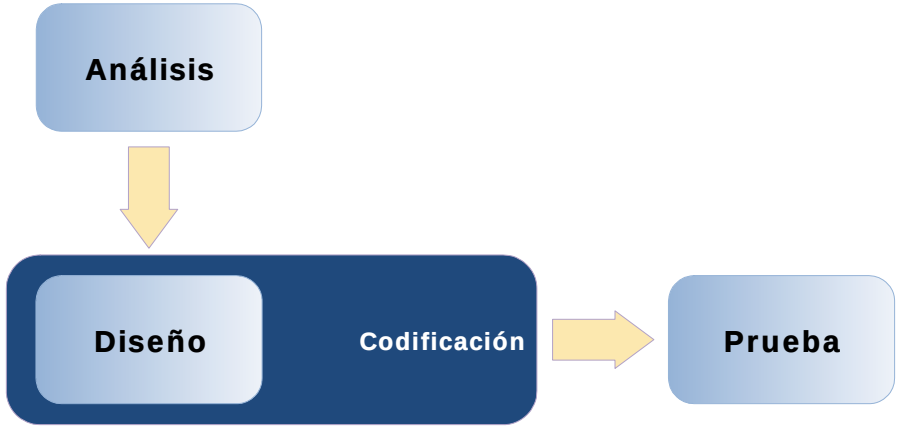
Universidad Nacional de la Patagonia San Juan Bosco
Facultad de Ingeniería

Cátedra: **Análisis y Diseño de Sistemas**

Unidad 4: El Proceso de Diseño

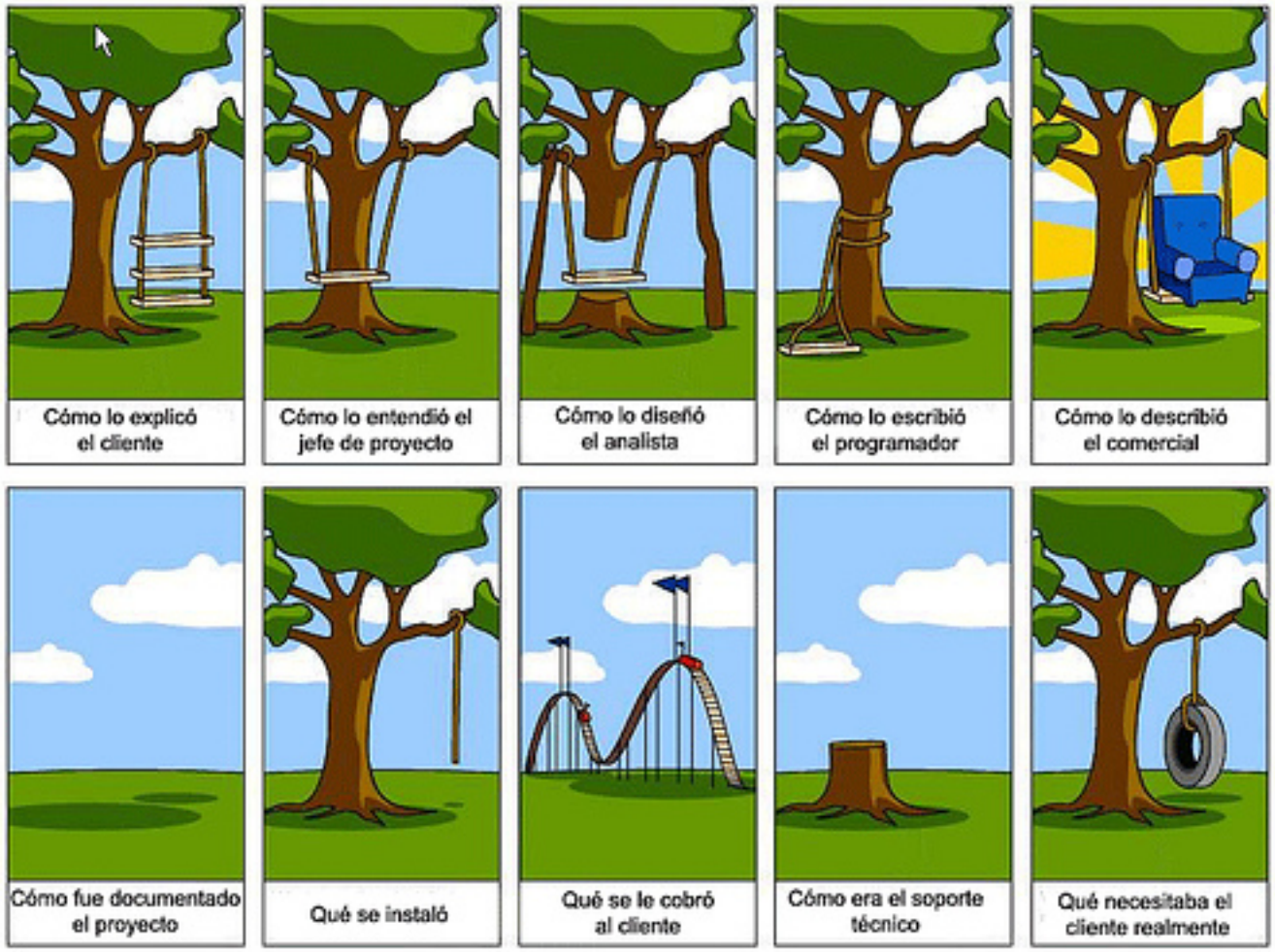
- Diseño e Ingeniería de Software
  - En el proceso de desarrollo de software las necesidades del usuario son traducidas en requisitos de SW, éstos transformados en diseño y el diseño implementado en código.
  - El **diseño es el primer paso en la fase de desarrollo** de cualquier producto o sistema de ingeniería.
  - El **proceso de desarrollo** consta de las siguientes actividades necesarias para desarrollar y verificar el software:
    - ✓ Diseño
    - ✓ Codificación
    - ✓ Prueba



  - Cada actividad transforma la información de manera que se obtenga finalmente un software válido.

- Diseño e Ingeniería de Software (Cont.)
  - Se podría definir el diseño como:
    - El **proceso de aplicar distintas técnicas y principios con el propósito de definir** un dispositivo, un proceso o un sistema con suficiente detalle como para permitir su realización.
  - El **objetivo del diseñador** es producir un modelo o representación de una entidad que se va a construir posteriormente.
  - El proceso combina:
    - ✓ Intuición y criterios basados en experiencia
    - ✓ Conjuntos de principios
    - ✓ Guías y/o heurísticas
    - ✓ Conjuntos de criterios que permiten juzgar la calidad

- Diseño e Ingeniería de Software (Cont.)
  - El **diseño** es el proceso sobre el que se asienta la **calidad del software**.
  - ↳ Por **calidad** se entiende la **adecuación del software a los requisitos exigidos.**



- El Proceso de Diseño
  - El **Diseño** es un proceso de resolución de problemas cuyo **objetivo** es encontrar y describir una forma para implementar los requisitos funcionales del sistema, respetando las restricciones impuestas por los requisitos no funcionales, ajustándose a los principios generales de calidad.
  - El **diseño del software es un proceso iterativo** mediante el cual los requisitos se traducen en un **plano** para desarrollar el software.
  - El diseño se representa a un alto nivel de abstracción. A medida que se itera se llega a una representación del diseño de mucho menor nivel de abstracción.
  - La calidad se evalúa con una serie de **Revisiones Técnicas Formales**, entre otras técnicas.
  - Las características para evaluar un buen diseño son:
    - ✓ Debe implementar todos los **requisitos explícitos** contenidos en el modelo de análisis y debe acomodar todos los **requisitos que desea el cliente**.
    - ✓ Debe ser una **guía** que puedan leer y entender los que construyan el código y los que prueban y mantienen el software.
    - ✓ Debería proporcionar una **idea completa** de lo que es el software.

- El Proceso de Diseño (Cont.)
  - Con el fin de evaluar la calidad de una representación de diseño, deberán establecerse los siguientes **criterios técnicos** para un buen diseño:

1. Un diseño debería presentar una **organización jerárquica** que haga un uso inteligente del control entre los componentes del software.
2. El diseño debería ser **modular**, es decir, se debería hacer una partición lógica del software en elementos que realicen funciones y subfunciones específicas.
3. Debería contener **abstracciones** de datos y procedimentales.
4. Debería producir módulos que presenten **características funcionales independientes**.
5. Debería conducir a **interfaces** que reduzcan la complejidad de las conexiones entre los módulos y el entorno exterior.
6. Se debería producir un diseño usando un **método** que pudiera repetirse según la información obtenida durante el análisis de requerimientos.

- El Proceso de Diseño – Evolución
  - La evolución del diseño del software es un proceso continuo que ha abarcado las últimas cuatro décadas.
  - El primer trabajo de diseño se concentraba en criterios para el desarrollo de **programas modulares** y métodos para refinar las estructuras del software de manera descendente. Los aspectos procedimentales de la definición de diseño evolucionaron en una **filosofía denominada Estructurada**.
  - Un trabajo posterior propuso métodos para la **conversión del flujo de datos** o estructura de datos en una definición de diseño.
  - Enfoques de diseño más recientes hacia la derivación de diseño proponen un **método Orientado a Objetos**.
  - Hoy en día, se ha hecho hincapié en un **diseño de software basado** en la **Arquitectura del software**.

  - Independientemente del modelo de diseño que se utilice, un ingeniero del software deberá aplicar un conjunto de principios fundamentales y conceptos básicos para el **diseño a nivel de componentes (procedimental), de interfaz, arquitectónico y de datos**.

- Principios de Diseño
  - El diseño es un proceso y un modelo a la vez.
  - El **Proceso de Diseño** es un conjunto de pasos repetitivos que permiten al diseñador describir todos los aspectos del SW a construir.
  - El **Modelo de Diseño** proporciona distintas visiones del programa.
  - Principios básicos para el diseño de software:
    - ✓ En el proceso de diseño no debería ponerse orejeras. Un buen diseñador debería considerar enfoques alternativos, juzgando cada uno en base a los requisitos del problema, los recursos disponibles para hacer el trabajo y los conceptos de diseño.
    - ✓ Se debería poder seguir los pasos del diseño hasta el modelo de análisis. Como un solo elemento del modelo de diseño se refiere a menudo a múltiples requisitos, es necesario tener los medios para hacer un seguimiento de cómo ha satisfecho los requisitos del modelo de diseño.
    - ✓ El diseño no debería inventar nada que esté inventado. Los sistemas se construyen usando un conjunto de estructuras de diseño, muchas de las cuales ya se han utilizado anteriormente. Estas estructuras (componentes de diseño reutilizables) deben considerarse siempre antes que reinventar nada. ¡El tiempo es corto y los recursos limitados! El tiempo invertido en diseño debe concentrarse en representar ideas verdaderamente nuevas y en integrar aquellas estructuras que ya existen.

