Resumen unidad 1 y 2 de IIS

Unidad 1- Fundamentos de Ingeniería

Ingeniería:

"Profesión en la que un conocimiento de las matemáticas y de las ciencias naturales obtenida por la experiencia, el estudio y la práctica se aplica con criterio para desarrollar medios, a fin de usar, económicamente, los materiales y las fuerzas de la naturaleza para el beneficio de la humanidad."

P. Grech Mayor. Introducción a la ingeniería. Un enfoque a través del diseño.

Ingeniero:

"Persona que por razón de su especial conocimiento y uso de las matemáticas, física y ciencias de la ingeniería, los principios, método del análisis, diseño en ingeniería, adquiridos por educación y experiencia, está calificado para ejercer la ingeniería."

P. Grech Mayor. Introducción a la ingeniería. Un enfoque a través del diseño

---

Problema: Surge del deseo de transformar un estado, forma o condición de las cosas a otro.

Clasificaciones de problema:

- *Situación problemática* ¿Cuál es el problema subyacente?
- *Oportunidad o nueva situación* ¿Cuál es el problema que se plantea?
- *Problema* ¿Cuál es la solución?
- *CERRADOS:* admiten una única solución.
- *ABIERTOS:* admiten múltiples soluciones. Aquella que satisface (restricciones) y optimiza un conjunto de condiciones (criterios de selección) resulta la mejor.

Ciencia e ingeniería:

Ingeniería	Ciencia
• Desarrolla producto final: artefacto, estructura, proceso, sistema de sw … • Estudia factibilidad económica • Busca seguridad humana • Desea aceptación del público • Asegura manufacturabilidad de sus obras	• Formula hipótesis para explicar fenómenos • Obtiene datos para probar teorías • Concibe, planea, prepara y ejecuta experimentos • Analiza observaciones y deduce conclusiones

Proceso básico: diseño *Objetivo y producto final:* desarrollo de productos mediante el diseño	• Intenta generalizar aprendizaje • Divulga los descubrimientos *Proceso básico:* investigación *Objetivo y producto final:* cuerpo de conocimiento

Método científico

1. Identificación de una anomalía
2. Recolección de datos significativos
3. Análisis de los datos
4. Elaboración de una explicación o la hipótesis
5. Predicción de eventos futuros con base en la hipótesis
6. Elaboración de experimentos para comprobar las predicciones
7. Modificación de la hipótesis y repetición de los pasos anteriores
8. Conversión de la hipótesis en teoría

Perfil del ingeniero

Conocimientos	Habilidades	Actitudes
◉ *Ciencias básicas* Matemática, física, química ◉ *Ciencia aplicada* ¿cómo? ¿dónde? ◉ *Conocimientos empíricos* Experiencia, observaciones ◉ *Especializaciones*	◉ Creatividad ◉ Pensamiento convergente y divergente ◉ Capacidad analítica ◉ Capacidad de trabajar en grupo ◉ Interdisciplinaridad ◉ Diseño conceptual ◉ Capacidad de comunicación ◉ Dominio de idioma técnico ◉ Aspecto humanístico	◉ Interrogante, curioso ◉ Permitirse dudar ◉ Objetividad ◉ Actitud profesional: servicio, responsabilidad, ética ◉ Mente abierta ◉ Continuar automejoramiento

1) Proceso de resolución de problemas

Proceso

a) Formulación del problema
b) Análisis del problema
c) Búsqueda de soluciones
d) Elección
e) Especificación de la solución

✔ **Formulación del problema:**
- Detectarlo: ¿Hay un problema? ¿Cuál es? ¿En qué consiste?
- No se recibe un enunciado detallado.
- Hacer preguntas.
- Describir el problema.
- Dar el detalle justo: clarificar el problema manteniendo descripciones amplias (mente abierta).
- Método de la "caja negra".

✔ **Análisis del problema**
- Recopilación de información relevante y detallada sobre el problema y las restricciones
- ¿Problema abierto o cerrado?
- Opiniones
- Descomposición del problema
- Descripciones detalladas
- Estado de origen y estado deseado
- Restricciones a cumplir y criterios para elegir
  - ✔ Técnicos
  - ✔ Económicos
  - ✔ De tiempo
  - ✔ Estéticos
  - ✔ Humanos
  - ✔ De seguridad
  - ✔ …

✔ **Búsqueda de soluciones**
- Conjunto de requerimientos, variables.
- Diferentes valores para cada variable o requerimiento en la solución.
- Búsqueda de soluciones parciales.
- Búsqueda de opciones.
- Imaginación, lluvia de ideas
- Analizar variables de a una por vez.
- Combinaciones de soluciones parciales.

✔ **Elección de la solución**
- Criterios de selección → Atributos
- Relevancia de cada atributo.
- Impacto de la solución en el atributo.

- Impacto de cada atributo.
- Comparación → Relación Costo/Beneficio
- Elección.

✔ **Especificación**

- ¡Solución elegida!
- Descripción detallada de todas las características de la solución.

Evaluación: factibilidad, rentabilidad, …

Construcción.

Controles/Monitoreo

- Modelos.
- Informes.

Proceso en Ing. de Software

Implementación	**Mantenimiento**	**Proceso en Ing. de software**
✔ Desarrollo de un nuevo producto ✔ Elaboración de documentos de uso y desarrollo ✔ Capacitación a usuarios ✔ Puesta en funcionamiento y control de soluciones existentes ✔ Certificación y auditoría de sistemas en producción	✔ Modificaciones de la implementación ✔ Incorporación de nueva funcionalidad ✔ Actualización de funcionalidades existentes ✔ Corrección de errores ✔ Migración a versiones nuevas de hw o sw ✔ Adaptación a nuevos entornos ✔ Integración con otros productos	✔ Formulación del problema ✔ Análisis del problema ✔ Búsqueda de soluciones ✔ Elección ✔ Especificación de la solución ✔ Implementación ✔ Mantenimiento

---

Solución

Criterio	**Restricción**
✔ norma o parámetro para la selección ✔ Suelen ser comunes. ✔ Varía su prioridad. ✔ Sus valores pueden oscilar en un rango	✔ Característica pre-fijada de la solución ✔ Requisito legal, naturaleza, … ✔ Condición que la solución al problema DEBE cumplir. ✔ MUY importante cuestionarlas

Comparación de soluciones

	Criterio 1	Criterio 2	...	Criterio n	Total
	Peso: p1%	*Peso: p2%*		*Peso: pn%*	*100%*
Solución A	*A1 * p1*	*A2 * p2*		*An * pn*	*T1*
Solución B	*B1 * p1*	*B2 * p2*		*Bn * pn*	*T2*
Solución C	*C1 * p1*	*C2 * p2*		*Cn * pn*	*T3*

1. Otorgamos una valoración numérica a cada uno
2. Multiplicamos por el peso del criterio
3. Sumamos el total de impacto de criterios para cada solución

---

Modelos de representación

¿Qué son?: Descripciones de la naturaleza o comportamiento de un objeto real.

¿Para qué usarlos?:

- ✔ Abstracción de pensamiento
- ✔ Comunicación
- ✔ Predicción
- ✔ Control
- ✔ Aprendizaje y entrenamiento

Tipos de modelos:

Físicas o icónicas	**Gráficas**	**Esquemáticas**	**Matemáticas**	**Simulaciones**
☺ Semejanza física con objeto real ☺ Representaciones en dos o tres dimensiones ☺ Conserva proporciones	☺ Visualizan relaciones y magnitudes relativas	☺ Representan de manera simbólica un objeto real	☺ Predicción y comunicación	☺ Experimentación usando una representación de un objeto real: • Físicas o icónicas • Analógicas • Digitales

¿Cuál/cuáles usar?:

Preguntarnos:

- ¿para qué?
- ¿para quién?

- ¿características del objeto real?

Mediciones

¿POR QUÉ medir?

Se desea:

- Comprender lo que se está realizando en el proceso de desarrollo.
- Hacer más visibles determinados aspectos del proceso y del producto.
- Conocer la complejidad del producto y del proceso.
- Realizar estimaciones de costos y de esfuerzo.
- Controlar los procesos y los proyectos.
- Incorporar mejoras en los procesos y en los productos

¿Qué medir?

Entidad: Objeto o evento del mundo real.

Atributo: Característica o propiedad de una entidad.

Medición: Proceso mediante el cual se asignan números o símbolos a los atributos de entidades del mundo real, de manera tal que se los describa de acuerdo a reglas definidas

- Cuantificación directa de los atributos: **medición**
- Cuantificación indirecta, combinación de más de una medición en un elemento que refleja el atributo que intentamos comprender: **cálculo**

Medida: Elemento que proporciona un indicio cuantitativo de la extensión, cantidad, dimensión, capacidad o tamaño de algún atributo de un producto o proceso. La medición es el acto de determinar una medida.

Métrica: Medida cuantitativa del grado en el que un sistema, componente o proceso posee un atributo determinado.

Método de medición elegido junto con la escala de medición.

Indicador: Métrica o combinación de métricas que proporcionan comprensión acerca del proceso, el proyecto o el producto en sí.

Los indicadores proporcionan comprensión para realizar ajustes, para incorporar

cambios que permitan hacer mejor las cosas.

Teoría de la representación

- ☺ Los datos obtenidos mediante mediciones deben representar los atributos observados.
- ☺ La manipulación de los datos debe preservar las relaciones observables entre las entidades.

Medición: mapeo del mundo empírico al mundo formal, o relacional.

Medida: el número o símbolo asignado por el mapeo a una entidad para caracterizar un atributo.

- ☺ Condición de representación: un mapeo M debe mapear entidades a números y relaciones empíricas a relaciones numéricas de manera que las relaciones empíricas preserven y sean preservadas por las relaciones numéricas.

Escalas de medición

Al mapeo de medición, junto con el sistema empírico y el sistema de relaciones numéricas lo llamamos *escala de medición.*

- ✵ De acuerdo a sus características identificamos cinco tipos de escala:
  Escala nominal: Se definen clases o categorías en las que se ubican las entidades de acuerdo al valor del atributo medido.
  La escala nominal tiene dos características principales:
  - ✔ El sistema de relación empírico consiste sólo de diferentes clases, no hay noción de orden entre ellas.
  - ✔ Cualquier representación simbólica o numérica para las distintas clases es aceptable. No hay noción de magnitud asociada a los números o símbolos

  Escala ordinal

  Define clases o categorías, al igual que la escala nominal, pero agrega información de ordenamiento de las clases.

  Características de la escala ordinal:

  - ✔ El sistema de relación empírico consiste de diferentes clases ordenadas con respecto al atributo.
  - ✔ Cualquier mapeo que preserve el ordenamiento es aceptable.
  - ✔ Los números solo representan un ranking (no tiene sentido sumar, restar o cualquier operación aritmética)

  Escala de intervalos

Esta escala captura información sobre el tamaño de los intervalos que separan a cada una de las clases ordenadas, el "salto" entre clase y clase.
Las características de la escala de intervalos son:

- ✔ Preserva el orden
- ✔ Preserva las diferencias (pero no proporciones)
- ✔ Se aceptan operaciones de suma y resta.

Escala de proporciones

Brinda información acerca de las diferencias de proporciones existentes entre cada clase.
Características de la escala de proporciones:

- ✔ Preserva el orden, el tamaño de los intervalos y las proporciones de los intervalos entre entidades.
- ✔ Existe el elemento cero, representa la ausencia total del atributo.
- ✔ El mapeo debe comenzar en cero e incrementarse en intervalos iguales llamados unidades.
- ✔ Todas las operaciones aritméticas son válidas.

