de **datos**
* La **arquitectura** del sistema
* La representación de la **interfaz**
* Detalles a nivel de **componentes**

## **¿Cómo influye el diseño (arquitectura) en el sistema?**

La arquitectura del sistema afecta al rendimiento, solidez, grado de distribución y mantenibilidad de un sistema (**Requerimientos No Funcionales**).

* Rendimiento → Subsistemas
* Protección → Capas
* Seguridad → Único Subsistema
* Disponibilidad → Componentes Redundantes
* Mantenibilidad → Componentes Independientes

## Elementos del Modelo de Análisis

El modelo de análisis debe lograr tres objetivos:

* Describir lo que quiere el cliente
* Establecer una base para el diseño de software
* Definir un conjunto de requerimientos que pueda validar una vez que se construye el software.

Cada parte del modelo de análisis proporciona la información necesaria para crear los cuatro modelos de diseño. El modelo de análisis extraído de un diseño estructurado es el siguiente:

* **DISEÑO DE DATOS** → estructuras de datos
* **DISEÑO ARQUITECTÓNICO** → Define relación entre los elementos estructurales,patrones de diseño, los requisitos y las restricciones
* **DISEÑO DE INTERFAZ** → la manera de comunicarse el software dentro de sí mismo
* **DISEÑO DE NIVEL DE COMPONENTES** → descripción procedimental de los componentes

## El Proceso de Diseño

El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen en un «plano». Se representa desde un nivel más global (abstracto) y va hacia lo más particular (comportamiento, funcionalidades y de datos).

## Diseño y Calidad

La calidad de la evolución del diseño se evalúa con una serie de revisiones técnicas formales:

* El diseño deberá implementar todos los requerimientos.
* El diseño proporcionará una imagen completa del software.
* El diseño es una guía para los que generan código.

Los **FACTORES DE CALIDAD EXTERNOS** son esas propiedades del software que pueden ser observadas fácilmente por los usuarios. Estos son:

* Funcionalidad
* Confiabilidad
* Usabilidad
* Eficiencia
* Mantenibilidad
* Portabilidad

Los **FACTORES DE CALIDAD INTERNOS** tienen importancia para los ingenieros de software. Desde una perspectiva técnica conducen a un diseño de calidad alta. Estos son:

* Modularidad
* Cohesión y Acoplamiento
* Comprensibilidad
* Testabilidad
* Reusabilidad
* Documentación
* Estabilidad

# Unidad 7

## **¿Qué es la arquitectura del software?**

La arquitectura de software es la **estructura integral de un sistema**, que se manifiesta en sus **componentes** principales, sus **interacciones** y el **entorno** en el que operan. Incluye los principios y directrices que guían su diseño, desarrollo y evolución a lo largo del tiempo.

## Vistas Arquitectónicas

Una vista es una **representación de** un conjunto de **elementos** del sistema y las **relaciones** asociadas con ellos.

Documentar una arquitectura significa **documentar las vistas relevantes bajo las cuales se puede observar un sistema**.

## Vistas según el modelo 4+1 de Kruchten

### 

## Documentos para una vista

La documentación para una vista incluye una **representación gráfica** de los elementos principales y sus relaciones, **un catálogo de elementos** que define sus propiedades, **una especificación de las interfaces** y **el comportamiento de cada elemento**.

## Documentación de un Arquitectura

En la documentación de una arquitectura de software, es importante considerar lo siguiente:

* **Representación del sistema en el contexto:** Se debe representar el sistema objetivo en relación con su entorno, identificando sistemas superiores, sistemas subordinados, sistemas entre iguales y actores que interactúan con el sistema.
* **Identificación de arquetipos de la arquitectura:** Los arquetipos son clases o patrones fundamentales para el diseño de la arquitectura. Representan abstracciones críticas y pueden ser componentes genéricos o módulos especializados con funciones específicas.
* **Refinamiento hacia los componentes:** Los arquetipos se refinan en componentes más específicos, definiendo interfaces e interacciones entre

**Errores más comunes al documentar:**

* Demasiado elaborado
* Incógnitas
* Se hace difícil de mantener

# Unidad 8

## Patrones

Un patrón es un par problema/solución con un nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos.

## Patrones GRASP

**GRASP** es una acrónimo de **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns.

**GRASP Básicos:** *Experto en información*, *Creador*, *Alta* *Cohesión*, *Bajo* *Acoplamiento* y *Controlador*

**GRASP Avanzados:** *Polimorfismo*, *Fabricación* *Pura*, *Indirección* y *Variaciones* *Protegidas*

## GRASP Básicos

### Experto en información

* **Problema:** ¿Quién asigna responsabilidades a los objetos?
* **Solución:** Asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para realizar la responsabilidad). Siguiendo la solución del patrón deberíamos buscar clases y objetos que tienen la información necesaria para determinar el total. Este patrón es un principio de guía básico en el diseño de objetos. Expresa la “intuición” de lo que creemos que los objetos del mundo real hacen y por eso es muy fácil de aplicar.
* **Contraindicaciones:** En algunas circunstancias la solución de “experto” no es deseable. Por ejemplo para almacenar en una BD la información.
* **Beneficios:** Se mantiene el encapsulamiento de la información (los objetos utilizan su propia información para trabajar) Se distribuye el comportamiento entre clases obteniendo clases más pequeñas, fáciles de entender y manejar. Se logra una alta cohesión.