- Principios de Diseño (Cont.)
  - ✓ El diseño debería minimizar la distancia intelectual entre el software y el problema tal y como existe en el mundo real. Es decir, la estructura del diseño del software debería -cuando sea posible- imitar la estructura del dominio del problema.
  - ✓ El diseño debería presentar uniformidad e integración. Un diseño es uniforme si parece que sólo una persona desarrolló todo el conjunto. Se deberían definir normas de estilo y de formato para el equipo antes de comenzar el trabajo de diseño.
  - ✓ El diseño debería estructurarse para admitir cambios. Muchos de los conceptos de diseño permiten conseguir este principio.
  - ✓ El diseño debería estructurarse para degradarse poco a poco, incluso cuando se enfrenta a datos, sucesos o condiciones operativas aberrantes. Un programa bien diseñado no debería explotar nunca. Debería diseñarse para aceptar circunstancias inusuales, y si debe terminar el procesamiento, hacerlo de una manera suave.

- Principios de Diseño (Cont.)
  - ✓ El diseño no es escribir código y escribir código no es diseñar. Incluso cuando se crean diseños procedimentales detallados para los componentes de un programa, el nivel de abstracción del modelo de diseño es mayor que el del código fuente. Las únicas decisiones de diseño hechas a nivel de código se refieren a los pequeños detalles de implementación que permiten codificar el diseño procedimental.
  - ✓ Se debería valorar la calidad del diseño mientras se crea, no después de terminarlo. Existe una variedad de conceptos de diseño y medidas de diseño disponibles para ayudar al diseñador en la valoración de la calidad.
  - ✓ Se debería revisar el diseño para minimizar los errores conceptuales (semánticos). A veces hay tendencia a concentrarse en minucias cuando se revisa el diseño, impidiendo los árboles ver el bosque. El diseñador debe asegurarse de que se revisan los elementos conceptuales principales del diseño (omisiones, ambigüedades, inconsistencias) antes de preocuparse de la sintaxis del modelo de diseño.

- Principios de Diseño (Cont.)
  - Cuando los principios de diseño descritos anteriormente se aplican adecuadamente, el ingeniero del software crea un diseño que muestra los factores de calidad tanto internos como externos:

    - ✓ Los **factores de calidad internos** tienen importancia para los ingenieros del software. Desde una perspectiva técnica conducen a un diseño de calidad alta. Para lograr los factores de calidad internos, el diseñador deberá comprender los conceptos de diseño básicos.
    - ✓ Los **factores de calidad externos** son esas propiedades del software que pueden ser observadas fácilmente por los usuarios (por ejemplo, velocidad, fiabilidad, grado de corrección, usabilidad)

- Conceptos de Diseño
  - Durante las últimas cuatro décadas se ha experimentado la evolución de un **conjunto de conceptos fundamentales de diseño de software**.
  - Aunque el grado de interés en cada concepto ha variado con los años, todos han experimentado el paso del tiempo.
  - Cada uno de ellos **proporcionarán la base** de donde el diseñador podrá aplicar los métodos de diseño más sofisticados.
  - Cada uno ayudará al ingeniero del software a responder las preguntas siguientes:
    - ✓ ¿Qué criterios se podrán utilizar para la partición del software en componentes individuales?
    - ✓ ¿Cómo se puede separar la función y la estructura de datos de una representación conceptual del software?
    - ✓ ¿Existen criterios uniformes que definen la calidad técnica de un diseño de software?

  ↳ El comienzo de la sabiduría para un ingeniero del software es reconocer la diferencia entre hacer que un programa funcione y conseguir que lo haga correctamente.

- Conceptos de Diseño (Cont.)
  - Los conceptos aportan al diseñador del SW un fundamento desde el que se pueden aplicar métodos de diseño sofisticados:
    - ✓ Abstracción
    - ✓ Refinamiento
    - ✓ Modularidad
    - ✓ Arquitectura del SW
    - ✓ Jerarquía de control
    - ✓ Partición estructural
    - ✓ Estructura de datos
    - ✓ Procedimiento de SW
    - ✓ Ocultamiento de la información
- Abstracción
  - Cuando consideramos una solución modular para cualquier problema se pueden plantear muchos niveles de abstracción.
  - Al nivel superior de abstracción, se establece una solución en términos amplios usando el lenguaje del entorno del problema. A niveles más bajos, se toma una orientación más procedimental. Se conjuga la terminología orientada a la implementación en el esfuerzo para plantear una solución. Finalmente, al nivel inferior de abstracción, la solución se establece de manera que pueda implementarse directamente.

- Conceptos de Diseño – Abstracción (Cont.)
  - Cada fase del proceso de ingeniería del software es un refinamiento en el nivel de abstracción de la solución SW. Durante la **ingeniería de sistemas**, el SW es asociado como un elemento de un sistema basado en computadora. Durante el **análisis de requisitos**, la solución SW se establece en términos que sean familiares en el entorno del problema. A medida que nos movemos a través del **proceso de diseño**, se reduce el nivel de abstracción. Finalmente, se alcanza el nivel inferior de abstracción cuando se **escribe el código**.
  - Una **abstracción procedimental** es una secuencia dada de instrucciones que tiene una función específica y limitada.
  - Una **abstracción de datos** es una colección determinada de datos que describen un objeto de datos.
  - La **abstracción de control** implica un mecanismo de control del programa sin especificar detalles internos. P.e. sincronización de un semáforo para coordinar actividades de un sistema operativo.

  - **Ventajas**:
    - ✓ Define y refuerza las restricciones de acceso
    - ✓ Facilita el mantenimiento y la evolución de los sistemas SW
    - ✓ Limita el impacto global de las decisiones de diseño locales
    - ✓ Favorece la encapsulación

- Conceptos de Diseño – Refinamiento
  - El refinamiento paso a paso es una **estrategia de diseño descendente**.
  - La arquitectura de un programa se desarrolla **refinando** sucesivamente **niveles de detalle procedimental**. Se desarrolla una jerarquía descomponiendo un enunciado macroscópico de función (abstracción procedimental) al estilo paso a paso hasta que se llega a los enunciados del lenguaje de programación.
  - El proceso de refinamiento del programa es análogo al proceso de refinamiento y partición que se emplea durante el análisis. La diferencia está en el nivel de detalle de implementación considerado.
  - El refinamiento es, de hecho, un proceso de elaboración. Empezamos con un enunciado de función (o descripción de información) definida a un alto nivel de abstracción. En el refinamiento el diseñador va elaborando el enunciado original, proporcionando más detalles con cada refinamiento (elaboración) sucesivo.
  - La abstracción y el refinamiento son conceptos complementarios. La abstracción permite a un diseñador especificar procedimientos y datos y aun así, suprimir detalles de bajo nivel. El refinamiento ayuda al diseñador a relevar detalles de bajo nivel a medida que progresa el diseño. Ambos conceptos ayudan al diseñador a crear un modelo de diseño completo a medida que evoluciona el diseño.

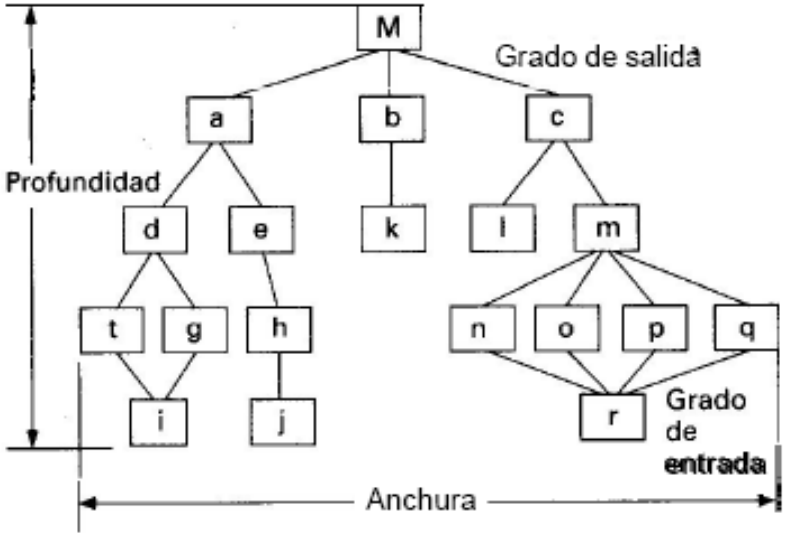
- Conceptos de Diseño – Modularidad
  - La arquitectura del software conlleva modularidad, es decir que divide el software en componentes identificables y tratables por separado, denominados módulos, que están integrados para satisfacer los requisitos del programa.
  - La modularidad es el atributo del software que permite a un programa ser **manejable intelectualmente**. Un software monolítico no puede ser entendido fácilmente por un lector.
  - Si subdividimos el software indefinidamente, el esfuerzo requerido para desarrollarlo es mínimo. Desgraciadamente, intervienen otras fuerzas, que hacen que esta conclusión sea falsa.
  - El esfuerzo (costo) para desarrollar un módulo de software individual disminuye a medida que crece el número total de módulos. A medida que crece el número de módulos, el esfuerzo (costo) asociados con la integración de módulos también crece.
  - Se debería evitar modularizar de más o de menos.

- Conceptos de Diseño – Modularidad (Cont.)
  - Criterios para evaluar un método de diseño con respecto a su capacidad de definir un sistema modular eficaz:
    - ✓ **Capacidad de descomposición modular:** Si un método de diseño proporciona un mecanismo sistemático de descomposición del problema en subproblemas, reducirá la complejidad del problema global, consiguiendo, por tanto, una solución modular eficaz.
    - ✓ **Capacidad de empleo de componentes modulares:** Si un método de diseño permite ensamblar componentes de diseño existentes (reutilizables) en un nuevo sistema, proporcionará una solución modular que no invente nada ya inventado.
    - ✓ **Capacidad de comprensión modular:** Si se puede entender un módulo como una unidad por sí sola (sin referencias a otros módulos) será más fácil de construir y de cambiar.
    - ✓ **Continuidad modular:** Si pequeños cambios en los requisitos del sistema provocan cambios en los módulos individuales, en vez de cambios generalizados en el sistema, se minimizará el impacto de los efectos secundarios de los cambios.
    - ✓ **Protección modular:** Si se da una condición aberrante dentro de un módulo y los efectos se restringen dentro de ese módulo, se minimizará el impacto de los efectos secundarios de los errores.

- Conceptos de Diseño – Arquitectura del SW
  - La arquitectura del software alude a la **estructura global del software** y las maneras en que esa estructura proporciona **integridad conceptual** a un sistema.
  - La arquitectura es la **estructura jerárquica de los componentes del programa** (módulos), la manera de interactuar de estos componentes, y la estructura de los datos, usados por estos componentes.
  - En un sentido más amplio, los componentes pueden generalizarse para representar elementos principales del sistema y sus interacciones.
  - Uno de los objetivos del diseño es crear una **versión arquitectónica** de un sistema. Esta versión sirve como estructura desde la que se pueden llevar a cabo actividades de diseño más detalladas. Un conjunto de estructuras arquitectónicas permite al ingeniero del software reutilizar conceptos a nivel de diseño.