Escala absoluta

No admite transformaciones excepto la identidad.
Características de la escala absoluta:

- ✔ La medición se realiza contando el número de elementos.
- ✔ Siempre toma la forma "número de ocurrencias de x en la entidad".
- ✔ Solo hay un mapeo posible de medición: la cuenta.
- ✔ Todas las operaciones aritméticas son significativas.

Unidad 2- Ingeniería de software

Software

Es producto y herramienta para la construcción de un producto al mismo tiempo.

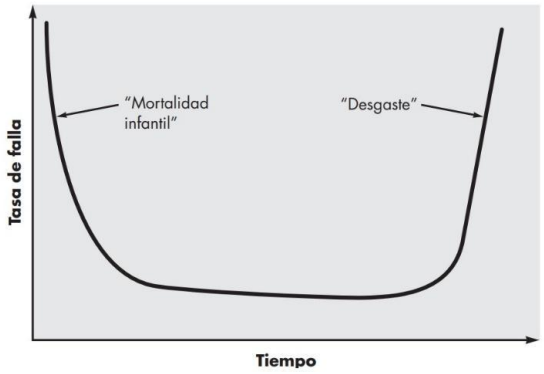
1- Instrucciones que cuando se ejecutan proporcionan las características, función y desempeño buscados.
2- Estructuras de datos que permiten que los programas manipulen en forma adecuada la información.
3- Información descriptiva que describe la operación y uso de los programas.

Software – Producto

¿En qué difiere de otros productos?

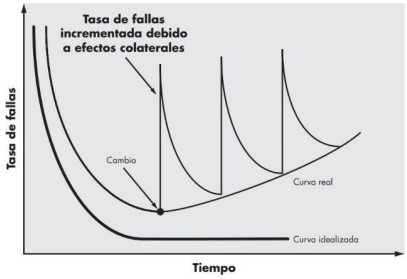
- Es producto y herramienta para la construcción de un producto al mismo tiempo.
- Se desarrolla y/o modifica utilizando el intelecto, no se fabrica en el sentido tradicional.
- El software no se “desgasta”.
- Utilización de componentes.

FIGURA 1.1
Curva de fallas del hardware



La figura 1.1 ilustra la tasa de falla del hardware como función del tiempo. La relación, que es frecuente llamar “curva de tina”, indica que el hardware presenta una tasa de fallas relativamente elevada en una etapa temprana de su vida (fallas que con frecuencia son atribuibles a defectos de diseño o manufactura); los defectos se corrigen y la tasa de fallas se abate a un nivel estable (muy bajo, por fortuna) durante cierto tiempo. No obstante, conforme pasa el tiempo, la tasa de fallas aumenta de nuevo a medida que los componentes del hardware resienten los efectos acumulativos de suciedad, vibración, abuso, temperaturas extremas y muchos otros inconvenientes ambientales. En pocas palabras, el hardware comienza a desgastarse.

FIGURA 1.2
Curvas de falla del software



El software no es susceptible a los problemas ambientales que hacen que el hardware se desgaste. Por tanto, en teoría, la curva de la tasa de fallas adopta la forma de la “curva idealizada” que se aprecia en la figura 1.2. Los defectos ocultos ocasionarán tasas elevadas de fallas al comienzo de la vida de un programa. Sin embargo, éstas se corrigen y la curva se aplana, como se indica. La

curva idealizada es una gran simplificación de los modelos reales de las fallas del software. Aun así, la implicación está clara: el software no se desgasta, ¡pero sí se deteriora!

Esta contradicción aparente se entiende mejor si se considera la curva real en la figura 1.2. Durante su vida, el software sufrirá cambios. Es probable que cuando éstos se realicen, se introduzcan errores que ocasionen que la curva de tasa de fallas tenga aumentos súbitos, como se ilustra en la "curva real" (véase la figura 1.2). Antes de que la curva vuelva a su tasa de fallas original de estado estable, surge la solicitud de otro cambio que hace que la curva se dispare otra vez. Poco a poco, el nivel mínimo de la tasa de fallas comienza a aumentar: el software se está deteriorando como consecuencia del cambio.
Otro aspecto del desgaste ilustra la diferencia entre el hardware y el software.

Cuando un componente del hardware se desgasta es sustituido por una refacción. En cambio, no hay refacciones para el software. Cada falla de éste indica un error en el diseño o en el proceso que tradujo el diseño a código ejecutable por la máquina. Entonces, las tareas de mantenimiento del software, que incluyen la satisfacción de peticiones de cambios, involucran una complejidad considerablemente mayor que el mantenimiento del hardware.

Dominios de aplicación del software

Actualmente, hay siete grandes categorías de software de computadora que plantean retos continuos a los ingenieros de software:

- *Software de sistemas:* conjunto de programas escritos para dar servicio a otros programas. Determinado software de sistemas (por ejemplo, compiladores, editores, etc) procesa estructuras de información complejas pero deterministas. Otras aplicaciones de sistemas (por ejemplo, componentes de sistemas operativos, software de redes) procesan sobre todo datos indeterminados. En cualquier caso, el área de software de sistemas se caracteriza por: gran interacción con el hardware de la computadora, uso intensivo por parte de usuarios múltiples, operación concurrente que requiere la secuenciación, recursos compartidos y administración de un proceso sofisticado, estructuras complejas de datos e interfaces externas múltiples.

- *Software de aplicación:* programas aislados que resuelven una necesidad específica de negocios. Las aplicaciones en esta área procesan datos comerciales o técnicos en una forma que facilita las operaciones de negocios o la toma de

decisiones administrativas o técnicas. Además de las aplicaciones convencionales de procesamiento de datos, el software de aplicación se usa para controlar funciones de negocios en tiempo real.

- *Software de ingeniería y ciencias:* se ha caracterizado por algoritmos "devoradores de números". Las aplicaciones van de la astronomía a la vulcanología, del análisis de tensiones en automóviles a la dinámica orbital del transbordador espacial, y de la biología molecular a la manufactura automatizada. Sin embargo, las aplicaciones modernas dentro del área de la ingeniería y las ciencias están abandonando los algoritmos numéricos convencionales. El diseño asistido por computadora, la simulación de sistemas y otras aplicaciones interactivas, han comenzado a hacerse en tiempo real e incluso han tomado características del software de sistemas.

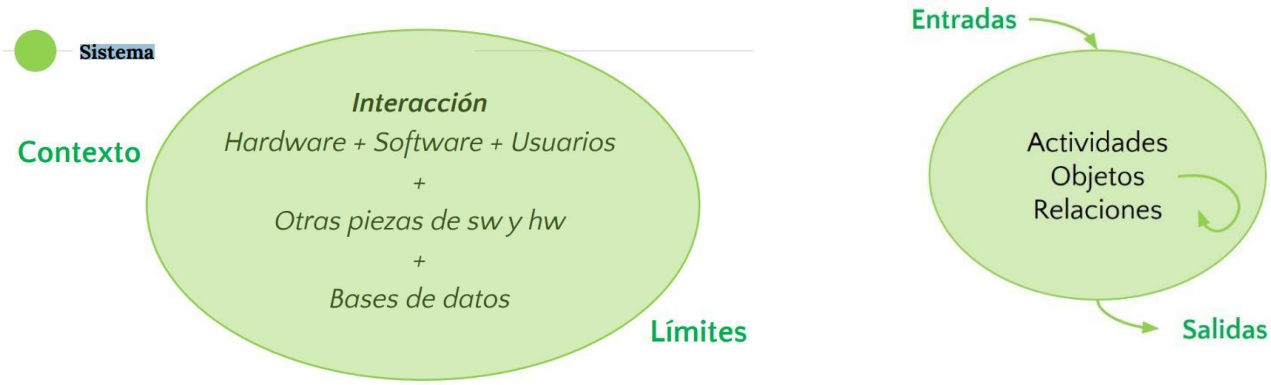
- *Software incrustado:* reside dentro de un producto o sistema y se usa para implementar y controlar características y funciones para el usuario final y para el sistema en sí. El software incrustado ejecuta funciones limitadas y particulares (por ejemplo, control del tablero de un horno de microondas) o provee una capacidad significativa de funcionamiento y control (funciones digitales en un automóvil, como el control del combustible, del tablero de control y de los sistemas de frenado).
- *Software de línea de productos:* es diseñado para proporcionar una capacidad específica para uso de muchos consumidores diferentes. El software de línea de productos se centra en algún mercado limitado y particular (por ejemplo, control del inventario de productos) o se dirige a mercados masivos de consumidores.

- *Aplicaciones web:* llamadas "webapps", esta categoría de software centrado en redes agrupa una amplia gama de aplicaciones. En su forma más sencilla, las webapps son poco más que un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas limitadas. Sin embargo, desde que surgió Web 2.0, las webapps están evolucionando hacia ambientes de cómputo sofisticados que no sólo proveen características aisladas, funciones de cómputo y contenido para el usuario final, sino que también están integradas con bases de datos corporativas y aplicaciones de negocios.

- *Software de inteligencia artificial:* hace uso de algoritmos no numéricos para resolver problemas complejos que no son fáciles de tratar computacionalmente o

con el análisis directo. Las aplicaciones en esta área incluyen robótica, sistemas expertos, reconocimiento de patrones (imagen y voz), redes neurales artificiales, demostración de teoremas y juegos.

Sistema:



Actividades: lo que sucede en el sistema.

Objetos: o entidades, elementos involucrados en las actividades.

Relaciones: entre los objetos y las actividades en las que participan.

Límite: o frontera del sistema, limita el sistema de su contexto.

Construcción de un sistema

- ✔ Proceso análogo.
- ✔ Clientes: presentan sus deseos y necesidades
- ✔ Equipo: desarrolla planos y modelos
- ✔ Presentan: pantallas, descripciones de uso
- ✔ Se discuten detalles de apariencia y funcionalidad
- ✔ Diseño global aprobado –> discuten detalles de implementación.
- ✔ Código escrito -> comienzan pruebas unitarias
- ✔ Integrar partes
- ✔ Pruebas de integración
- ✔ Producto final: se verifica que los requerimientos se hayan cumplido
- ✔ El proceso se describe de manera lineal. En la práctica es común que los pasos se repitan.
- ✔ Esto da lugar a diferentes procesos de desarrollo.

Ingeniería de software

La aplicación de un método sistemático, disciplinado y cuantificable para el desarrollo, la operación y el mantenimiento de software; esto es, la aplicación de la ingeniería al software y el estudio de los métodos mencionados.

El establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales.

Método: técnica formal para producir un resultado.

Paradigma: enfoque particular o filosofía.

Procedimiento: combinación de herramientas y técnicas que producen un resultado. Formas de concretar el método.

Herramienta: instrumento o sistema automatizado para realizar una tarea de mejor manera.

Participantes del proceso-Roles

Cliente: Desarrollo para alguien que lo necesita. Quien solicita y paga el sistema que se va a crear

Desarrollador: Quien construye (persona, compañía, organización) el software para el cliente es el desarrollador.

Usuario: Quien utilizará el software creado. Tiene necesidades específicas sobre el sistema.