### Creador

* **Problema:** ¿Quién debería ser el responsable de la creación de una nueva instancia de alguna clase?
* **Solución:** Asignar a la clase B la responsabilidad de crear una instancia de A si se cumple uno o más de los siguientes casos: B **agrega** objetos de A, B **contiene** objetos de A, B **registra** instancias de objetos de A, B es un creador de los objetos A, etc. La intención del patrón Creador es encontrar el creador que necesita a futuro conectarse al objeto creado en alguna situación. El Creador sugiere que las clase contenedoras son buenas para tener la responsabilidad de la creación de sus clases contenidas.
* **Contraindicaciones:** Cuando la creación requiere alta complejidad (reciclo para mayor rendimiento) es complicado la creación de instancias.
* **Beneficios:** Se soporta el bajo acoplamiento.

### Bajo Acoplamiento

* **Problema**: ¿Cómo soportar bajas dependencias, bajo impacto en el cambio e incremento de la reutilización?
* **Solución**: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo. El acoplamiento es una medida de la fuerza con que un elemento está conectado a otros elementos. Un elemento con bajo acoplamiento no depende de demasiados otros elementos. La no existencia de acoplamiento (clase independientes) no es deseable ya que atenta contra el objetivo de los diseños orientados a objetos donde el sistema es un conjunto interconectado de objetos trabajando entre sí.
* **Contraindicaciones**: La aplicación de bajo acoplamiento para “Futuras Necesidades” donde no hay motivos realistas para hacerlo puede ser contraproducente.
* **Beneficios**: No afectan los cambios en otros componentes Fácil de Entender de manera aislada Conveniente para reutilizar .

### Alta Cohesión

* **Problema**: ¿Cómo mantener la complejidad manejable?
* **Solución**: Asignar una responsabilidad de manera que la cohesión permanezca alta. La cohesión es una medida de fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un objeto. Una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo. Las clases con baja cohesión representan un grano grande de abstracción (deberían haberse delegado en otros objetos) Cohesión y acoplamiento el yin y el yang à Una mala cohesión causa normalmente un mal acoplamiento y viceversa debido a que uno influye en el otro.
* **Beneficios**: Se incrementa la claridad y comprensión del diseño. Se simplifica el mantenimiento y mejoras. Se soporta a menudo bajo acoplamiento. El grado fino de funcionalidad incrementa la reutilización.

### Controlador

* **Problema:** ¿Quién debe manejar eventos del sistema?
* **Solución**: Asignar la responsabilidad de recibir o manejar eventos del sistema a una clase. Un controlador es un objeto responsable del manejo de los eventos del sistema. Se pueden asignar a una o más clases controlador. Normalmente los controladores deben delegar las tareas a otros objetos, su misión solo debería ser coordinar o controlar las actividades. Esto para no sobrecargar la clase y perder cohesión. Es decir un controlador saturado.
* **Beneficios**: Aumenta el potencial para reutilizar y permite el intercambio de interfaces fácilmente.

## GRASP Avanzados

### Polimorfismo

En la programación orientada a objetos el polimorfismo es permitir que varias clases se comporten de manera distinta dependiendo del tipo que sean. Siempre que se tenga que llevar a cabo una responsabilidad que dependa de un tipo, se tiene que hacer uso del polimorfismo.

* **Problema:** ¿Cómo manejar alternativas basadas en el tipo? ¿Cómo crear componentes de software conectables?
* **Solución**: Cuando las alternativas o comportamientos relacionados varían con el tipo de clase, asigne la responsabilidad del comportamiento (utilizando operaciones polimórficas) a los tipos para los que varía el comportamiento. Es un principio fundamental para gestionar variaciones similares.
* **Contraindicaciones**: Se corre el riesgo de diseñar sistemas con interfaces para “futuras necesidades” frente a variaciones desconocidas.
* **Beneficios**: Se añaden fácilmente extensiones necesarias de nuevas variaciones y Las nuevas implementaciones no afectan a los clientes.

### Fabricación Pura

* **Problema**: ¿Qué hacer cuando se quiere mantener los objetivos de Alta Cohesión y Bajo Acoplamiento pero las soluciones que ofrece Experto no son adecuadas?
* **Solución**: Asigne un conjunto de responsabilidades altamente cohesivo a una clase artificial o de conveniencia, que no representa un concepto del dominio del problema. Muchas veces la elección de Experto trae aparejado problemas de poca reutilización, alto acoplamiento y baja cohesión como por ejemplo el caso del almacenamiento de datos en una base de datos.
* **Beneficios**: Soporta Alta cohesión y el potencial de reutilización aumenta debido a la presencia de clases Fabricación Pura que tienen aplicación en otros dominios.
* **Contraindicaciones**: La inexperiencia puede llevar a abusar de la creación de clases Fabricación Pura convirtiendo en muchos casos funciones en objetos.