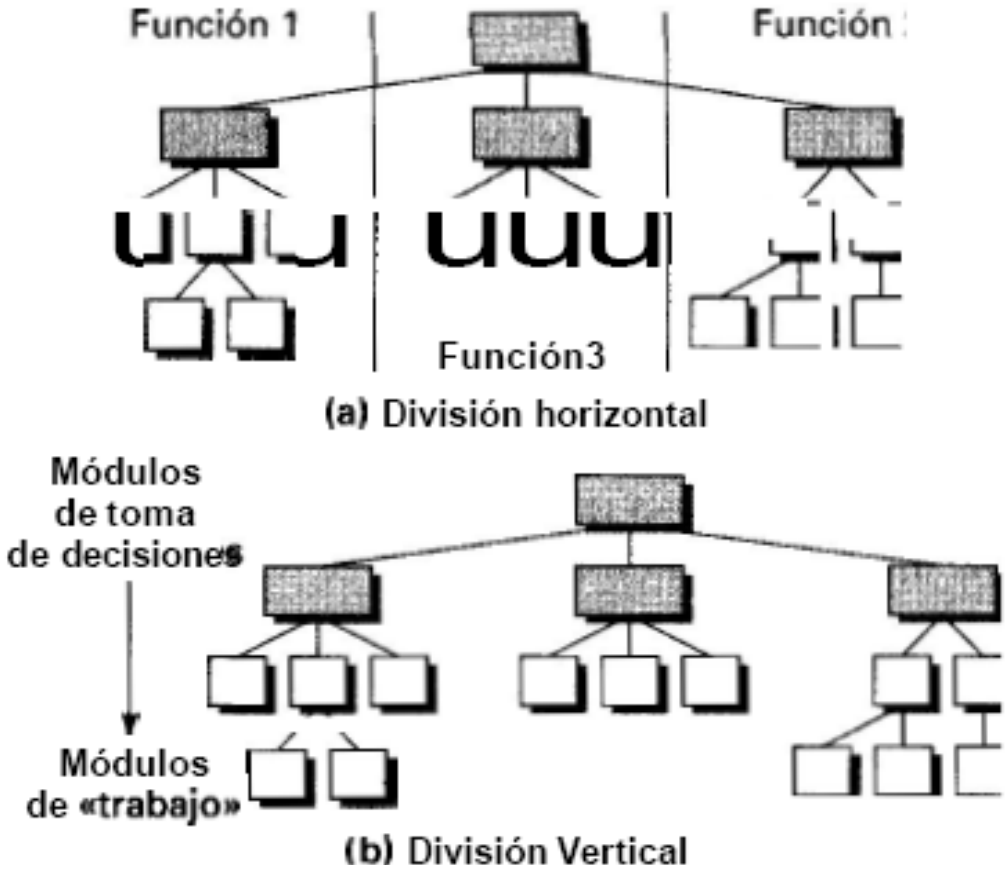
- Conceptos de Diseño – Arquitectura del SW (Cont.)
  - Las propiedades que deberían especificarse como parte de un diseño arquitectónico son:
    - ✓ **Propiedades estructurales:** Este aspecto de la representación del diseño arquitectónico define los componentes de un sistema y la manera en que se empaquetan estos componentes e interactúan los unos con los otros.
    - ✓ **Propiedades extra-funcionales:** La descripción del diseño arquitectónico debería ocupase de cómo consigue la arquitectura del diseño los requisitos de rendimiento, capacidad, fiabilidad, seguridad, adaptabilidad y otras características del sistema.
    - ✓ **Familias de sistemas relacionados:** El diseño arquitectónico debería dibujarse sobre patrones repetibles que se basen comúnmente en el diseño de familias de sistemas similares. El diseño debería tener la capacidad de volver a utilizar bloques de construcción arquitectónicos.

- Conceptos de Diseño – Jerarquía de Control
  - La jerarquía de control se denomina **estructura del programa**, representa la organización (a menudo jerárquica) de componentes del programa (módulos) e **implica una jerarquía de control**. No representa aspectos procedimentales del software tales como secuencias de procesos, ocurrencia/orden de decisiones o repeticiones de operaciones.
  - Para representar la jerarquía control se utiliza un conjunto de notaciones diferentes. El diagrama más común es el de forma de árbol que representa el control jerárquico para las arquitecturas de llamada y de retorno.
  - Con objeto de facilitar estudios posteriores de estructura, definiremos una serie de medidas y términos simples.



- Conceptos de Diseño – Jerarquía de Control (Cont.)
  - La **profundidad** y la **anchura** proporcionan una indicación de los distintos niveles de control y el ámbito global del control respectivamente.
  - El **grado de salida** es una medida del número de módulos que son controlados directamente por otro módulo.
  - El **grado de entrada** indica cuántos módulos controlan directamente un módulo dado.
  - La relación de control entre los módulos se expresa de la siguiente manera: un módulo que controla a otro se dice que es superior a él. Inversamente, un módulo controlado por otro se dice que es subordinado del controlador.
  - La jerarquía de control también representa dos características sutilmente diferentes de la arquitectura del software: **visibilidad** y **conectividad**.
    - ✓ La **visibilidad** indica el conjunto de componentes de programa que pueden invocarse o usarse sus datos, incluso cuando esto se realiza indirectamente.
    - ✓ La **conectividad** indica el conjunto de componentes que son invocados directamente o usados sus datos por un componente determinado.

- Conceptos de Diseño – Partición Estructural
  - La estructura del programa debería partirse tanto horizontalmente como verticalmente.

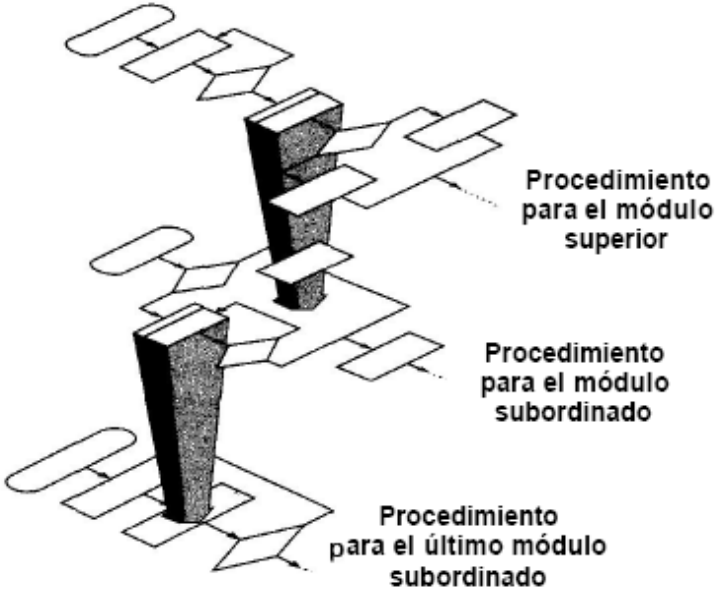


- Conceptos de Diseño – Partición Estructural (Cont.)
  - La **partición horizontal** define ramas separadas de la jerarquía modular para cada función principal del programa, los módulos de control se usan para coordinar la comunicación entre ellos y la ejecución de las funciones del programa. El enfoque más simple de la partición horizontal define tres particiones: entrada, transformación de datos y salida.
  - La partición horizontal de la arquitectura proporciona distintos beneficios:
    - ✓ Proporciona software más fácil de probar
    - ✓ Lleva a un software más fácil de mantener
    - ✓ Propaga menos efectos secundarios
    - ✓ Proporciona software más fácil de ampliar
  - Como las funciones principales se desacoplan las unas de las otras, los cambios tienden a ser menos complejos y las ampliaciones del sistema tienden a ser más fáciles de conseguir sin efectos secundarios.
  - En la parte negativa, la partición horizontal causa a menudo el paso de más datos a través de interfaces de módulos y puede complicar el control global del flujo del programa.

- Conceptos de Diseño – Partición Estructural (Cont.)
  - Si el estilo arquitectónico de un sistema es jerárquico, la estructura del programa se puede dividir tanto horizontal como verticalmente.
  - La **partición vertical** sugiere que el control (toma decisiones) y el trabajo se distribuyan descendentemente en la arquitectura del programa. Los módulos del nivel superior deberían realizar funciones de control y poco trabajo de procesamiento. Los módulos que residen en la parte baja de la arquitectura deberían ser los trabajadores, realizando todas las tareas de entrada, cálculo y salida.
  - La naturaleza de los cambios en las arquitecturas de programa justifica la necesidad de la partición vertical. Un cambio en el módulo de control (en la parte más alta de la arquitectura) tendrá mayor probabilidad de propagar efectos secundarios a los módulos subordinados a él. Un cambio en un módulo de trabajo, dado su bajo nivel en la estructura, es menos probable que cause la propagación de efectos secundarios. En general, los cambios en los programas de computadora giran alrededor de la entrada, cálculos o transformación y la salida. Es mucho menos probable que cambie la estructura global de control del programa.
  - Por esta razón las arquitecturas de partición vertical tienen **menos probabilidad de ser susceptibles a efectos secundarios** cuando se hacen cambios y tendrán por tanto **mejor capacidad de mantenimiento**, un factor clave para la calidad.

- Conceptos de Diseño – Estructura de datos
  - Es una representación de la **relación lógica entre los elementos individuales de datos**. La estructura de datos es tan importante como la estructura de programa en la representación de la arquitectura del software.
  - La estructura de datos dicta las alternativas de organización, métodos de acceso, capacidad de asociación y procesamiento de la información.
  - La organización y complejidad de estructura de datos están limitadas sólo por el ingenio del diseñador. Hay, sin embargo, unas pocas estructuras clásicas de datos que forman la base para construir estructuras más sofisticadas.
    - ✓ Un **elemento escalar** es la forma más simple de todas las estructuras de datos, representa un elemento simple que puede ser tratado como un identificador.
    - ✓ Cuando se organizan los elementos escalares como una lista o grupo contiguo, se forma un **vector secuencial**.
    - ✓ Cuando se amplia el vector, se crea una **matriz n-dimensional**.
    - ✓ Una **lista enlazada** es una estructura de datos que organiza elementos escalares, vectores o espacios no contiguos de manera que les permita ser procesados como una lista. Se pueden agregar nodos en cualquier punto de la lista redefiniendo punteros para acomodar la nueva entrada a la lista.