Participantes – Miembros del equipo

Analistas de requerimientos -Análisis: determinar qué desea el cliente, documentar requerimientos.

Diseñadores - Describir qué debe hacer el sistema.

Programadores - Implementar los requerimientos en código.

Testers - Detectar defectos.

Terminado el trabajo, se entrega al cliente

Capacitadores - Entrenan al cliente en el uso del sistema.

Equipo de mantenimiento – trabaja en corregir defectos y/o cambiar aspectos del sistema con el correr del tiempo

Proceso

Conjunto de actividades, acciones y tareas a ejecutar para crear algún producto.

- *Actividad*: objetivo amplio, se desarrolla en cualquier dominio de aplicación, tamaño del proyecto o grado de rigor del trabajo
- *Acción*: conjunto de tareas para obtener un producto del trabajo
- *Tarea*: objetivo pequeño, bien definido, con resultado tangible

Características de un proceso

1) Establece las principales *actividades*
2) Utiliza *recursos* y está sujeto a *restricciones*
3) Está compuesto por *subprocesos* encadenados y/o jerarquizados
4) Cada actividad tiene criterios de *entrada* y de *salida*
5) Conjunto de *principios* orientadores que explican las *metas* de cada actividad

Proceso vs. Procedimiento

Procedimiento: combinación estructurada de herramientas y técnicas.

Proceso: conjunto de procedimientos.

Proceso del software

Estructura general:

1- *Comunicación:* Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente (y con otros participantes). Se busca entender los objetivos de los participantes respecto del proyecto, y reunir los requerimientos que ayuden a definir las características y funciones del software.

2- *Planificación:* Cualquier viaje complicado se simplifica si existe un mapa. Un proyecto de software es un viaje difícil, y la actividad de planeación crea un "mapa" que guía al equipo mientras viaja. El mapa llamado plan del proyecto de software, define el trabajo de ingeniería de software al describir las tareas técnicas por realizar, los riesgos probables, los recursos que se requieren, los productos del trabajo que se obtendrán y una programación de las actividades.

3- *Modelado:* Un ingeniero de software hace lo mismo al crear modelos a fin de entender mejor los requerimientos del software y el diseño que los satisfará.

4- *Construcción:* Esta actividad combina la generación de código (ya sea manual o automatizada) y las pruebas que se requieren para descubrir errores en éste.

5- *Despliegue:* El software (como entidad completa o como un incremento parcialmente terminado) se entrega al consumidor que lo evalúa y que le da retroalimentación, misma que se basa en dicha evaluación.

Actividades sombrilla

Se aplican a lo largo del proyecto para administrar y controlar:

- Avance
- Calidad
- Cambio
- Riesgo

Ejemplos:

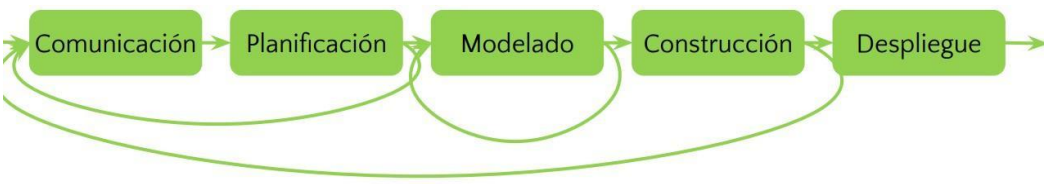
- Seguimiento y control
- Administración del riesgo
- Aseguramiento de la calidad
- Revisiones técnicas
- Medición
- Administración de la configuración

Flujo del proceso

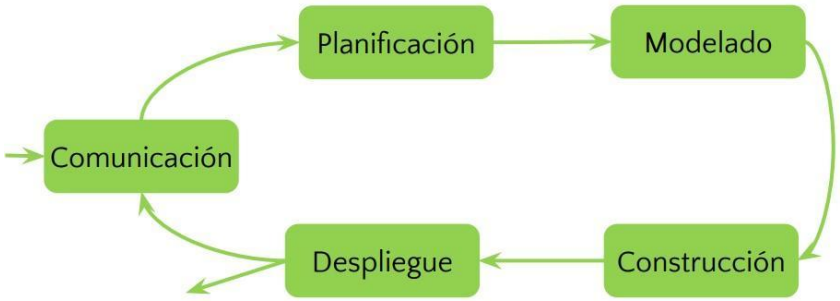
Organización de las actividades y sus tareas con respecto a secuencia y tiempo:

☺ ***Lineal:*** Un flujo de proceso lineal ejecuta cada una de las cinco actividades estructurales en secuencia, comenzando por la comunicación y terminando con el despliegue. Secuencia de actividades.

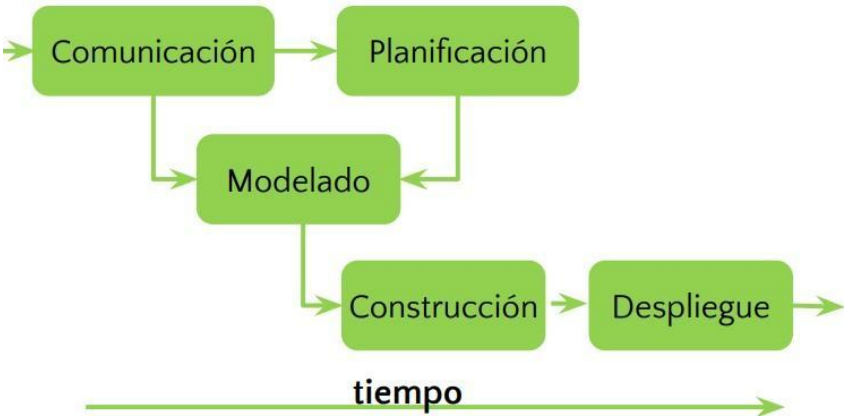
☺ ***Iterativo:*** Un flujo de proceso iterativo repite una o más de las actividades antes de pasar a la siguiente.



☺ ***Evolutivo:*** Un flujo de proceso evolutivo realiza las actividades en forma "circular". A través de las cinco actividades, cada circuito lleva a una versión más completa del software.



☺ ***Paralelo:*** Un flujo de proceso paralelo ejecuta una o más actividades en paralelo con otras (por ejemplo, el modelado de un aspecto del software tal vez se ejecute en paralelo con la construcción de otro aspecto del software).



Modelos de proceso

✔ ***Cascada:*** El modelo de la cascada, a veces llamado ciclo de vida clásico, sugiere un enfoque sistemático y secuencial para el desarrollo del software, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue, para concluir con el apoyo del software terminado.

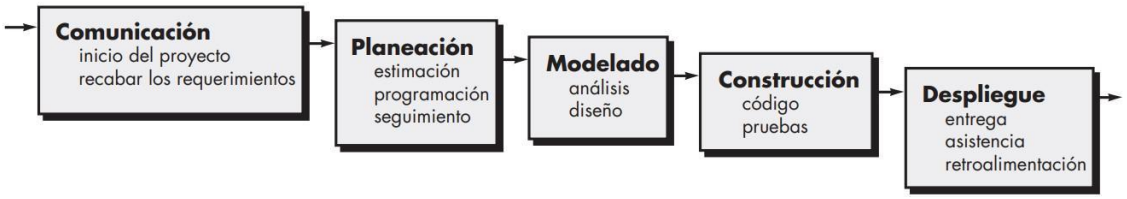
Ventajas:

- Es simple de explicar y entender para personas no familiarizadas con el desarrollo de software.
- Explicita productos intermedios necesarios para avanzar de etapa.

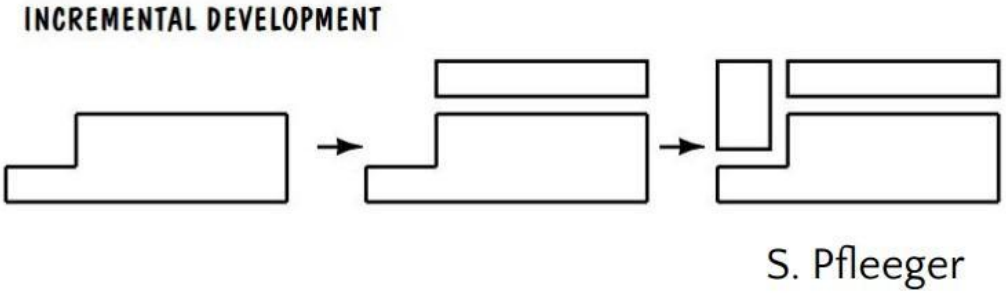
Desventajas:

- No refleja la realidad más frecuente. No es flexible, no se puede aplicar cuando hay requerimientos dinámicos.
- El cliente debe esperar por una versión funcional.

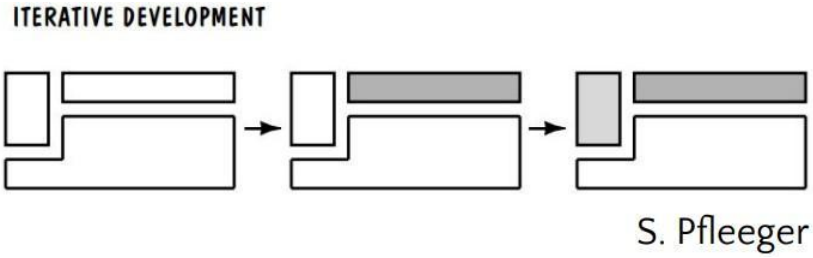
Modelo de la cascada



✔ ***Modelo incremental:*** Secuencias lineales aplicadas de manera escalonada. Puede haber un grado de paralelismo entre las distintas secuencias. Cada secuencia produce "incrementos" en el sw. Útil cuando no se dispone de personal suficiente para la implementación completa. Ayuda a la administración de riesgos técnicos.



✔ ***Modelo evolutivo o por iteraciones***: El sistema se entrega completo al comienzo. Las iteraciones modifican (mejoran, corrigen, extienden) la funcionalidad.



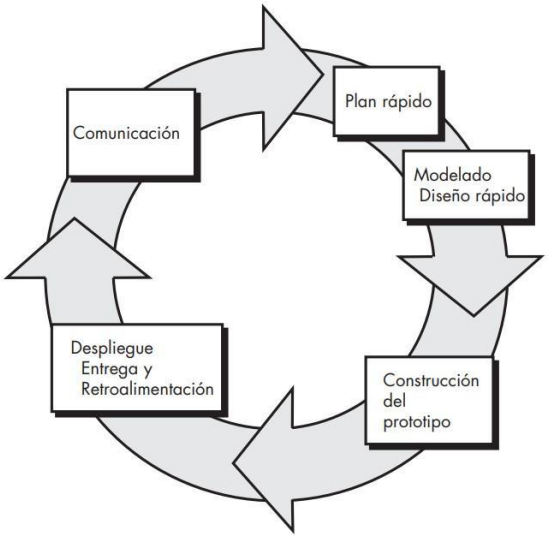
☺ *Prototipos*: Inicialmente se planea rápidamente una iteración para producir un primer prototipo. Cada iteración refina el conocimiento, el plan y el modelo, y genera un nuevo prototipo. Útil cuando inicialmente el cliente solo define objetivos generales y no identifica requerimientos detallados. Algunos prototipos son desechables, otros evolucionan.