### Indirección

* **Problema**: ¿Dónde asignar una responsabilidad para evitar el acoplamiento directo entre dos o más cosas? ¿Cómo desacoplar objetos de manera que se soporte el bajo acoplamiento y el potencial para reutilizar permanezca alto?
* **Solución**: Asigne la responsabilidad a un objeto intermedio que funcione como mediador entre otros componentes o servicios de manera que no se acoplen directamente.
* **Beneficios**: Disminuir acoplamiento entre componentes.

### Variaciones Protegidas

* **Problema**: ¿Cómo diseñar objetos, subsistemas, y sistemas de manera que las variaciones o inestabilidades en estos elementos no tengan un impacto no deseable en otros elementos?
* **Solución**: Asignar responsabilidades para crear una interfaz estable alrededor de puntos de variaciones previstas o de inestabilidad.
* **Beneficios**: Se añaden fácilmente las extensiones que se necesitan para nuevas variaciones, se pueden introducir nuevas implementación es sin afectar a los clientes. Se reduce el acoplamiento.

# Arquitecturas

## Estilo Flujo de Datos

### Tuberías y filtros

El estilo de tuberías y filtros es uno de los ejemplos más claros de las arquitecturas de flujo de datos. En esta arquitectura, los componentes (**filtros) están conectados por conductos (tuberías**), permitiendo que los datos fluyan de un filtro a otro. **Cada filtro realiza una transformación** en los datos que recibe y luego los pasa al siguiente filtro en la secuencia. Este estilo es muy adecuado para tareas que requieren procesamiento secuencial de datos, como los compiladores o sistemas operativos basados en UNIX​​.

## Estilo Centrado en Datos

### Arquitecturas de pizarra o repositorio

En una arquitectura de pizarra o repositorio, existen dos componentes principales: una **estructura de datos central** que representa el estado actual y una colección de **componentes independientes que operan sobre esa estructura**. Dependiendo de cómo se manejan las transacciones y los datos, el repositorio puede ser una base de datos tradicional o una pizarra pura, donde el estado de los datos dispara los procesos a ejecutar​​.

## Estilo Llamada y Retorno

### Model View Controller (MVC)

El patrón MVC divide una aplicación en **tres componentes principales**: **Modelo** (gestiona los datos), **Vista** (presenta los datos al usuario) y **Controlador** (maneja la entrada del usuario y actualiza el modelo y la vista). Este estilo permite una separación clara de las preocupaciones, facilitando el mantenimiento y la escalabilidad de las aplicaciones.

### Arquitectura Orientada a Objetos

En este estilo, el sistema está compuesto por **objetos que interactúan entre sí**. Cada objeto **encapsula datos y comportamiento**, y la comunicación entre objetos se realiza mediante **mensajes**. Este enfoque promueve la reutilización de código y la modularidad.

### Arquitectura en Capas

La arquitectura en capas organiza el sistema en **niveles jerárquicos**, donde cada capa **proporciona servicios a la capa superior y utiliza los servicios de la capa inferior**. Esto facilita el mantenimiento y la escalabilidad, ya que cada capa puede desarrollarse y modificarse de manera independiente.

### Arquitectura Basada en Componentes

Similar a la arquitectura orientada a objetos, pero a una escala mayor, donde los **componentes son unidades de implementación independientes que interactúan a través de interfaces bien definidas**. Esto permite una alta reutilización de componentes y una fácil integración de nuevas funcionalidades.

## Estilo de Código Móvil

### Arquitectura de Máquinas Virtuales

En este estilo, se utiliza una **máquina virtual para ejecutar código que puede moverse entre diferentes entornos de ejecución**. Esto proporciona **portabilidad y aislamiento**, permitiendo que el mismo código se ejecute en diferentes plataformas sin modificación.

## Estilos Heterogéneos

### Sistema de Control de Procesos

Este estilo se emplea en sistemas donde el control de procesos es crucial, como en la automatización industrial. Combina **diferentes enfoques arquitectónicos para manejar tanto el control en tiempo real como el procesamiento de datos y la interfaz de usuario**.

### Arquitectura Basada en Atributos

Esta arquitectura se enfoca en **satisfacer ciertos atributos de calidad como la seguridad, el rendimiento o la disponibilidad**. Los componentes y sus interacciones se diseñan para optimizar estos atributos específicos.

## Estilos Peer-to-Peer

### Arquitectura Basada en Eventos

En esta arquitectura, los **componentes se comunican mediante el envío y recepción de eventos**. Es adecuada para sistemas altamente distribuidos y descentralizados, donde la escalabilidad y la flexibilidad son esenciales.

### Arquitectura Orientada a Servicios (SOA)

SOA organiza el sistema en **servicios independientes que interactúan entre sí a través de interfaces bien definidas**. Cada servicio realiza una función específica y puede ser reutilizado en diferentes aplicaciones, facilitando la integración y la escalabilidad.

### Arquitectura Basada en Recursos

Este estilo se centra en **recursos** identificables y sus representaciones. Los **componentes interactúan mediante operaciones estándar sobre estos recursos**, lo cual es común en arquitecturas RESTful utilizadas en el desarrollo de servicios web.