- Conceptos de Diseño – Procedimiento de software
  - La estructura del programa define la jerarquía de control sin tener en cuenta la secuencia de procesamiento y las decisiones. El procesamiento de software se **centra en los detalles de procesamiento de cada módulo individualmente**.
  - El procedimiento debe proporcionar una especificación exacta del procesamiento, incluyendo la secuencia de acontecimientos, puntos exactos de decisión, operaciones repetitivas, etc.
  - Hay una relación entre la estructura y el procedimiento. El procesamiento indicado para cada módulo debe incluir una referencia a todos los módulos subordinados al módulo que se describe. Es decir, una representación procedimental del software se distribuye en capas:



- Conceptos de Diseño – Ocultamiento de la información
  - Sugiere que los módulos se caractericen por decisiones de diseño que haga que cada uno se oculte de los demás.
  - Con otras palabras se debería especificar y diseñar los módulos para que **la información (procedimiento y datos) contenida dentro de un módulo sea inaccesible a otros módulos** que no necesiten esa información.
  - La ocultación implica que se puede conseguir una modularidad eficaz definiendo un conjunto de **módulos independientes que se comunican** entre ellos sólo con la **información necesaria** para conseguir la función del software.
  - La abstracción ayuda a definir las entidades procedimentales que componen el software.
  - Los beneficios son cuando se requieren modificaciones durante las pruebas y durante el mantenimiento.

- Diseño Modular efectivo
  - Un diseño modular efectivo:
    - ✓ Reduce la complejidad.
    - ✓ Facilita los cambios.
    - ✓ Fomenta el desarrollo paralelo de distintas partes del sistema.

- Independencia Funcional
  - La independencia funcional es producto directo de la modularidad, abstracción y ocultamiento de información.
  - La independencia funcional se consigue desarrollando módulos con una función única. Dicho de otra manera, queremos diseñar software de manera que **cada módulo trate una subfunción específica** de los requisitos **y tenga una interfaz sencilla** cuando se vea desde otras partes de la estructura del programa.
  - El software con modularidad efectiva es más fácil de desarrollar porque la función se puede compartimentar y las interfaces se simplifican.
  - **Los módulos independientes son más fáciles de mantener**.
  - La independencia funcional es la clave de un buen diseño y un buen diseño es la clave de la calidad.
  - La independencia se mide usando dos criterios cualitativos: **cohesión** y **acoplamiento**.

- Diseño Modular efectivo – Cohesión
  - Un módulo con cohesión realiza una sola tarea dentro de un procedimiento de software. Buscamos una **alta cohesión**.
  - La escala de cohesión no es lineal. Es decir, la parte baja de la cohesión es mucho **peor** que el rango medio, que es casi tan **bueno** como la parte alta de la escala. En la práctica, un diseñador no tiene que preocuparse de categorizar la cohesión en un módulo específico. Más bien, se deberá entender el concepto global, y así se deberán evitar los niveles bajos de cohesión al diseñar los códigos.

1. **Cohesión Coincidental:** Un módulo es coincidentalmente cohesivo cuando varios elementos contribuyen a actividades sin relación significativa y son de distinta categoría (tareas poco relacionadas).
2. **Cohesión Lógica:** Un módulo es lógicamente cohesivo cuando varios elementos contribuyen a actividades de la misma categoría en el cual la actividad o actividades a ejecutarse son seleccionadas desde afuera del módulo. Ejemplo: Emitir listados.
3. **Cohesión Temporal:** Un módulo es temporalmente cohesivo cuando varios elementos están involucrados en actividades que están relacionados por el tiempo. Ejemplo: Inicializar

- Diseño Modular efectivo – Cohesión (Cont.)

4. **Cohesión Procedimental:** Un módulo es procedimentalmente cohesivo cuando varios elementos están involucrados en actividades diferentes y no relacionadas en donde flujos de control van de una actividad a la próxima. Los elementos de un módulo están relacionados y deben ejecutarse en un orden.
5. **Cohesión Comunicacional:** Un módulo es comunicacionalmente cohesivo cuando varios elementos contribuyen a actividades que usan los mismos datos de entrada o salida, trabajan sobre la misma estructura de datos. No importa el orden de ejecución de las actividades.
6. **Cohesión Secuencial:** Un módulo es secuencialmente cohesivo cuando varios elementos están involucrados en actividades en el cual el dato de salida desde una actividad sirve como entrada de datos a la próxima actividad. Siempre se trabaja con la misma estructura de datos de entrada.
7. **Cohesión Funcional:** Un módulo es funcionalmente cohesivo cuando contiene elementos que contribuyen a la ejecución de una y solo una tarea relacionada. Devuelve un único resultado. Ejemplo: Calcular salario neto

- Diseño Modular efectivo – Cohesión (Cont.)
  - En la parte inferior (y no deseable) del espectro, encontraremos un módulo coincidentalmente cohesivo y en la más alta uno funcionalmente cohesivo .

  - Cómo determinar la Cohesión de un módulo:
    - ✓ Redactar la frase que describa el propósito o función del módulo.
    - ✓ Examinar la frase.

  - Si la frase es una sentencia compuesta con más de un verbo, entonces realiza varias funcionales que probablemente tienen vinculación secuencial o comunicacional.
  - Si la frase contiene palabras referidas al tiempo (primero, a continuación, entonces, después, cuándo, al comienzo, etc.) entonces tienen vinculación secuencia o temporal.
  - Si el predicado no tiene un objeto específico, sencillo luego del verbo, entonces está acotado lógicamente.
  - Inicializar, limpiar, etc. implican vinculación temporal.

- Diseño Modular efectivo – Acoplamiento
  - Es la medida de interconexión entre los módulos de una estructura de programa. Se busca un **bajo acoplamiento**.
  - El acoplamiento depende de la complejidad de interconexión entre los módulos, el punto donde se realiza una entrada o referencia a un módulo, y los datos que pasan a través de la interfaz.
  - Los distintos niveles son:

1. **Acoplamiento de datos:** Se pasa a través de la interfaz del módulo un argumento simple.
2. **Acoplamiento por estampado:** Se pasa una estructura de datos.
3. **Acoplamiento de control:** Se pasa un indicador de control de módulos, influye en la ejecución del módulo.
4. **Acoplamiento externo:** Cuando dos módulos están atados a un entorno externo al software (Ejemplo: E/S).
5. **Acoplamiento común:** Varios módulos hacen referencia a un área global de datos.
6. **Acoplamiento por contenido:** Un módulo hace uso de datos o información de control mantenidos dentro de los límites de otro módulo.

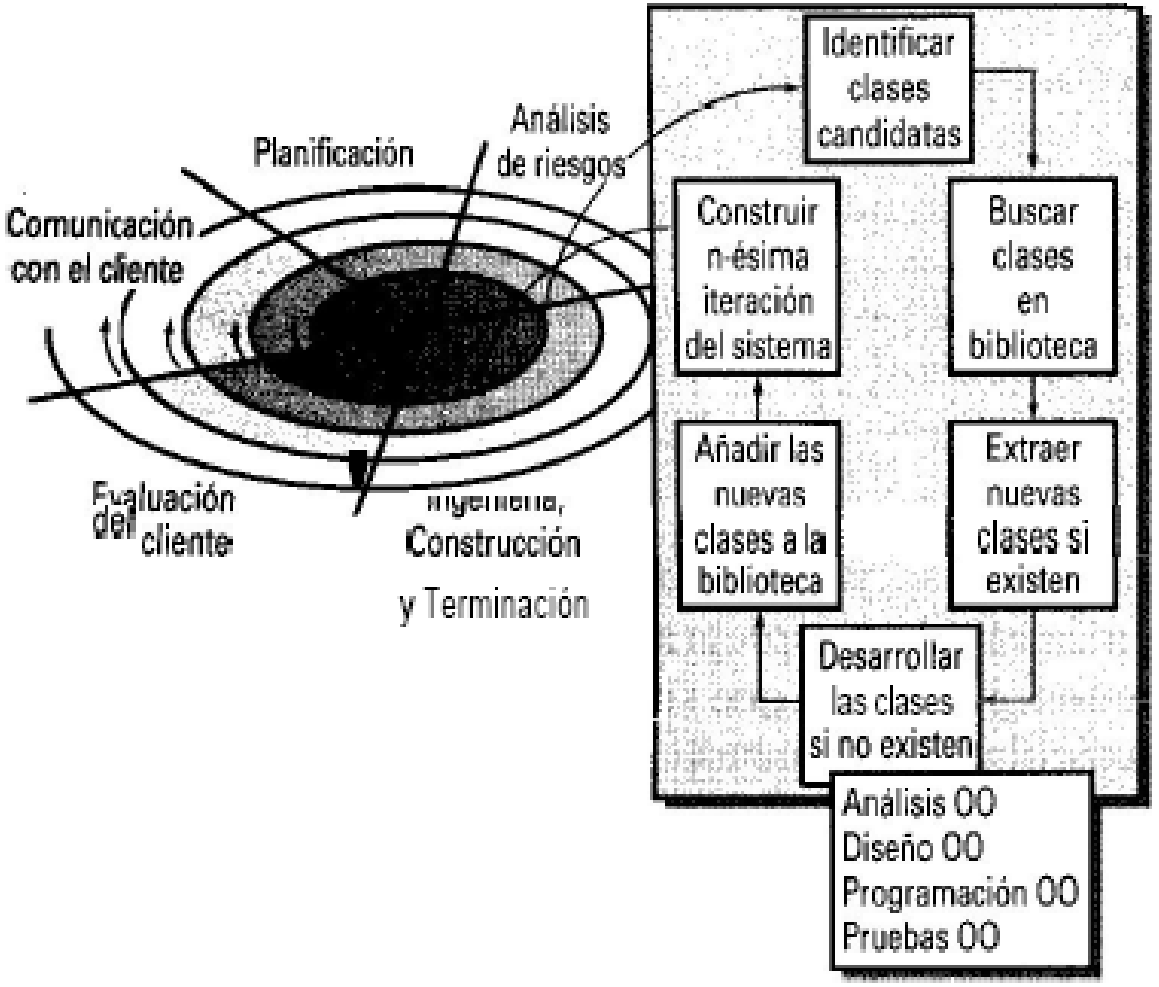
  - El nivel de acoplamiento más bajo es el llamado acoplamiento de datos, mientras que el grado más alto de acoplamiento es por contenido.

- Heurísticas de diseño para una modularidad efectiva
  - Una vez que se ha desarrollado una estructura de programa, se puede conseguir una modularidad efectiva aplicando los conceptos de diseño.
  - La arquitectura del programa se manipula de acuerdo con un conjunto de heurísticas (directrices o consejos):

1. Evaluar la primera iteración de la estructura del programa para reducir el acoplamiento y mejorar la cohesión.
2. Intentar minimizar las estructuras con mucho grado salida; intentar concentrar a medida que aumenta la profundidad.
3. Mantener el alcance del efecto de un módulo dentro del alcance del control de ese módulo.
4. Evaluar las interfaces de los módulos para reducir la complejidad, la redundancia y mejorar la consistencia.
5. Definir módulos cuya función sea predecible, pero evitar módulos que sean demasiado restrictivos.
6. Intentar conseguir módulos de entrada controlada, evitando conexiones patológicas (Acoplamiento por contenido).
7. Empaquetar el software basándose en las restricciones del diseño y los requisitos de portabilidad.