Ventaja:

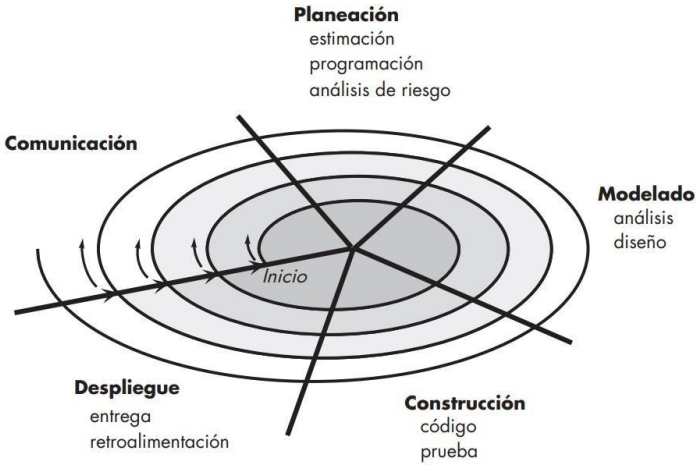
- Dar una idea real más temprana al usuario.
- Obtener mejor definición de los requerimientos.

Desventajas:

- Riesgo de perder de vista que lo que se obtiene es un prototipo y no el producto definitivo.



☺ *Espiral*: Se itera en forma de espiral. Las primeras iteraciones pueden producir modelos o prototipos. Cada iteración ajusta todos los documentos: plan, definición, modelos, costos, incluso el número de iteraciones necesarias. Es adaptable para usarse durante todo el ciclo de vida del sistema.



Modelos de proceso especializados

Tienen las características de los procesos vistos.

Se aplican para enfoques específicos.

- Desarrollo basado en componentes.
- Orientado a aspectos.
- Métodos formales

Métodos formales

- ✳ Especificación matemática del software y sus características
  - Invariantes
  - Precondiciones y postcondiciones
- ✳ Especificar, desarrollar y verificar los productos de manera rigurosa
- ✳ Lenguajes de especificación formal:
  - Sintáxis que define la notación
  - Semántica para definir universo de objetos que definen al sistema
  - Conjunto de relaciones de reglas semánticas para manipular los objetos

Ventajas

- ✔ Posible verificar y validar de manera rigurosa.
- ✔ Permite descubrir ambigüedades e inconsistencias.
- ✔ Baja costos de mantenimiento

Desventajas

- ✔ Pocos desarrolladores con la capacitación adecuada.
- ✔ Puede encarecer el desarrollo.
- ✔ Difícil de utilizar para comunicarse con los clientes.

Otros modelos de proceso

- Concurrente: El modelo de desarrollo concurrente, en ocasiones llamado ingeniería concurrente, permite que un equipo de software represente elementos iterativos y concurrentes de cualquiera de los modelos de proceso descritos en este capítulo. Por ejemplo, la actividad de modelado definida para el modelo espiral se logra por medio de invocar una o más de las siguientes acciones de software: hacer prototipos, análisis y diseño. El modelado concurrente es aplicable a todos los tipos de desarrollo de software y proporciona un panorama apropiado del estado actual del proyecto. En lugar de confinar las actividades, acciones y tareas de la ingeniería de software a una secuencia de eventos, define una red del proceso.
- Proceso unificado de software
- Metodologías ágiles

Metodologías ágiles

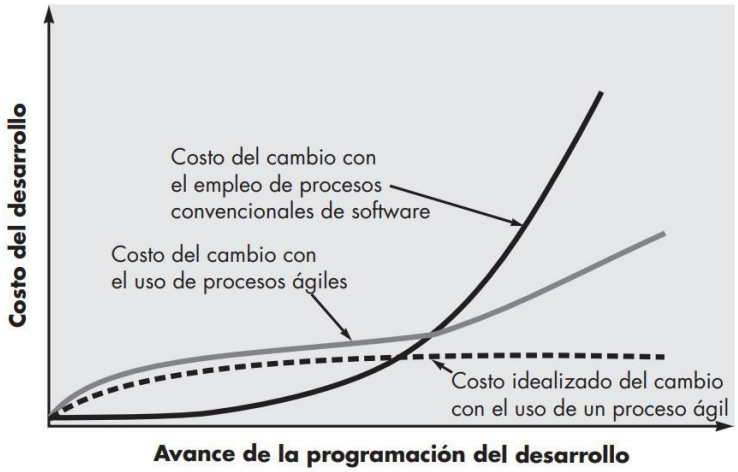
Grupo conocido como “Alianza Ágil” firma:

Estamos descubriendo formas mejores de desarrollar software, desarrollándolo y ayudando a otros a desarrollarlo. Ese trabajo nos ha hecho valorar:

- Los individuos y sus interacciones más que a los procesos y las herramientas.
- El software funciona más que la documentación exhaustiva.
- La colaboración con el cliente más que la negociación de contratos.
- Responder al cambio más que seguir un plan.

Costo del cambio

El costo se incrementa de manera no lineal con el avance del proyecto.



Proceso ágil

Cualquier proceso del software ágil se caracteriza por la forma en la que aborda cierto número de suposiciones clave acerca de la mayoría de proyectos de software:

1) Es difícil predecir qué requerimientos de software persistirán y cuáles cambiarán. También es difícil pronosticar cómo cambiarán las prioridades del cliente a medida que avanza el proyecto.
2) Para muchos tipos de software, el diseño y la construcción están imbricados. Es decir, ambas actividades deben ejecutarse en forma simultánea, de modo que los modelos de diseño se prueben a medida que se crean. Es difícil predecir cuánto diseño se necesita antes de que se use la construcción para probar el diseño.

3) El análisis, el diseño, la construcción y las pruebas no son tan predecibles como nos gustaría (desde un punto de vista de planeación).

Un proceso ágil debe ser adaptable. Pero la adaptación continua logra muy poco si no hay avance. Entonces, un proceso de software ágil debe adaptarse incrementalmente. Para lograr la adaptación incremental, un equipo ágil requiere retroalimentación con el cliente. Deben entregarse incrementos de software (prototipos ejecutables o porciones de un sistema operativo) en periodos cortos de tiempo, de modo que la adaptación vaya a ritmo con el cambio (impredecible). Este enfoque iterativo permite que el cliente evalúe en forma regular el incremento de software, dé la retroalimentación necesaria al equipo de software e influya en las adaptaciones del proceso que se realicen para aprovechar la retroalimentación.

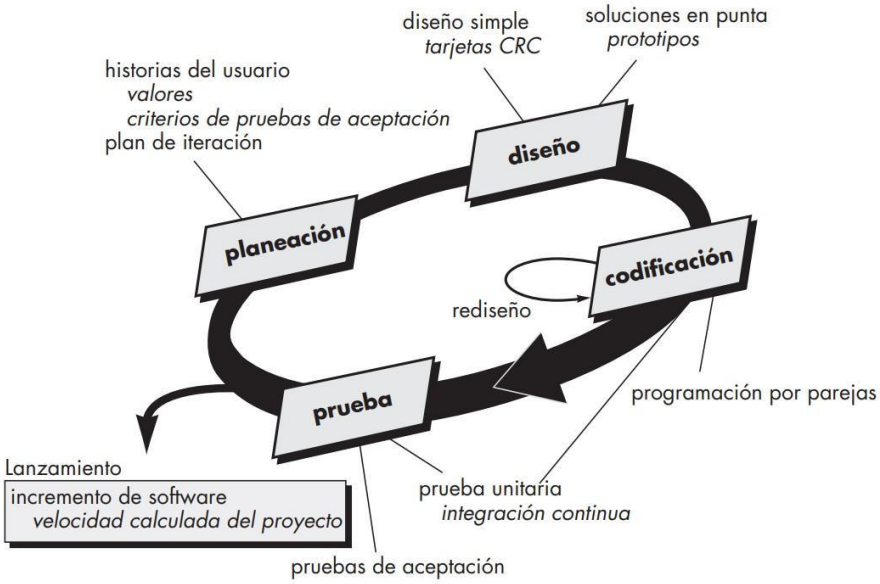
Diferentes modelos enfatizan distintos principios. Todos siguen un "espíritu ágil". Enfatizan la importancia de factores personales:

- Competencia
- Enfoque común
- Colaboración
- Habilidad para tomar decisiones
- Capacidad para resolver problemas difusos
- Confianza y respeto mutuos
- Organización propia

Programación Extrema (XP)

Pone el énfasis en la colaboración estrecha pero informal entre clientes y desarrolladores para establecer metáforas que comuniquen conceptos importantes, en la retroalimentación continua y en evitar documentación voluminosa. La retroalimentación se obtiene de tres fuentes: el software implementado, el cliente y otros miembros del equipo de software. El grado en el que el software implementa la salida, función y comportamiento del caso de uso es una forma de retroalimentación. Por último, conforme se obtienen nuevos requerimientos como parte de la planeación iterativa, el equipo da al cliente una retroalimentación rápida con miras al costo y al efecto en la programación de actividades.
La programación extrema usa un enfoque orientado a objetos como paradigma preferido de desarrollo, y engloba un conjunto de reglas y

prácticas que ocurren en el contexto de cuatro actividades estructurales: planeación, diseño, codificación y pruebas.



- *Planificación:*
  - ✔ Recabar requerimientos
  - ✔ Crear historias y asignarles valores,
  - ✔ Asignar costo medido en semanas de desarrollo.
  - ✔ Luego de la primera entrega se calcula “velocidad”.

- *Diseño:* Un diseño sencillo siempre se prefiere sobre una representación más compleja. Además, el diseño guía la implementación de una historia conforme se escribe: nada más y nada menos. Se desalienta el diseño de funcionalidad adicional porque el desarrollador supone que se requerirá después. Si en el diseño de una historia se encuentra un problema de diseño difícil, XP recomienda la creación inmediata de un prototipo operativo de esa porción del diseño. Entonces, se implementa y evalúa el prototipo del diseño, llamado solución en punta. El objetivo es disminuir el riesgo cuando comience la implementación verdadera y validar las estimaciones originales para la historia que contiene el problema de diseño. Rediseñar significa que el diseño se hace de manera continua conforme se construye el sistema.

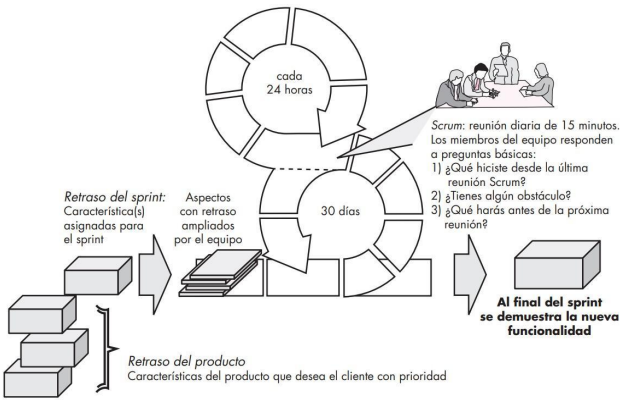
- *Codificación:* Después de que las historias han sido desarrolladas y de que se ha hecho el trabajo de diseño preliminar, el equipo no

inicia la codificación, sino que desarrolla una serie de pruebas unitarias a cada una de las historias que se van a incluir en la entrega en curso. Una vez creada la prueba unitaria, el desarrollador está mejor capacitado para centrarse en lo que debe implementarse para pasar la prueba. Una vez que el código está terminado, se le aplica de inmediato una prueba unitaria, con lo que se obtiene retroalimentación instantánea para los desarrolladores. Un concepto clave durante la actividad de codificación es la programación por parejas. XP recomienda que dos personas trabajen juntas en una estación de trabajo con el objeto de crear código para una historia. Esto da un mecanismo para la solución de problemas en tiempo real y para el aseguramiento de la calidad también en tiempo real. También mantiene a los desarrolladores centrados en el problema de que se trate.

- *Pruebas:* Las pruebas unitarias que se crean deben implementarse con el uso de una estructura que permita automatizarlas (de modo que puedan ejecutarse en repetidas veces y con facilidad). Esto estimula una estrategia de pruebas de regresión siempre que se modifique el código. Las pruebas de aceptación XP, también llamadas pruebas del cliente, son especificadas por el cliente y se centran en las características y funcionalidad generales del sistema que son visibles y revisables por parte del cliente. Las pruebas de aceptación se derivan de las historias de los usuarios que se han implementado como parte de la liberación del software.

Scrum

Los principios Scrum son congruentes con el manifiesto ágil y se utilizan para guiar actividades de desarrollo dentro de un proceso de análisis que incorpora las siguientes actividades estructurales: requerimientos, análisis, diseño, evolución y entrega. Cada actividad realiza las tareas bajo un patrón de proceso llamado sprint.



Scrum acentúa el uso de un conjunto de patrones de proceso del software que han demostrado ser eficaces para proyectos con plazos de entrega muy apretados, requerimientos cambiantes y negocios críticos. Cada uno de estos patrones de proceso define un grupo de acciones de desarrollo:

- *Retraso:* Lista de prioridades de los requerimientos o características del proyecto que dan al cliente un valor del negocio. Es posible agregar en cualquier momento otros aspectos al retraso. El gerente del proyecto evalúa el retraso y actualiza las prioridades según se requiera.
- *Sprints:* Consiste en unidades de trabajo que se necesitan para alcanzar un requerimiento definido en el retraso que debe ajustarse en una caja de tiempo predefinida (lo común son 30 días). Durante el sprint no se introducen. Así, el sprint permite a los miembros del equipo trabajar en un ambiente de corto plazo, pero estable.
- *Reuniones Scrum:* Son reuniones breves (de 15 minutos, por lo general) que el equipo Scrum efectúa a diario. Hay tres preguntas clave que se pide que respondan todos los miembros del equipo:
  - ✔ ¿Qué hiciste desde la última reunión del equipo?
  - ✔ ¿Qué obstáculos estás encontrando?
  - ✔ ¿Qué planeas hacer mientras llega la siguiente reunión del equipo?

  Un líder del equipo, llamado maestro Scrum, dirige la junta y evalúa las respuestas de cada persona. La junta Scrum ayuda al equipo a descubrir los problemas potenciales tan pronto como sea posible. Asimismo, estas juntas diarias llevan a la "socialización del conocimiento", con lo que se promueve una estructura de equipo con organización propia.

- *Demostraciones preliminares:* Entregar el incremento de software al cliente de modo que la funcionalidad que se haya implementado pueda demostrarse al cliente y éste pueda evaluarla. Es importante notar que las demostraciones preliminares no contienen toda la funcionalidad planeada.

Ingeniería de requerimientos

Requerimiento: Característica del sistema o descripción de algo que el sistema es capaz de hacer con el objeto de satisfacer su propósito.

Ingeniería de requerimientos: La ingeniería de requerimientos proporciona el mecanismo apropiado para entender lo que desea el cliente, analizar las necesidades, evaluar la factibilidad, negociar una solución razonable, especificar la solución sin ambigüedades, validar la especificación y administrar los requerimientos a medida que se transforman en un sistema funcional. Incluye siete tareas diferentes: concepción, indagación, elaboración, negociación, especificación, validación y administración. Es importante notar que algunas de estas tareas ocurren en paralelo y que todas se adaptan a las necesidades del proyecto.

- ☺ *Concepción:* En ciertos casos, una conversación casual es todo lo que se necesita para desencadenar un trabajo grande de ingeniería de software. Pero en general, la mayor parte de proyectos comienzan cuando se identifica una necesidad del negocio o se descubre un nuevo mercado o servicio potencial. Identificar a los participantes. Reconocer los diferentes puntos de vista. Buscar la colaboración: identificar áreas de interés común y las de conflictos. Comenzar con las preguntas: centrar en el cliente y participantes, así como en metas y beneficios generales: ¿Quién …? ¿Cuál …? ¿Opciones …?
- ☺ *Indagación:* Algunos problemas que se encuentran cuando ocurre la indagación:
  - ✵ ***Problemas de alcance:*** La frontera de los sistemas está mal definida o los clientes o usuarios finales especifican detalles técnicos innecesarios que confunden, más que clarifican, los objetivos generales del sistema.
  - ✵ ***Problemas de entendimiento:*** Los clientes o usuarios no están completamente seguros de lo que se necesita, comprenden mal las capacidades y limitaciones de su ambiente de computación, no entienden todo el dominio del problema, tienen problemas para comunicar las necesidades al ingeniero de sistemas, omiten información que creen que es "obvia", especifican requerimientos que están en conflicto con las necesidades de otros clientes o usuarios, o solicitan requerimientos ambiguos o que no pueden someterse a prueba.
  - ✵ ***Problemas de volatilidad:*** Los requerimientos cambian con el tiempo.

Para superar estos problemas, debe enfocarse en la obtención de requerimientos en forma organizada.

Distinguir requerimientos

1) Que DEBEN ser absolutamente satisfechos. A veces llamados atributos o requerimientos críticos.
2) Deseables, pero no indispensables.
3) Posibles pero que podrían eliminarse.

- Requerimientos funcionales:
  - ¿Qué hará el sistema?
  - Describen interacciones entre el sistema y su entorno
- Requerimientos no funcionales (o restricciones)
  - Describen restricciones sobre el sistema que limitarán las elecciones en la construcción de la solución al problema.

- Es necesario entender cómo van a utilizar los usuarios finales las funciones y características.
- Escenarios y Casos de uso:
  - Captan contratos que describen el comportamiento del sistema en distintas condiciones en las que responde a peticiones de los participantes.
  - Narra una historia sobre la interacción con el usuario.
  - Se definen los actores de la historia.

☺ *Elaboración:* La información obtenida del cliente durante la concepción e indagación se expande y refina durante la elaboración. Esta tarea se centra en desarrollar un modelo refinado de los requerimientos que identifique distintos aspectos de la función del software, su comportamiento e información.
La elaboración está motivada por la creación y mejora de escenarios de usuario que describen cómo interactúa el usuario final con el sistema. Cada escenario de usuario se enuncia con sintaxis apropiada para extraer clases de análisis, que son entidades del dominio del negocio visibles para el usuario final. Se definen los atributos de cada clase de análisis y se identifican los servicios que requiere cada una de ellas. Se identifican las relaciones y colaboración entre clases, y se producen varios diagramas adicionales.

☺ *Negociación:* No es raro que los clientes y usuarios piden más de lo que puede lograrse dado lo limitado de los recursos del negocio. También es

relativamente común que distintos clientes o usuarios propongan requerimientos conflictivos con el argumento de que su versión es "esencial para nuestras necesidades especiales". Estos conflictos deben reconciliarse por medio de un proceso de negociación. Se pide a clientes, usuarios y otros participantes que ordenen sus requerimientos según su prioridad y que después analicen los conflictos. Con el empleo de un enfoque iterativo que da prioridad a los requerimientos, se evalúa su costo y riesgo, y se enfrentan los conflictos internos; algunos requerimientos se eliminan, se combinan o se modifican de modo que cada parte logre cierto grado de satisfacción.

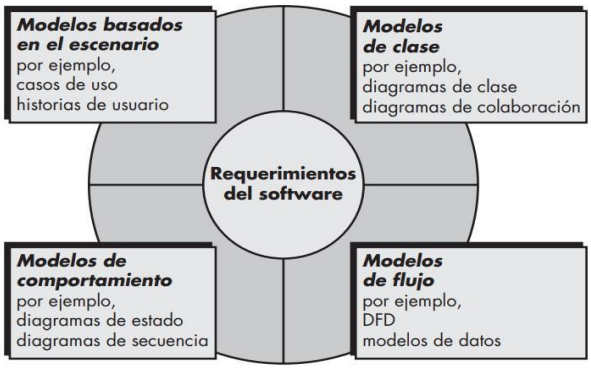
- ☺ *Especificación:* Una especificación puede ser un documento escrito, un conjunto de modelos gráficos, un modelo matemático formal, un conjunto de escenarios de uso, un prototipo o cualquier combinación de éstos. Algunos sugieren que para una especificación debe desarrollarse y utilizarse una "plantilla estándar", con el argumento de que esto conduce a requerimientos presentados en forma consistente y por ello más comprensible. Sin embargo, en ocasiones es necesario ser flexible cuando se desarrolla una especificación. Para sistemas grandes, el mejor enfoque puede ser un documento escrito que combine descripciones en un lenguaje natural con modelos gráficos.
- ☺ *Validación:* La validación de los requerimientos analiza la especificación a fin de garantizar que todos ellos han sido enunciados sin ambigüedades; que se detectaron y corrigieron las inconsistencias, las omisiones y los errores, y que los productos del trabajo se presentan conforme a los estándares establecidos para el proceso, el proyecto y el producto. El mecanismo principal de validación de los requerimientos es la revisión técnica.
- ☺ *Administración de los requerimientos:* La administración de los requerimientos es el conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y dar seguimiento a los requerimientos y a sus cambios en cualquier momento del desarrollo del proyecto.

Modelado y herramientas

Modelado de requerimientos:

- ☠ *Basados en el escenario*: Requerimientos desde el punto de vista de "actores" del sistema.
- ☠ *De datos*: Ilustran el dominio de información del problema.

- ☠ *Orientados a clases:* Representan objetos y la manera en la que colaboran para cumplir requerimientos del sistema.
- ☠ *Orientados al flujo:* Transformación de los datos a medida que avanzan en el sistema.
- ☠ *De comportamiento:* Modo en el que se comporta el software como consecuencia de eventos.



Expresión de requerimientos

- ☺ *Modelos UML:*
  - ✔ *Casos de uso:* Un caso de uso describe una funcionalidad o característica que el sistema proporciona a sus actores. Los actores son entidades externas al sistema que interactúan con él, como usuarios, sistemas externos, etc. Un caso de uso representa una secuencia de acciones que un sistema realiza, normalmente en respuesta a una solicitud de un actor, para lograr un resultado deseado.

    Cada caso de uso debe aportar un valor perceptible para algún actor del sistema. Pueden describirse en términos de metas o tareas que el actor desea lograr.