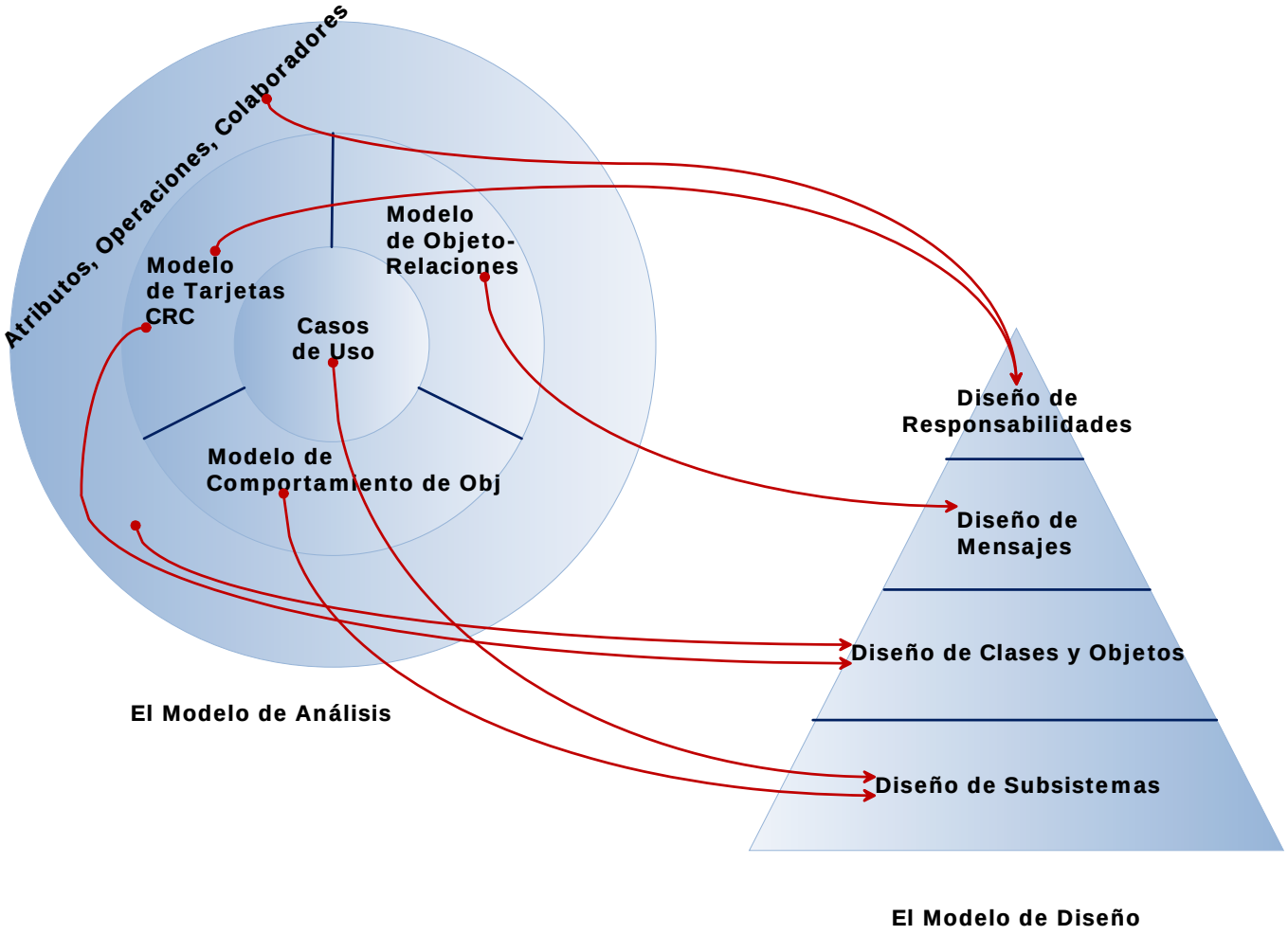
- Métodos de diseño – Diseño Orientado a Objetos
  - Los beneficios de la tecnología orientada a objetos se fortalecen si se usa antes y durante el proceso de ingeniería del software.
  - Esta tecnología orientada a objetos considerada debe hacer sentir su impacto en todo el proceso de ingeniería del software.
  - Un simple uso de programación orientada a objetos (POO) no brindará los mejores resultados.
  - Los ingenieros del software y sus directores deben considerar tales elementos el análisis de requisitos orientado a objetos (**AROO**), el diseño orientado a objetos (**DOO**), el análisis del dominio orientado a objetos (**ADOO**), sistemas de gestión de bases de dalos orientadas a objetos (**SGBDOO**) y la ingeniena del software orientada a objetos asistida por computadora (**ISOOAC**).

- Métodos de diseño – Diseño Orientado a Objetos (Cont.)
  - El modelo de Proceso OO



- Métodos de diseño – Diseño Orientado a Objetos (Cont.)
  - El proceso OO se mueve a través de una **espiral evolutiva** que comienza con la comunicación con el usuario. Es aquí donde se define el **dominio del problema** y se identifican las **clases básicas** del problema.
  - La **planificación y el análisis de riesgos** establecen una base para el plan del proyecto OO.
  - El trabajo técnico asociado con la **ingeniería del software OO** sigue el camino iterativo mostrado en la caja sombreada.
  - La ingeniería del software OO hace hincapié en la **reutilización**. Por lo tanto las clases se buscan en una biblioteca de clases existentes antes de construirse. Cuando una clase no puede encontrarse en la biblioteca, el desarrollador de software aplica análisis orientado a objetos (**AOO**), diseño orientado a objetos (**DOO**), programación orientada a objetos (**POO**) y pruebas orientadas a objetos (**PROO**) para crear la clase y los objetos derivados de la clase.
  - La nueva clase se pone en la biblioteca de tal manera que pueda ser reutilizada en el futuro.

- Métodos de diseño – Diseño Orientado a Objetos (Cont.)
  - El **diseño orientado a objetos** transforma el modelo de análisis creado usando análisis orientado a objetos, en un **modelo de diseño** que sirve como anteproyecto para el desarrollo de software.



  - CRC: Clases-Responsabilidades-Colaboraciones

- Métodos de diseño – Diseño Orientado a Objetos (Cont.)
  - Para **sistemas OO**, podemos definir una pirámide, las cuatro capas de diseño OO son:
    - ✓ **La capa subsistema**: Contiene una representación de cada uno de los subsistemas, para permitir al software conseguir sus requisitos definidos por el cliente e implementar la infraestructura que soporte los requerimientos del cliente.
    - ✓ **La capa de clases y objetos**: Contiene la jerarquía de clases, que permiten al sistema ser creado usando generalizaciones y cada vez especializaciones más acertadas. Esta capa también contiene representaciones.
    - ✓ **La capa de mensajes**: Contiene detalles de diseño, que permite a cada objeto comunicarse con sus colaboradores. Esta capa establece interfaces externas e internas para el sistema.
    - ✓ **La capa de responsabilidades**: Contiene estructuras de datos y diseños algorítmicos, para todos los atributos y operaciones de cada objeto.

  - A pesar de que existen similitudes entre los diseños convencionales y OO, se ha optado por renombrar las capas de la pirámide de diseño, para reflejar con mayor precisión la naturaleza de un diseño OO.

- Métodos de diseño – Diseño Orientado a Objetos (Cont.)
  - Como vimos anteriormente, fue propuesta y utilizada una gran variedad de métodos de análisis y diseño OO durante los 80 y 90. Estos métodos establecieron los fundamentos para la notación moderna de DOO, heurísticas de diseño y modelos.
  - A pesar de que la terminología y etapas de proceso para cada uno de estos métodos de DOO difieren, los procesos de DOO global son bastante consistentes. Para llevar a cabo un DOO, se deben ejecutar las siguientes **etapas generales**:

1. Describir cada subsistema y asignar a procesadores y tareas.
2. Elegir una estrategia para implementar la administración de datos, soporte de interfaz y administración de tareas.
3. Diseñar un mecanismo de control, para el sistema.
4. Diseñar objetos creando una representación procedural para cada operación, y estructuras de datos para los atributos de clase.
5. Diseñar mensajes, usando la colaboración entre objetos y relaciones.
6. Crear el modelo de mensajería.
7. Revisar el modelo de diseño y renovarlo cada vez que se requiera.

↳ Las etapas de diseño son iterativas. Deben ser ejecutadas incrementalmente, junto con las actividades de AOO, hasta que se produzca el diseño completo.

- Métodos de diseño – Diseño de Interfaz
  - **Describe cómo se comunica el software** consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean.
  - Una interfaz **implica un flujo de información** por ejemplo datos y/o control.
  - El diseño de interfaz se concentra en **tres áreas importantes**:
    - ✓ Diseño de interfaz entre los módulos
    - ✓ Diseño de interfaz entre el software y productores/consumidores no humanos de información (entidades externas)
    - ✓ Diseño de interfaz entre el hombre y la computadora
  - El diseño de interfaz interna depende de los datos que deben fluir entre los módulos y las características del lenguaje de programación en el que se va a implementar el software. P. e. El DFD describe como se transforman los objetos de datos al moverse a través del sistema. Las transformaciones del DFD se convierten en módulos dentro de la estructura del programa.
  - El diseño de interfaz externa empieza con una evaluación de cada entidad externa representada en el modelo de análisis. Se determinan los requisitos de datos y control de la entidad externa y se diseñan las interfaces externas apropiadas.

- Métodos de diseño – Diseño de Interfaz (Cont.)
- Aspectos del Diseño
  - Theo Mantel en su libro crea **tres reglas de oro** para el diseño de la interfaz:

1. Dar el control al usuario

- Existe una serie de principios que permiten lograrlo:
  - ✓ Definir los modos de interacción de manera que no obligue a que el usuario realice acciones innecesarias y no deseadas. P. e. corrector ortográfico
  - ✓ Tener en consideración una interacción flexible. P. e. uso de diferentes dispositivos como mouse, teclado.
  - ✓ Permitir que la interacción del usuario se pueda interrumpir y deshacer.
  - ✓ Aligerar la interacción a medida que avanza el nivel de conocimiento y permitir personalizar la interacción. P.e. macros.
  - ✓ Ocultar al usuario ocasional los entresijos técnicos.
  - ✓ Diseñar la interacción directa con los objetos que aparecen en la pantalla.

↳ Entresijo= Aspecto o característica poco conocido u oculto de una persona o cosa

- Métodos de diseño – Diseño de Interfaz – Aspectos del Diseño (Cont.)

2. Reducir la carga de memoria del usuario

– Cuanto más tenga que recordar un usuario, más propensa a errores será su interacción con el sistema.

– Una interfaz de usuario bien diseñada no pondrá a prueba la memoria del usuario. Siempre que sea posible, el sistema deberá «recordar» la información pertinente y ayudar a que el usuario recuerde mediante un escenario de interacción.

– Principios de diseño:

  ✓ Reducir la demanda de memoria a corto plazo.

  ✓ Establecer valores por defecto útiles.

  ✓ Definir las deficiencias que sean intuitivas. P.e. Ctrl/Alt + letra.

  ✓ El formato visual de la interfaz se deberá basar en una metáfora del mundo real.

  ✓ Desglosar la información de forma progresiva. De manera tal que la interfaz deberá organizarse de forma jerárquica.

- Métodos de diseño – Diseño de Interfaz – Aspectos del Diseño (Cont.)

3. Construir una interfaz consecuente

– La interfaz deberá adquirir y presentar la información de forma consecuente (consistente). Esto implica:

I. Toda la información visual está organizada de acuerdo con el diseño estándar que se mantiene en todas las pantallas.

II. Que todos los mecanismos de entrada se restrinjan a un conjunto limitado y que se utilicen consecuentemente por toda la aplicación.

III. Que los mecanismos para ir de tarea a tarea se hayan definido e implementado consecuentemente.

– Principios de diseño:

✓ Permitir que el usuario realice una tarea en el contexto adecuado. P.e. títulos de ventanas, iconos, codificaciones en colores consecuentes que posibiliten al usuario conocer el contexto del trabajo que está llevando a cabo.

✓ Mantener la consistencia en toda la familia de aplicaciones.

✓ Los modelos interactivos anteriores han esperanzado al usuario, no realicemos cambios a menos que exista alguna razón convincente para hacerlo. P.e Ctrl + G para grabar.

- Métodos de diseño – Diseño de Interfaz – Aspectos del Diseño (Cont.) – Modelos de diseño & Tipos de Usuarios
  - Cuando se diseña la interfaz entran en juego **4 modelos** diferentes:
  1. **Modelo de diseño** creado por el ingeniero de software
  2. **Modelo del usuario**, puede ser creado por el ingeniero de software u otros ingenieros
  3. **Percepción del usuario**
  4. **Imagen del sistema** creada por los que implementan el sistema
  - Estos 4 modelos se pueden conciliar y derivar en una representación consecuente de la interfaz, para lo cual se deben conocer los perfiles de edad, sexo, habilidades físicas, educación, antecedentes culturales o étnicos, motivación, objetivos y personalidad de los usuarios. Además se pueden establecer las siguientes categorías de usuarios:
    - ✓ **Principiantes**: no tienen conocimientos sintácticos ni conocimientos semánticos de la utilización de la aplicación.
    - ✓ **Usuarios esporádicos y con conocimientos**: poseen un conocimiento semántico razonable, pero una retención baja de la información necesaria para utilizar la interfaz.
    - ✓ **Usuarios frecuentes y con conocimientos**: poseen el conocimiento sintáctico y semántico suficiente, buscan modos abreviados de interacción.

- Métodos de diseño – Diseño de Interfaz – Aspectos del Diseño (Cont.)
  - A medida que evoluciona el diseño de la interfaz del usuario, emergen casi siempre cuatro aspectos comunes del diseño:
    - ✓ El tiempo de respuesta del sistema
    - ✓ La manipulación de la información de errores
    - ✓ Las facilidades de ayuda al usuario
    - ✓ El etiquetado de órdenes

  - El **tiempo de respuesta** del sistema es **la queja principal** de muchos sistemas interactivos. En general, el tiempo de respuesta de un sistema se mide desde el momento en que el usuario realiza alguna acción de control (por ejemplo pulsa Enter o hace click con el mouse) hasta que el SW responde con la salida o acción deseada.
  - Tiene 2 características importantes:
    - ✓ **Duración o retardo**: Si la duración de la respuesta es demasiado larga, es inevitable obtener como resultado la frustración y estrés del usuario. Pero una respuesta rápida también puede ser perjudicial ya que puede obligar al usuario a que se precipite y cometa errores.
    - ✓ **Variabilidad**: se refiere a la desviación del tiempo de respuesta promedio.

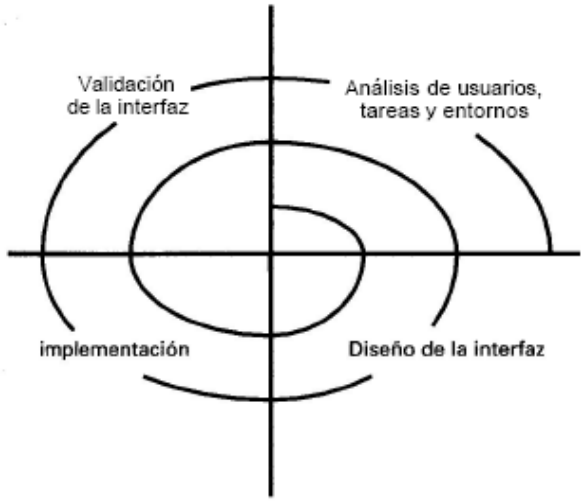
- Métodos de diseño – Diseño de Interfaz – Aspectos del Diseño (Cont.)
  - Los **mensajes de error y advertencias** son **malas noticias** enviadas a los usuarios cuando algo ha ido mal.
  - En el peor de los casos, ofrecen información inútil o engañosa y sirven solo para aumentar la frustración del usuario. P.e. “Error grave del sistema 14 A”. Este tipo de mensaje no ofrece ninguna indicación real de lo que está mal o dónde buscar información adicional.
  - En general un mensaje de error o de advertencia debería tener las siguientes **características**:
    - ✓ Describir el problema en un lenguaje que pueda entenderse
    - ✓ Proporcionar una información constructiva para recuperarse del error
    - ✓ Indicar cualquier consecuencia negativa del error (P. e. Archivos de datos potencialmente dañados) de manera que el usuario pueda comprobarlos para asegurarse de que no ha ocurrido, o corregirlos si lo están
    - ✓ Debería ir acompañado por una señal audible o visible
    - ✓ No debería hacer juicios → El texto nunca debería culpar al usuario

- Métodos de diseño – Diseño de Interfaz – Aspectos del Diseño (Cont.)
  - Casi todos los usuarios de un sistema necesitan la **ayuda** de vez en cuando. En algunos casos, una pregunta fácil dirigida a un compañero con más conocimientos basta. En otros, la única opción puede ser hacer una detallada investigación en los **manuales de usuario**. En muchos casos, el SW proporciona una **ayuda en línea** que permite al usuario responder una pregunta o resolver un problema sin abandonar la interfaz.

  - Las **órdenes escritas** con el teclado fueron durante un tiempo la forma más común de interacción entre el usuario y el SW, y fue comúnmente usado para aplicaciones de todo tipo. Hoy en día, el uso de interfaces orientadas a ventanas, selección y elección han reducido la dependencia del teclado, pero muchos usuarios continúan prefiriendo un modo de interacción orientado a órdenes.
  - Cuando se proporcionan órdenes se deben considerar aspectos tales como que todas las opciones del menú tengan su correspondiente orden, que tengan el mismo formato (usando Alt, o teclas de función), que el usuario pueda personalizar o abreviar las órdenes, etc.
  - En muchas aplicaciones se proporciona una utilidad para **macros** de órdenes que permite al usuario almacenar una secuencia de órdenes, utilizadas con frecuencia bajo un nombre definido por él mismo.

- Métodos de diseño – Diseño de Interfaz – Elementos del diseño
  - Para elaborar el diseño de la interfaz se utilizarán las siguientes herramientas:
  1. Diagrama de menúes
  2. Diseño de cada una de las pantallas del sistema de acuerdo con el diagrama jerárquico.
  3. Conteo de Puntos de función

- El proceso de diseño de la interfaz
  - El proceso de diseño de la interfaz de usuario es **iterativo** y se puede representar mediante un **modelo espiral**.
  - Se puede observar que el proceso de diseño de la interfaz de usuario acompaña cuatro actividades distintas de marco de trabajo:
  1. Análisis y modelado de usuarios. tareas v entornos.
  2. Diseño de la interfaz
  3. Implementación de la interfaz
  4. Validación de la interfaz



- Métodos de diseño – Diseño de Interfaz – Directrices para el Diseño de Interfaz
  - El diseño de interfaz se basa profundamente en la **experiencia del diseñador y en la documentación técnica** que se posea. Sin embargo, enumeraremos algunos consejos más importantes:
    - ✓ **Ser consistente**: usar un formato consistente para la selección del menú, introducción de órdenes, visualización de datos, etc.
    - ✓ **Ofrecer respuestas significativas**: proporcionar al usuario respuestas visuales y auditivas para garantizar que se establece una comunicación en los dos sentidos (usuario e interfaz)
    - ✓ **Pedir verificación de cualquier acción destructiva importante**: si un usuario pide eliminar un archivo, indica que se va a sobrescribir una información sustancial o pide finalizar un programa, debería aparecer un mensaje del tipo "¿Está ud. Seguro?"
    - ✓ **Permitir deshacer la mayoría de las acciones**: esta función, al igual que la de rehacer, deberían estar disponible en todas las aplicaciones
    - ✓ **Reducir la cantidad de información que se debe memorizar entre acciones**: no se debe esperar que el usuario recuerde una lista de números o nombres para reutilizarlos en las subsiguientes funciones.

- Métodos de diseño – Diseño de Interfaz – Directrices para el Diseño de Interfaz (Cont.)
  - ✓ **Buscar la eficiencia en el diálogo, el movimiento y el pensamiento:** el número de pulsaciones debería minimizarse, se debería considerar la distancia que debe moverse con el mouse entre selecciones al diseñar la distribución del monitor, el usuario no debería encontrarse con situaciones donde se tenga que preguntar “qué significa alguna cosa”.
  - ✓ **Perdonar los errores**: el sistema debería autoprotegerse de los errores que le pueden hacer fallar.
  - ✓ **Categorizar las actividades por función y organizar la pantalla de acuerdo a esto:** una de las ventajas del menú desplegable es la capacidad de organizar las órdenes por tipo. El diseñador debería intentar conseguir cohesión en la colocación de órdenes y acciones.
  - ✓ **Proporcionar ayudas sensibles al contexto.**
  - ✓ **Usar verbos de acción sencillos o frases verbales cortas para nombrar las órdenes:** un nombre largo es más difícil de reconocer y de recordar, además ocupan mucho espacio en las listas de menú.

- Métodos de diseño – Diseño de Interfaz – Directrices para el Diseño de Interfaz (Cont.)
  - ✓ **Mostrar solo la información que sea relevante en el contexto actual**: el usuario no debería vagar a través de datos, menúes y gráficos extraños para obtener la información relativa a una función específica del sistema.
  - ✓ **No abrumar al usuario con datos:** utilizar un formato de visualización que le permita una rápida asimilación de la información, los grafos o esquemas deberían reemplazar a las tablas voluminosas.
  - ✓ **Usar etiquetas consistentes, abreviaciones estándar y colores predecibles.**
  - ✓ **Producir mensajes de error significativos.**
  - ✓ **Usar mayúsculas y minúsculas**, tabulaciones y agrupamiento de texto para ayudar a entenderlo.
  - ✓ **Usar ventanas** para compartimentar diferentes tipos de información.
  - ✓ **Estudiar la "geografía" disponible en la pantalla** y usarla eficientemente.
  - ✓ **Minimizar el número de acciones de entrada de datos** que necesita realizar el usuario.