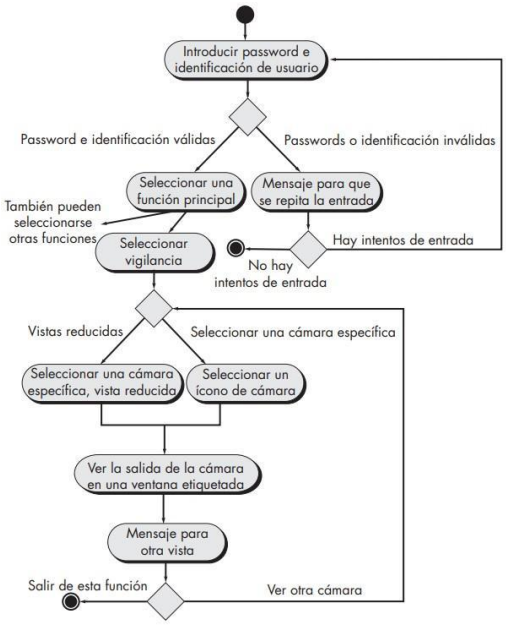
    Los diagramas de casos de uso son gráficos que representan los casos de uso de un sistema y cómo los actores interactúan con el sistema a través de estos casos de uso. Aquí te explico cómo se construyen estos diagramas:
    - *Actor:* Representa un rol que interactúa con el sistema. Pueden ser personas, otros sistemas, dispositivos, etc.
    - *Caso de Uso:* Representa una funcionalidad del sistema que proporciona algún valor a uno o más actores. Se muestra como una elipse.

- *Asociación:* Une a un actor con un caso de uso para indicar que el actor está involucrado en el caso de uso. La asociación se dibuja como una línea.
- *Inclusión:* Representa una relación en la que un caso de uso incluye la funcionalidad de otro caso de uso. Se muestra con una flecha punteada desde el caso de uso incluyente al caso de uso incluido.
- *Extensión:* Representa una relación en la que un caso de uso puede extenderse a otro caso de uso. Se muestra con una flecha punteada desde el caso de uso extensivo al caso de uso extendido.

El diagrama de casos de uso ayuda a comprender cómo los actores interactúan con el sistema y proporciona una visión general de las funcionalidades del sistema desde la perspectiva del usuario.

✔ *Diagrama de actividad:* El diagrama de actividad UML enriquece el caso de uso al proporcionar una representación gráfica del flujo de interacción dentro de un escenario específico. Un diagrama de actividades es similar a uno de flujo, y utiliza rectángulos redondeados para denotar una función específica del sistema, flechas para representar flujo a través de éste, rombos de decisión para ilustrar una ramificación de las decisiones (cada flecha que salga del rombo se etiqueta) y líneas continuas para indicar que están ocurriendo actividades en paralelo.

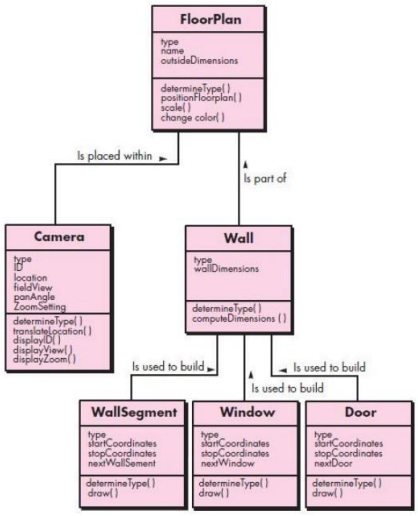
Por ejemplo, un usuario quizá sólo haga algunos intentos de introducir su identificación y password. Esto se representa por el rombo de decisión debajo de “Mensaje para que se repita la entrada”.



✔ *Diagramas de canal:* El diagrama de canal de UML es una variación útil del diagrama de actividades y permite representar el flujo de actividades descritas por el caso de uso; al mismo tiempo, indica qué actor (si hubiera muchos involucrados en un caso específico de uso) o clase de análisis es responsable de la acción descrita por un rectángulo de actividad. Las responsabilidades se representan con segmentos paralelos que dividen el diagrama en forma vertical, como los canales o carriles de una alberca.

- ☺ Modelado basado en clases
  - Objetos y Clases
  - Operaciones (métodos o servicios)
  - Relaciones entre los objetos
  - Colaboraciones entre clases
- ☺ Especificar atributos
- ☺ Definir operaciones
  - Que manipulan datos
  - Que realizan cálculos
  - Que consultan el estado del objeto
  - Que vigilan ocurrencias de eventos
- ☺ Identificar clases, objetos y atributos
  - Guiarse con: sustantivos y sinónimos

- ☺ Entidades externas que producen o consumen la información del sistema
- ☺ Cosas que forman parte del dominio de información
- ☺ Ocurrencias o eventos que suceden dentro del contexto



✔ *Diagramas de secuencia*: Estos diagramas representan la interacción entre objetos en un sistema en función del tiempo y se utilizan para visualizar el comportamiento de un sistema a través de la interacción de sus componentes a lo largo del tiempo. A continuación, te explicaré cómo se estructuran y representan los diagramas de secuencia en UML, seguido de un ejemplo para mayor claridad.

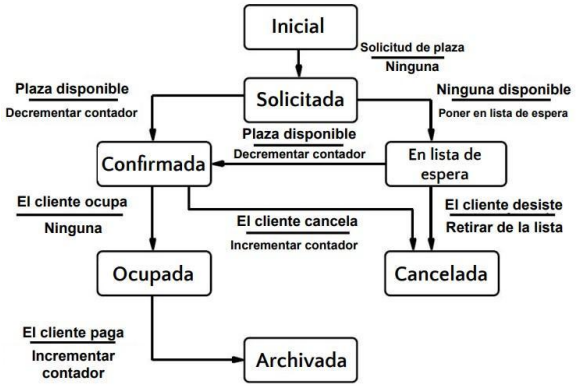
Estructura de un Diagrama de Secuencia

- ☺ *Actores:* Representan cualquier entidad externa al sistema con la que interactúa el sistema.
- ☺ *Objetos:* Representan instancias de clases que participan en la secuencia.
- ☺ *Líneas de Vida:* Representan el tiempo de vida de un objeto durante la secuencia. Están marcadas con una caja vertical.
- ☺ *Mensajes:* Representan la comunicación entre objetos y están etiquetados con un nombre y parámetros opcionales.
- ☺ *Activaciones:* Representan el período de tiempo en el que un objeto está realizando una operación.
- ☺ *Fragmentos:* Representan fragmentos condicionales o iterativos de la secuencia.

✔ *Diagramas de clases:* El modelado basado en clases representa los objetos que manipulan el sistema, las operaciones (también llamadas métodos o servicios) que se aplicarán a los objetos para efectuar la manipulación, las relaciones (algunas de ellas jerárquicas) entre los objetos y las colaboraciones que tienen lugar entre las clases definidas. Los elementos de un modelo basado en clases incluyen las clases y los objetos, atributos, operaciones, modelos clase-responsabilidad-colaborador, diagramas de colaboración y paquetes.

Modelos de comportamiento. Descripciones dinámicas

- **Descripciones funcionales y diagramas de transición:** Conjunto de estados y reacciones del sistema ante eventos. Sistema pensado como serie de funciones.



- **Tablas de decisión:** Condiciones posibles, reglas para reaccionar ante estímulos y acciones a ser tomadas como resultado.

	R1	R2	R3	R4	R5
Notas altas de examen	V	F	F	F	F
Grados superiores	-	V	F	F	F
Actividades externas	-	-	V	F	F
Buenas recomendaciones	-	-	-	V	F
Enviar carta de rechazo			X	X	X
Enviar formularios de admisión	X	X			

- **Diagramas de flujos de datos:** Permiten modelar el flujo de los datos hacia, en y desde el sistema.
- Existen muchas técnicas y notaciones.
- Otras:
  - Técnicas jerárquicas

- SREM: Metodología de la ingeniería de requerimientos de software
- Técnica de análisis y diseño estructurado
- Lenguajes de especificación formal.

Validación de los requerimientos

Proceso por el cual se determina si la especificación es consistente con la definición de los requerimientos.

- Primero: se asegura que cada especificación pueda ser rastreada hasta su requerimiento en el documento de definición.
- Luego: se chequea la definición para ver si cada requerimiento es rastreable hasta la especificación.

Técnicas de validación

Técnicas manuales	Lectura. Cruce de referencias manual. Entrevistas. Revisiones. Listas de comprobación. Modelos manuales para chequeo de funciones y relaciones. Escenarios. Pruebas matemáticas.
Técnicas automatizadas	Cruce de referencias automatizado. Modelos automatizados para poner en ejecución funciones. Prototipos.

Medición de los requerimientos

- Se requiere detalle del proceso de requerimientos y de la calidad de los requerimientos mismos.
- Se enfoca en tres áreas: producto, proceso, recursos
- Ejemplo: producto (definición y especificación)
  - Número de requerimientos
  - Cantidad de cambios introducidos
  - Si es posible tomar medidas por tipo de requerimiento

Requerimientos: características

- *Correctos:* sin errores
- *Consistentes:* sin conflictos ni ambigüedades
- *Completos:* externa e internamente completos
- *Realistas:* ¿puede hacerse realmente lo solicitado?

- *Necesarios:* sin restricciones innecesarias
- *Verificables:* ¿se puede probar que se cumplen?
- *Rastreables:* ¿se pueden identificar fácilmente?

Diseño

"Diseñar un sistema es determinar un conjunto de componentes y de interfaces entre componentes que satisfagan un conjunto específico de requerimientos."

DeMarco (1982)

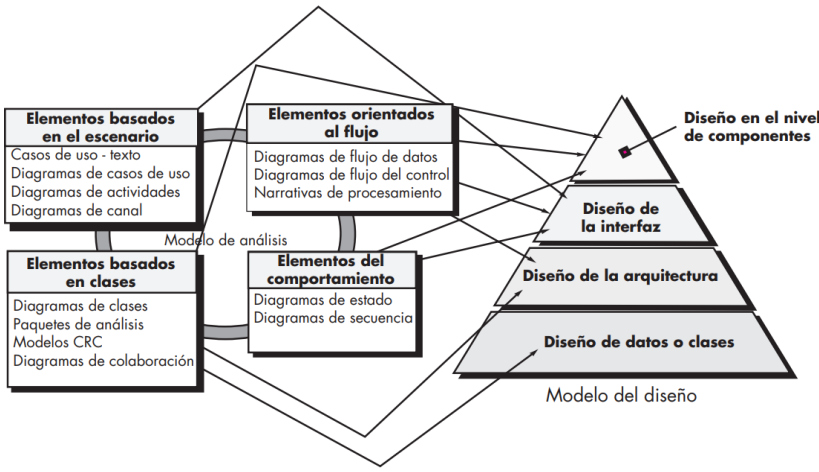
Descomposición

Toda descomposición separa el diseño en partes llamadas módulos o componentes. Un sistema es modular cuando una de las actividades la realiza un único componente y que además tiene bien definidas todas sus entradas y salidas.

- ★ *Descomposición modular:* descripciones de alto nivel de las funciones de cada componente. La descomposición modular es un enfoque en ingeniería de software que consiste en dividir un sistema complejo en módulos o componentes más pequeños y manejables. Cada módulo tiene una función específica y puede interactuar con otros módulos a través de interfaces bien definidas.
- ★ *Descomposición orientada por datos:* basada en las estructuras externas de los datos. La descomposición orientada por datos es un enfoque de diseño y organización de sistemas que se centra en la separación de los datos y las operaciones que se realizan sobre esos datos en diferentes módulos o componentes. Este enfoque busca lograr una estructura más clara y mantenible al separar las funciones de procesamiento de datos y las estructuras de datos en entidades independientes.
- ★ *Descomposición orientada por eventos:* basado en los eventos a manejar en el sistema. Cómo los eventos cambian el estado del sistema. La descomposición orientada por eventos es un enfoque de diseño y arquitectura de sistemas que se centra en la interacción y comunicación basada en eventos entre diferentes componentes o módulos del sistema. En este enfoque, los módulos están diseñados para responder y actuar en función de los eventos que ocurren en el sistema.
- ★ *Diseño "de afuera hacia adentro"*: enfoque de caja negra. Basado en las entradas del usuario.