- Métodos de diseño – Diseño de Interfaz – Algunas consideraciones para el diseño de interfaz
  - Una de las cuestiones básicas a considerar en el diseño de Interfaz de usuario, es que **el usuario "no quiere utilizar tu aplicación"**. Quiere hacer su trabajo de la forma más sencilla y rápida posible, y la aplicación no es más que otra herramienta para ayudarle a lograrlo. Cuanto menos estorbe tu aplicación al usuario, mejor. Hay un par de citas del libro de Alan Cooper, About Face 2.0 que lo resumen bastante bien: "Imagina usuarios muy inteligentes pero muy ocupados" y "No importa lo genial que sea tu interfaz, menos es más siempre".
  - La **Ley de Fitt**: dice que cuanto más grande y más cercano es un objeto al puntero del mouse, más sencillo es el hacer clic sobre él. Esto es de sentido común pero muchas veces es ignorado completamente en el diseño de interfaces.
  - El tamaño de un elemento de la interfaz puede parecer mayor si lo colocamos en el borde de la pantalla, partiendo de ello, podemos concluir que los controles sobre los que queremos que sea más sencillo el hacer clic, deberían colocarse en los bordes o esquinas de la pantalla. El ejemplo más simple es el de los botones de cerrar, maximizar, etc. en una ventana. Otro ejemplo son las barras de desplazamiento, en la mayor parte de las aplicaciones se encuentran en los bordes de las ventanas.

- Métodos de diseño – Diseño de Interfaz – Algunas consideraciones para el diseño de interfaz (Cont.)
  - **Interfaces innecesarias**: cuando un usuario está trabajando, su atención está centrada en el trabajo que está realizando. Cada vez que tienen que concentrarse en la aplicación, les lleva tiempo el volver a centrarse en el trabajo. Por lo tanto, deberíamos minimizar la cantidad de distracción y de interferencias por parte de la aplicación. Cada aplicación tiene un elemento clave en la que centrarse, en un editor de texto, es el texto; en un navegador Web, es la página Web; así que deberíamos hacer de este elemento clave el centro de la interfaz. Un ejemplo son los diálogos de confirmación y de progreso. P. e. En algunas aplicaciones se lanza una ventana de diálogo cada vez que se pulsa sobre “Enviar/Recibir” para informarle del progreso en el proceso de comprobación, es preferible eliminar esta ventana de diálogo y reemplazarlo por una barra de progreso. Como corolario de este punto, tener en cuenta de **No colocar barreras en el camino del usuario** y lanzar una ventana de diálogo sólo si ésta contiene información útil.

- Métodos de diseño – Diseño de Interfaz – Algunas consideraciones para el diseño de interfaz (Cont.)
  - **Utilizar la potencia de la computadora**: cada vez que exista una decisión que tomar o trabajo que hacer, se debe intentar que la interfaz lo haga por el usuario. P. e. Si tengo abierta dos ventanas de la misma aplicación, cada entrada debería mostrar en la barra de tareas suficiente información para poder distinguirlas, de esta forma es posible seleccionar multitud de aplicaciones diferentes sin tener que pensar demasiado. Por otro lado, si las computadoras tienen tanto espacio de almacenamiento disponible, ¿por qué hay tantas aplicaciones que olvidan las preferencias del usuario cuando las cierran? P. e. Si cada vez que abro la ventana de una aplicación, hago clic para maximizar, escribo algo y cierro la aplicación; la siguiente vez que abra la misma, debería **recordar** el tamaño previo de la ventana, la posición, etc., a fin de que el usuario evite varios clicks.

- Métodos de diseño – Diseño de Interfaz – Algunas consideraciones para el diseño de interfaz (Cont.)
  - **Debe ser fácil distinguir los elementos y buscarlos**: los elementos de la pantalla que hacen cosas distintas deberían ser fácilmente distinguibles unos de otros. Los humanos somos buenos distinguiendo entre **cinco elementos**: podemos hacerlo instantáneamente sin pensar. Si hay más de cinco elementos tenemos que parar un momento y utilizar el cerebro para pensar qué es cada cosa. Por ello, es aconsejable que en la barra de herramientas se coloquen siempre los elementos más utilizados, minimizando el trabajo que el usuario tiene que realizar en los casos más comunes. Resumiendo, debemos **tener en cuenta que**: elementos que hacen cosas distintas deben ser fácilmente distinguibles entre sí, no abrumar al usuario con demasiadas opciones y los elementos seleccionados deben ser sencillos de distinguir y leer.

- Métodos de diseño – Diseño de Interfaz – Usando iconos en el diseño de interfaces
  - Los iconos no son meros elementos decorativos, sino parte esencial de los mecanismos de interacción de cualquier interfaz que deben ser diseñados cuidadosamente. Aunque los iconos tienen limitaciones, su uso adecuado aporta grandes **ventajas**:
    - ✓ Los iconos **sustituyen a una unidad de significado** (idea, concepto, acción, etc.) que representada con texto ocuparía más espacio. La principal ventaja es que, mediante iconos, se pueden representar más unidades en un menor espacio.
    - ✓ Por ello los iconos son de gran utilidad en interfaces en las que es muy importante obtener una **funcionalidad máxima** en el mínimo espacio y con la máxima rapidez.
    - ✓ **El significado debe ser aprendido**, pero esto no es problema cuando el uso es frecuente y repetido o cuando existe una alta motivación, razones de trabajo o estudios.
    - ✓ Los iconos también son adecuados para interfaces donde es **importante el aspecto visual**. Cuando despertar la curiosidad del usuario forma parte integral del objetivo de la aplicación o cuando se desea que el usuario investigue y descubra por sí mismo el funcionamiento de la interfaz, los iconos juegan un gran papel.

- Métodos de diseño – Diseño de Interfaz – Usando iconos en el diseño de interfaces
  - Los iconos son representaciones o metáforas y como todas las metáforas tienen una serie de **limitaciones**:
    - ✓ A pesar de la creencia común, los iconos no se reconocen más rápido que los textos. En las primeras experiencias los usuarios cometen significativamente más errores en el reconocimiento de iconos que en el de textos. Estos errores pueden ser críticos en una primera visita a un sitio Web al que el usuario puede decidir no volver, pero menos importantes en aplicaciones donde el usuario tiene más motivación para el aprendizaje y el uso más repetitivo.
    - ✓ Los iconos son siempre subjetivos, están sujetos a la interpretación individual y subjetiva de cada persona a partir de su experiencia. Nunca son totalmente claros e inequívocos y existe riesgo de malentenderlos. Por esta razón no se recomienda usar iconos para operaciones críticas.

- Métodos de diseño – Importancia del diseño
  - Antes de comenzar con el desarrollo de cualquier proyecto, se conduce el **Análisis de Sistemas** para detectar todos los detalles de la situación actual de la empresa.
  - La información reunida con este análisis sirve como base para crear varias **estrategias de Diseño**. Los administradores del proyecto son quienes deciden las estrategias a seguir.

  - Por lo tanto, podemos inferir que:

1. Durante el diseño tomamos decisiones que afectarán al éxito del desarrollo del software y a la facilidad con la que se podrá mantener.
2. El Diseño es la única manera de materializar con precisión los requerimientos del cliente.
3. El diseño de software sirve como fundamento para todas las fases posteriores o la ingeniería y mantenimiento del software.

  - La importancia del diseño del software se puede resumir con una sola palabra: **CALIDAD**. Dentro del diseño es donde se fomenta la calidad del Proyecto.

- Calidad del Software
  - En el mercado competitivo actual, vender es lo más importante, y sólo se consigue con la seguridad de la aceptación por parte del cliente.
  - La aceptación es consecuencia de la adecuación del producto a las necesidades del cliente, es decir, depende de la **calidad del producto** → La calidad se ha convertido en un objetivo fundamental de las empresas.
  - Todas las metodologías y herramientas de la ingeniería de software tienen un único fin: **producir software de gran calidad.**
  - Los requisitos del software son la base de las medidas de la calidad.

- Procesos de Gestión de Calidad
  - Los procesos de Gestión de calidad del proyecto **incluyen todas las actividades** de la organización que determinan las políticas, los objetivos y las responsabilidades relativas a la calidad de modo que el proyecto satisfaga las necesidades por las cuales se emprendió.

  - Implementa el sistema de gestión de calidad a través de la política, los procedimientos y los procesos de planificación de calidad, aseguramiento y control de calidad, con actividades de mejora continua de los procesos que se realizan durante todo el proyecto.

- Calidad del Software – Definición
  - Grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario. (IEEE Std. 610-1990)
  - El conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas. (ISO 8402)
  - Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente. (Pressman)
  - La anterior definición sirve para hacer hincapié en tres puntos:

1. Los **requisitos del software** son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.
2. Los **estándares** especificados definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del SW. Si no se siguen esos criterios, casi siempre habrá falta de calidad.
3. Existe un conjunto de **requisitos implícitos** que a menudo no se mencionan (P.e. facilitar el uso o buen mantenimiento). Si el SW se ajusta a sus requisitos explícitos pero falla en alcanzar los requisitos implícitos, la calidad del SW queda en entredicho.

- Calidad del Software y Garantía de calidad (SQA)
  - La garantía o aseguramiento de calidad del software (**SQA** - **S**oftware **Q**uality **A**ssurance ó **GCS** – **G**arantía de **C**alidad del **S**oftware) es una **actividad de protección** que se aplica a lo largo de todo el proceso del software.
  - El costo de corregir y detectar errores producidos en las primeras fases de desarrollo de software es mayor a medida que nos encontramos más alejados de éstas. La propuesta de SQA es empujar las tareas relacionadas con la calidad desde las primeras fases del proyecto.
  - SQA engloba:

1. Un enfoque de gestión de calidad
2. Tecnología de ingeniería del software efectiva (métodos y herramientas de análisis, diseño, codificación y prueba)
3. Revisiones técnicas formales (RTF) que se aplican durante el proceso del software
4. Una estrategia de prueba multiescalada
5. El control de la documentación del software y de los cambios realizados
6. Un procedimiento que asegure un ajuste a los estándares de desarrollo del software (cuando sea posible)
7. Mecanismos de medición y de generación de informes

- Calidad del Software y Garantía de calidad (Cont.)
  - La **historia** de la garantía de calidad en el desarrollo de software es paralela a la historia de la calidad en la creación de hardware.
  - Durante **los primeros años** de la informática (los años cincuenta y sesenta), la calidad era responsabilidad únicamente del programador.
  - Durante los años **setenta** se introdujeron estándares de garantía de calidad para el software en los contratos militares para desarrollo de software y se han extendido rápidamente a los desarrollos de software en el mundo comercial.
  - En los **años posteriores** se continuó introduciendo y desarrollando estándares para garantizar la calidad y se desarrollo un **plan de SQA**, que estudiaremos a continuación.