★ *Diseño orientado a objetos:* identificar clases de objetos y sus interrelaciones. A nivel alto: descripción de objetos. A niveles bajos: atributos de los objetos, acciones. El diseño orientado a objetos (DOO) es un enfoque fundamental en ingeniería de software que se centra en la creación de sistemas basados en objetos. Un objeto es una entidad que encapsula datos (atributos) y comportamientos (métodos o funciones) relacionados, permitiendo una representación modular y organizada de la funcionalidad del sistema.

Encapsulación: Agrupa los datos y las operaciones relevantes dentro de un objeto, lo que proporciona control sobre el acceso y la modificación de los datos. Los detalles internos de un objeto están ocultos al exterior, y solo se accede a través de interfaces bien definidas.



Conceptos de diseño

❖ *Abstracción:* Es el proceso de identificar y enfocarse en las características clave de una entidad, eliminando o ignorando los aspectos que no son esenciales para el contexto en cuestión.

Abstracciones de:

➔ Una *abstracción de procedimiento* es una secuencia de instrucciones que tienen una función específica y limitada. El nombre de la abstracción de procedimiento implica estas funciones, pero se omiten detalles específicos. Un ejemplo de esto sería la palabra abrir, en el caso de una puerta. Abrir

implica una secuencia larga de pasos del procedimiento (caminar hacia la puerta, llegar y tomar el picaporte, girar éste y jalar la puerta, retroceder para que la puerta se abra, etcétera).

- ➔ Una *abstracción de datos* es un conjunto de éstos con nombre que describe a un objeto de datos. En el contexto de la abstracción de procedimiento abrir, puede definirse una abstracción de datos llamada puerta. Como cualquier objeto de datos, la abstracción de datos para puerta agruparía un conjunto de atributos que describirían la puerta (tipo, dirección del abatimiento, mecanismo de apertura, peso, dimensiones, etc.).

❖ *Arquitectura:* La arquitectura es una etapa más amplia y de mayor nivel en la que se define la estructura global y la organización del sistema. Se centra en aspectos de alto nivel como los componentes principales del sistema, sus relaciones, los patrones arquitectónicos y cómo se comunican. A partir de ella se realizará el diseño más detallado. Se hace uso de patrones.

El diseño arquitectónico se representa con el uso de uno o más de varios modelos diferentes:

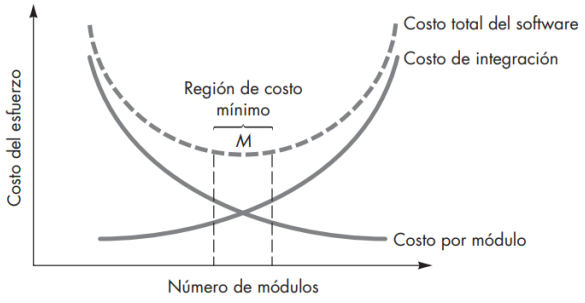
- ➢ Los *modelos estructurales* representan la arquitectura como un conjunto organizado de componentes del programa.
- ➢ Los *modelos de marco* aumentan el nivel de abstracción del diseño, al tratar de identificar patrones de diseño arquitectónico repetibles que se encuentran en tipos similares de aplicaciones.
- ➢ Los *modelos dinámicos* abordan los aspectos estructurales de la arquitectura del programa e indican cómo cambia la estructura o la configuración del sistema en función de eventos externos.

- ➢ Los *modelos del proceso* se centran en el diseño del negocio o proceso técnico al que debe dar acomodo el sistema.
- ➢ Los *modelos funcionales* se usan para representar la jerarquía funcional de un sistema.

❖ *Patrones:* Un patrón de diseño describe una estructura de diseño que resuelve un problema particular del diseño dentro de un contexto específico y entre "fuerzas" que afectan la manera en la que se aplica y en la que se utiliza dicho patrón.

❖ *Separación de intereses:* La división de problemas es un concepto de diseño que sugiere que cualquier problema complejo puede manejarse con más facilidad si se subdivide en elementos susceptibles de resolverse u optimizarse de manera independiente.Tiene efecto directo en el diseño y en muchos otros aspectos. La complejidad disminuye si los problemas son más simples. La división de problemas se manifiesta en otros conceptos de diseño relacionados: modularidad, aspectos, independencia de funcionamiento y mejora.

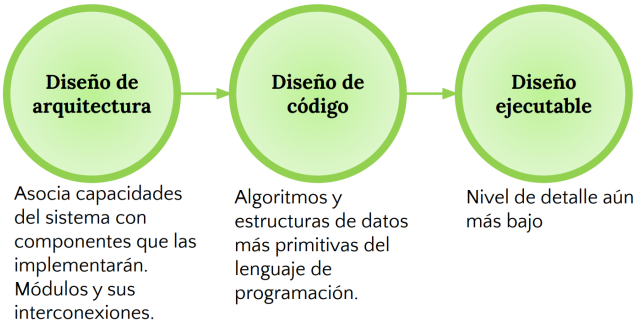
❖ *Modularidad:* La modularidad es la manifestación más común de la división de problemas. El software se divide en componentes con nombres distintos y abordables por separado, en ocasiones llamados módulos, que se integran para satisfacer los requerimientos del problema. En función de las circunstancias, el diseño debe descomponerse en muchos módulos con la esperanza de que sea más fácil entenderlos y, en consecuencia, reducir el costo requerido para elaborar el software. Un sistema es modular cuando una de las actividades la realiza un único componente y que además tiene bien definidas todas sus entradas y salidas.



- *Ocultamiento de la información:* El principio del ocultamiento de información sugiere que los módulos se "caracterizan por decisiones de diseño que se oculten (cada una) de las demás". En otras palabras, deben especificarse y diseñarse módulos, de forma que la información (algoritmos y datos) contenida en un módulo sea inaccesible para los que no necesiten de ella.El ocultamiento implica que la modularidad efectiva se logra definiendo un conjunto de módulos independientes que intercambien sólo aquella información necesaria para lograr la función del software. La abstracción ayuda a definir las entidades de procedimiento (o informativas) que constituyen el software. El ocultamiento define y hace cumplir las restricciones de acceso tanto a los detalles de procedimiento como a cualquier estructura de datos local que utilice el módulo. El uso del ocultamiento de información como criterio de diseño para los sistemas modulares proporciona los máximos beneficios cuando se requiere hacer modificaciones durante las pruebas, y más adelante, al dar mantenimiento al software.
- *Independencia funcional:* El concepto de independencia funcional es resultado directo de la separación de problemas y de los conceptos de abstracción y ocultamiento de información. La independencia funcional se logra desarrollando módulos de manera que cada módulo resuelva un subconjunto específico de requerimientos y tenga una interfaz sencilla cuando se vea desde otras partes de la estructura del programa.
- *Refinamiento:* En realidad, el refinamiento es un proceso de elaboración. Se comienza con un enunciado de la función (o descripción de la información), definida en un nivel de abstracción alto. Es decir, el enunciado describe la función o información de manera conceptual, pero no dice nada sobre los trabajos internos de la función o de la estructura interna de la información. Después se trabaja sobre el enunciado original, dando más y más detalles conforme tiene lugar el refinamiento (elaboración) sucesivo. La abstracción y el refinamiento son conceptos complementarios. La primera permite especificar internamente el procedimiento y los datos, pero elimina la necesidad de que los "extraños" conozcan los detalles de bajo nivel. El refinamiento ayuda a revelar estos detalles a medida que avanza el diseño. Es el proceso por medio del cual se disminuye el nivel de abstracción.

Diseño arquitectónico

Niveles de diseño



Estilos de arquitectura

- ★ *Centradas en los datos (Repositorios):* Almacenamiento central de datos.Conjunto de componentes que operan sobre el almacenamiento: almacenar, recuperar, actualizar .Ventaja: disponibilidad de los datos .Desventaja: el formato debe ser aceptable para todos los clientes que lo utiliza.
- ★ *De flujos de datos (Tuberías y filtros):* Esta arquitectura se aplica cuando datos de entrada van a transformarse en datos de salida a través de una serie de componentes computacionales o manipuladores. Los filtros son independientes entre sí.Comprender los efectos del sistema sobre la entrada y la salida y la composición de las transformaciones.Se reutilizan fácilmente los filtros en otros sistemas. La evolución es más simple: incorporar o quitar filtros. Facilita la simulación del sistema para analizar propiedades. Ejecución concurrente. Desventajas:Alienta el procesamiento por lotes, no es apto para aplicaciones interactivas, Mantener la correspondencia entre dos corrientes de datos relacionadas, Al ser independientes los filtros pueden duplicar la funcionalidad que efectúan otros.
- ★ *De llamada y retorno:* Este estilo arquitectónico permite obtener una estructura de programa que es relativamente fácil de modificar y escalar. Clásica estructura jerárquica. Un programa principal. Varios subprogramas componentes
- ★ *Diseño orientado a objetos:* Es un enfoque de desarrollo de software que organiza tanto el problema como su solución como una colección de objetos. En la representación incluye tanto los datos (y su estructura) como el comportamiento. Esto hace que en

muchos casos la descripción del problema, y el diseño de la solución sea muy similar o casi idéntica. La representación puede ocultarse a los restantes objetos del sistema (encapsulación).

★ *Capas:* Los modelos estratificados tienen organización jerárquica. Niveles superiores tienen mayor abstracción. Ventaja: agregar o modificar capas es sencillo ya que el cambio solo afecta a las capas adyacentes. Desventajas: No siempre es fácil estructurar sistemas de esta manera, Se agrega el costo de la coordinación de las capas

Diseño de interfaz

Dejar el control al usuario	*Reducir la carga de memoria del usuario*	*Hacer que la interfaz sea consistente*
➢ No obligar a realizar acciones innecesarias o no deseadas. ➢ Dar interacción flexible ➢ Interacción interrumpible y reversible ➢ Permitir personalización: facilitar aumento de habilidad ➢ Ocultar tecnicismos internos ➢ Permitir manipulación directa de objetos en pantalla	➢ Reducir demanda de memoria a corto plazo ➢ Que lo preestablecido sea significativo ➢ Atajos intuitivos ➢ Metáforas en la distribución ➢ Revelar información de manera progresiva	➢ Colocar la tarea en contexto significativo ➢ Consistencia en toda la familia de aplicaciones ➢ No hacer cambios sobre las expectativas del usuario

Buenas prácticas

Características de un buen diseño

➔ Se intenta que cada componente del diseño sea lo más independiente del resto posible: Mejora la comprensión de cada componente individual,

Es más sencillo de modificar una característica de un componente si no afecta a los demás, Es más fácil identificar fallas.

➔ Para medir la independencia de los componentes usamos dos conceptos: Acoplamiento y Cohesión.

Acoplamiento:

Dos componentes están altamente acoplados cuando existe mucha dependencia entre ellos. Los componentes poco acoplados tienen pocas dependencias y/o las interconexiones son débiles. El acoplamiento depende de: Las referencias hechas de un componente a otro, La cantidad de datos pasados entre componentes, El grado de control de un componente sobre otro, La complejidad de la interfaz entre los componentes.

Tipos de acoplamiento:

★ *Acoplamiento de contenido:* Cuando un componente A modifica a un componente B, este es completamente dependiente del que lo modifica.

★ *Acoplamiento común:* Cuando un almacenamiento de datos es común a dos o más módulos.

★ *Acoplamiento de control:* Cuando un componente pasa parámetros de control a otro.

★ *Acoplamiento de molde:* Cuando se utiliza una estructura de datos compleja para pasar información de un módulo a otro.

★ *Acoplamiento de datos:* Si solo comparten datos.

Cohesión

La cohesión se refiere al grado de adhesión (unión) interna que tiene el componente. A mayor grado de cohesión, más relacionadas entre sí están sus partes internas. Un componente es cohesivo si todos sus elementos están orientados a realizar una única tarea y son esenciales para llevarla a cabo.

Tipos de cohesión:

★ *Cohesión coincidental:* Ocurre cuando las partes no tienen relación alguna entre sí.

★ *Cohesión lógica: L*os elementos están relacionados lógicamente. Ej: todas las funciones sirven para la entrada de datos, sin importar el origen.

- ★ *Cohesión temporal*: Cuando las funciones de un componente están relacionadas por el momento en el que ocurren o son invocadas.
- ★ *Cohesión de procedimiento:* En ocasiones las funciones se agrupan únicamente porque están relacionadas por un procedimiento, hay un orden de ejecución.
- ★ *Cohesión comunicativa:* Cuando un componente asocia funciones que trabajan (operan, producen) en el mismo conjunto de datos.
- ★ *Cohesión secuencial:* Si la relación entre las funciones es que una produce la salida que sirve de entrada a la siguiente.
- ★ *Cohesión funcional:* En estos componentes las partes están relacionadas por la función que realizan: es una única función y todas las partes son esenciales para realizarla. Es la cohesión ideal.

Un buen diseño modular minimiza el acoplamiento y maximiza la cohesión.

Diseño de situaciones no deseadas

- ❖ La especificación de requerimientos dice lo que debe hacer el sistema. No explicita lo que se supone que no va a ocurrir.
- ❖ Identificar lo que conocemos como *excepciones*: situaciones que se sabe que no son las esperadas.
- ❖ El diseño puede incluir *manejo de excepciones* de modo que el sistema pueda reaccionar de manera satisfactoria ante un evento de este tipo.
- ❖ Si el defecto es en el diseño se puede propagar a los demás productos del proceso.
- ❖ *Falla*: desvío del sistema de su comportamiento requerido. Es percibida por el usuario.
- ❖ El *defecto* origina la falla. El defecto solo lo puede percibir el desarrollador.
- ❖ No todo defecto deriva en falla: las condiciones pueden no darse nunca.

Técnicas para mejorar el diseño

- ➢ Reducción de la complejidad: Reducir la complejidad desde que se detecta en los modelos.
- ➢ Diseño por contrato:
    - Considera el sistema como un conjunto de componentes que se comunican.
    - Las comunicaciones están especificadas en contratos que deben definir exactamente lo que hace cada uno y cómo interactúan.

- No pueden garantizar exactitud pero dan base para pruebas y validaciones

Implementación

Codificación del software

- ➢ Múltiples lenguajes de programación
- ➢ Varias personas involucradas en el proceso
- ➢ Grupos de trabajo: cooperación y coordinación

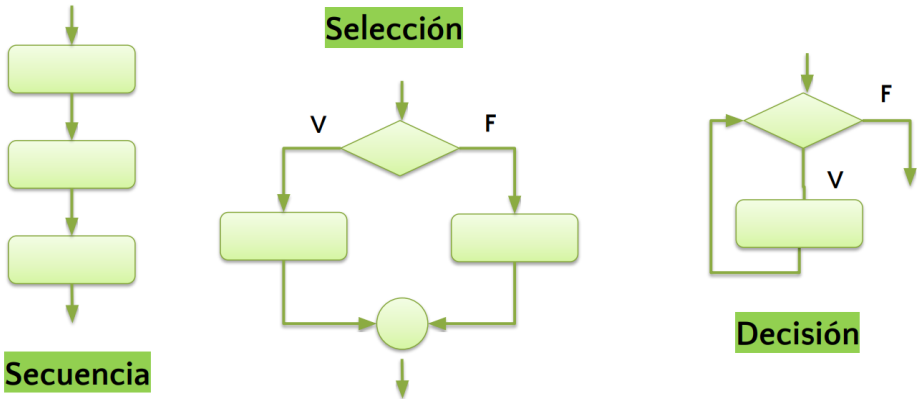
Se definen estándares de programación: Estilo, formato: código y documentación.

Pautas para la programación

A partir del diseño del componente el programador tiene libertad para aplicar su creatividad en la implementación.

Cada componente de un programa implica al menos tres aspectos principales:

1) Estructuras de control



Determinadas por la estructura misma del código.Debe poder leerse fácilmente un componente de lo general a lo particular.

Modularidad: también aplicarla al código

- ★ Facilita el mantenimiento y la reutilización
- ★ Generalizar todo lo posible sin afectar la performance o la facilidad de comprensión.

2) Algoritmos

Se debe tener en cuenta:

- ❖ *Tiempo* de ejecución
- ❖ Costo
  - ➢ de *escritura*
  - ➢ de *prueba:* en tiempo y en dificultad de construcción de los casos de prueba
  - ➢ de *modificación* cuando sea necesario

No sacrificar claridad y corrección por velocidad de ejecución

3) Estructuras de datos

   Buscar:

   - ★ Que el almacenamiento y la manipulación de los datos sea directa.
   - ★ Que ayude a simplificar el programa.

La estructura de los datos puede determinar la estructura del programa

Documentación del código

Explica qué hace el programa y cómo lo hace.

- ➔ Documentación *interna*: material descriptivo escrito directamente dentro del código
- ➔ Documentación *externa*: toda otra documentación que acompaña al programa

Documentación interna

Identifica el programa.Describe algoritmos, estructuras de datos y flujo de control.Usualmente ubicada al comienzo de cada componente. Incluye:

- ➢ Denominación del componente
- ➢ Quién lo ha escrito
- ➢ Dónde va ubicado en el diseño general del sistema
- ➢ Cuándo fue escrito/revisado/modificado
- ➢ Justificación del componente
- ➢ Uso que hace de las estructuras

Documentación interna y externa

- ➔ Recomendable en la documentación interna
  - ◆ Comentarios que acompañen el código

- ◆ Nombres de variables significativas
- ◆ Dar formato que ayude a la comprensión

➔ Incluir en la documentación externa

- ◆ Descripción del problema
- ◆ Descripción de los algoritmos elegidos
- ◆ Descripción de los datos

Pruebas

Se concentra en la búsqueda de defectos en los programas.

¿Por qué falla el software?

- ★ Especificación errónea o requerimientos omitidos
- ★ Requerimientos imposibles de implementar dado el hw y sw con los que se cuenta
- ★ El diseño del sistema es defectuoso
- ★ El diseño del programa es defectuoso
- ★ El código tiene errores

Objetivo de la prueba

Se prueba para demostrar la existencia de defectos. La prueba es destructiva, se considera exitosa solamente si encuentra defectos o si se producen fallas como consecuencia de los procedimientos de prueba. La visión del programador es opuesta: busca demostrar que su código es correcto. Esto va en contra del espíritu de las pruebas.

Tipos de defectos:

- ❖ *Algorítmicos*: el algoritmo o la lógica de un componente no producen la salida apropiada para la entrada dada, por errores en los pasos del procesamiento.
  - ➢ Bifurcar antes o después de tiempo
  - ➢ Probar una condición errónea
  - ➢ Olvidar inicializar variables o constantes de un bucle
  - ➢ Olvidar probar una condición particular
  - ➢ Comparar variables de tipos de datos inapropiados
- ❖ *De sintaxis*: en las estructuras del lenguaje de programación.
- ❖ *De computación y de precisión:* implementación erróneas de fórmulas o cálculos sin la exactitud necesaria.

- *De documentación*: cuando esta no se corresponde con lo que el programa verdaderamente hace
- *Por estrés o sobrecarga*: se sobrepasan las estructuras
- *De capacidad o límites*: se deteriora el desempeño del sistema por superar sus límites especificados
- *De sincronización o coordinación*: coordinación inadecuada de procesos que deben ejecutar de forma simultánea
- *De rendimiento o desempeño*: el sistema no opera con el desempeño (velocidad, seguridad, exactitud, confiabilidad) esperado
- *De recuperación*: la respuesta a una falla no es la esperada

Visiones de los objetos de prueba

- *Caja cerrada o negra:* La prueba de caja negra se refiere a las pruebas que se llevan a cabo en la interfaz del software. Una prueba de caja negra examina algunos aspectos fundamentales de un sistema con poca preocupación por la estructura lógica interna del software.
  - El objeto se analiza desconociendo el contenido
  - Las pruebas consisten en alimentar la "caja negra" con entradas y observar cuáles son las salidas
- *Caja abierta o blanca:* La prueba de caja blanca del software se basa en el examen cercano de los detalles de procedimiento. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles.
  - *Con la visión de caja cerrada no se puede elegir casos de prueba representativos*
  - *Se utiliza la estructura del objeto para probar de diversas maneras*
  - *Por ejemplo: todos los caminos posibles del código*

Organización de las pruebas

1. De módulo, de componente, o de unidad
   - Verificar funcionamiento del componente con los tipos de entrada esperados, estructuras de los datos, lógica y condiciones de límites para los datos
2. De integración
   - Verificar el funcionamiento conjunto de los componentes del sistema

3. Funcional
   ➔ Determinar si las funciones descriptas en la especificación son realmente ejecutadas por el sistema integrado
4. De rendimiento o desempeño
   ➔ Comparar el sistema con los requerimientos de hardware y software.
5. De aceptación
   ➔ En conjunto con el cliente se prueba el sistema contra las especificaciones de requerimientos
6. De instalación
   ➔ Se prueba una vez más, para garantizar que una vez instalado funcione como debe hacerlo

Casos de prueba

Casos de prueba: conjunto particular de casos de entrada a ser utilizado en la prueba de un programa. La selección de los casos de prueba se hace para utilizar al menos un enfoque:

★ Comprobación de sentencias
★ Pruebas de la ramificación
★ Comprobación del camino

Minuciosidad de la prueba

➢ *Comprobación de sentencias*: ejecutar al menos una vez cada sentencia del componente
➢ *Pruebas de la ramificación*: cada rama de cada decisión en el código se elige al menos una vez
➢ *Comprobación del camino:* cada camino distinto formado en el código se ejecuta por lo menos una vez

Planificación de la prueba

Pasos

❖ Determinación de los objetivos de la prueba
❖ Diseño de los casos de prueba
❖ Preparación escrita de los casos de prueba
❖ Verificación de los casos de prueba
❖ Ejecución de los casos de prueba
❖ Evaluación de los resultados

Equipo de prueba

Los programadores están demasiado familiarizados con la estructura e intención de la implementación y pueden tener dificultades reconociendo errores y diferencias con los requerimientos. Idealmente el equipo de pruebas es independiente del equipo de implementación.

Las personas encargadas de las pruebas son profesionales con experiencia y se involucran en esta tarea desde el comienzo del proyecto.

- ➢ Organizar las pruebas
- ➢ Diseñar las pruebas
- ➢ Ejecutar las pruebas