  - Ampliando la definición presentada anteriormente, SQA es un **patrón de acciones planificado y sistemático** que se requiere para asegurar la calidad del software.
  - El ámbito de la responsabilidad de la garantía de calidad se puede caracterizar mejor parafraseando un popular anuncio de coches: **La calidad es la 1° tarea**.
  - La implicación para el SW es que aquellos que constituyen una organización tienen responsabilidad de SQA: los ingenieros de software, jefes de proyectos, clientes, vendedores, y aquellas personas que trabajan dentro de un grupo de SQA.

- Calidad del Software y Garantía de calidad (Cont.)
  - La garantía de calidad del software es el conjunto de actividades planificadas y sistemáticas (auditoría y funciones de información de la gestión) necesarias para aportar la confianza en que el producto (software) satisfará los requisitos dados de calidad.
  - SQA **se diseña para cada aplicación** antes de comenzar a desarrollarla y no después.
  - El **objetivo** de la garantía de calidad es proporcionar la gestión para informar de los datos necesarios sobre la calidad del producto, así se va adquiriendo una visión más profunda y segura de que la calidad del producto está cumpliendo sus objetivos.
  - Por supuesto, si los datos proporcionados mediante la garantía de calidad **identifican problemas**, es responsabilidad de la gestión afrontarlos y aplicar los recursos necesarios para resolver aspectos de calidad.
  - Algunos autores prefieren decir garantía de calidad en vez de aseguramiento.
    - ✓ **Garantía** puede confundir con garantía de productos
    - ✓ **Aseguramiento** pretende dar confianza en que el producto tiene calidad

- Calidad del Software y Garantía de calidad (Cont.)
  - Se define a la calidad como **una característica o atributo de algo**.
  - Como un atributo, la calidad se refiere a las **características mensurables**, cosas que se pueden comparar con estándares conocidos como longitud, color, propiedades eléctricas, maleabilidad, etc.
  - De esta forma se podría decir que la **calidad de los productos** puede medirse como una **comparación de sus características y atributos** y así, este concepto puede aplicarse a cualquier producto.

  - Dentro del contexto de la ingeniería del software:
    - ✓ Una **medida** proporciona una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto.
    - ✓ La **medición** es el acto de determinar una medida.
    - ✓ El IEEE Standard Glossary of Software Engineering Terms '93 define **métrica** como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

  - Uno de los principales objetivos de dar calidad a los productos es **minimizar las diferencias entre unidades producidas**.

- Calidad del Software y Garantía de calidad (Cont.)
  - Sin embargo, el **software como entidad intelectual**, es más difícil de caracterizar que los objetos físicos.
  - No obstante, sí existen las medidas de características de un programa → Métricas:
    - ✓ Complejidad ciclomática
    - ✓ Cohesión
    - ✓ Número de puntos de función
    - ✓ Líneas de código
  - Cuando se examina un elemento según sus características mensurables, se pueden encontrar dos tipos de calidad:
    - ✓ **Calidad del diseño**
    - ✓ **Calidad de concordancia**

- Calidad del Software y Garantía de calidad (Cont.)
  - Calidad del diseño
  - La calidad de diseño se refiere a las **características** que especifican los ingenieros de software **para un elemento**. El grado de materiales, tolerancias y las especificaciones del rendimiento contribuyen a la calidad del diseño. Cuando se utilizan materiales de alto grado y se especifican tolerancias más estrictas y niveles más altos de rendimiento, la calidad de diseño de un producto aumenta, si el producto se fabrica de acuerdo con las especificaciones.

  - Calidad de concordancia
  - La calidad de concordancia es el **grado de cumplimiento de las especificaciones de diseño** durante su realización. Una vez más, cuanto mayor sea el grado de cumplimento, más alto será el nivel de calidad de concordancia.

  - En el desarrollo del software, la calidad de diseño comprende los requisitos, especificaciones y el diseño del sistema.
  - La calidad de concordancia es un aspecto centrado principalmente en la implementación. Si la implementación sigue el diseño, y el sistema resultante cumple los objetivos de requisitos y de rendimiento, entonces la calidad de concordancia es alta.

- Calidad del Software y Garantía de calidad (Cont.)
- Inconvenientes para su establecimiento
  - Los responsables del desarrollo se resisten a hacer frente a los costes extras inmediatos.
  - Los profesionales creen que ya están haciendo todo lo que hay que hacer.
  - Nadie sabe dónde situar esa función dentro de la organización.
  - Todos quieren evitar el papeleo que SQA tiende a introducir en el proceso de ingeniería del software.

- Prerrequisitos
  - Adoptar métodos, herramientas y procedimientos de ingeniería del software.
  - Elegir el paradigma efectivo para el desarrollo.

- Calidad del Software y Garantía de calidad (Cont.)
- Propósito:
  - Proporcionar visibilidad sobre los procesos utilizados por el proyecto de software y sobre los productos que genera.

- Objetivos
  1. Planificar las actividades de aseguramiento de la calidad.
  2. Revisar y auditar objetivamente los productos y las actividades para verificar que estén conformes con los procedimientos y estándares.
  3. Proporcionar los resultados de estas revisiones o auditorías informando a la dirección.

- El grupo encargado de SQA
  - Trabaja con el equipo del proyecto desde el inicio.
  - Debe ser objetivo e independiente.
  - Ayuda al proyecto, más que controlar sus actividades.

  ↳ La actividad de SQA es el proceso de verificación de que los estándares sean aplicados correctamente. En los proyectos pequeños esto se puede realizar por el equipo de desarrollo, pero en proyectos grandes, se debe dedicar a este rol un grupo específico.

- Calidad del Software y Garantía de calidad (Cont.)
- Pasos a seguir
  1. Realizar una auditoría SQA/GCS
  2. La SQA es una actividad que se aplica a lo largo del proceso de I.S. desde que comienza el desarrollo hasta que el software queda fuera de circulación.
  3. Las actividades de SQA sirven para identificar, organizar y controlar los cambios así como garantizar que se implementen correctamente y que se informe acerca de los mismos.
  4. Se evalúa la necesidad del SQA, así como los pros y los contras.
  5. Se establece un plan de SQA y se adquieren o desarrollan unos estándares.

- Calidad del Software y Garantía de calidad (Cont.)
- Ventajas
  - Menos defectos latentes en el software
  - Menor esfuerzo y tiempo, tanto de prueba como de mantenimiento
  - Mayor fiabilidad y satisfacción del cliente
  - Reducción de los costes de mantenimiento
  - Disminución del coste del ciclo de vida

- Desventajas
  - No disponibilidad de recursos necesarios en empresas pequeñas
  - La dificultad del cambio cultural
  - La necesidad de un gasto destinado a la ingeniería del software

- Calidad del Software y Garantía de calidad – Técnicas
- Revisiones del software
  - Las revisiones constituyen un **filtro para el proceso** de ingeniería del software.
  - Una revisión puede servir para:
    - ✓ Señalar la **necesidad de mejoras** en el producto.
    - ✓ Confirmar las partes de un producto en las que no es deseable o **no es necesaria una mejoría**.
    - ✓ Conseguir un **trabajo técnico de una calidad más uniforme**, o al menos más predecible que la que se puede conseguir sin revisiones, con el fin de hacer más manejable el trabajo técnico.

  - Los métodos más comunes son:

1. **Auditoría PPQA** (**P**rocess and **P**roduct **Q**uality **A**ssurance): Es la actividad de garantizar que el proceso y el producto de trabajo se ajustan al plan acordado.
2. **Pruebas de Validación**: Es el acto de introducir datos, los cuales el tester sabe que son erróneos en la aplicación.

- Calidad del Software y Garantía de calidad – Técnicas – Revisiones del software (Cont.)

3. **Comparación de datos**: Técnica que se realiza comparando los resultados de una aplicación con parámetros específicos con los resultados de otra aplicación previamente creada, introduciendo los mismos parámetros de manera que se obtenga un resultado exacto.
4. **Prueba de esfuerzo** (Stress Testing): Se realiza cuando el SW es utilizado de la manera más "ruda" posible en un período de tiempo para ver si trabaja con altos niveles de carga.
5. **Pruebas de Uso**: Contar con usuarios que no estén familiarizados con el SW para probarlo por un tiempo determinado, ofrece retroalimentación a los desarrolladores acerca de las dificultades que encontraron. Esta es la mejor manera de realizar mejoras a la interfaz.
6. **Revisiones por Pares** (Peer Reviews): Son actividades efectivas para el control de la calidad. Pueden aplicarse al análisis, diseño y codificación.
7. **Revisión Técnica formal** (RTF): Es una actividad de garantía de calidad de SW. Es una revisión que incluye recorridos, inspecciones y revisiones cíclicas.

- Calidad del Software y Garantía de calidad – Técnicas (Cont.)
- Revisiones por Pares (Peer Reviews)
  - Consiste en la revisión del código de un programador por otros programadores (sus pares). Se puede poner en práctica creando un panel que se encarga de revisar periódicamente muestras de código.
  - En la revisión por pares se analizan o revisan factores:
    - ✓ Técnicos
    - ✓ De procedimiento
    - ✓ Humanos
  - Las revisiones por pares son las siguientes:
    - ✓ Walkthroughs
    - ✓ Inspecciones de código
  - **Walkthroughs** (recorridos): es usado para revisar especificaciones de requerimientos o de diseño.
  - Objetivos:
    - ✓ Detectar posibles defectos.
    - ✓ Identificar oportunidades de mejora.
    - ✓ Examinar alternativas.

- Calidad del Software y Garantía de calidad – Técnicas (Cont.)
- Revisiones por Pares (Peer Reviews)
  - **Inspecciones de código**: No son excluyentes con el testing. Cada desarrollador puede encontrar distintos tipos de defectos. No hay tiempo ni dinero para inspeccionar todo. Se suele centrar la inspección en los módulos más críticos. Es recomendable realizarla después de una prueba básica.
  - Objetivos:
    - ✓ Detectar defectos.
    - ✓ Elegir el camino de resolución.
    - ✓ Verificar la resolución (Los defectos deben ser corregidos).
    - ✓ Asegurar consenso sobre el trabajo y la calidad.
    - ✓ Potenciar el trabajo en equipo.
    - ✓ Obtener datos para las métricas.

  ↳ Se utilizará un checklist para realizar las revisiones.

- Calidad del Software y Garantía de calidad – Técnicas (Cont.)
- Revisión Técnica Formal (RTF)
  - RTF o inspección es el **filtro más efectivo** para mejorar la calidad del software.
  - Una revisión técnica formal es una actividad de garantía de calidad del SW llevada a cabo por los ingenieros de SW y el equipo de SQA.
  - La **RTF se lleva a cabo mediante una reunión** y sólo tendrá éxito si está bien planificada, controlada y atendida.
  - La RTF también sirve para promover la seguridad y continuidad, ya que varias personas se familiarizarán con partes del SW que de algún modo no habían visto.

  - Los objetivos son:
    - ✓ Descubrir errores en la función, la lógica o la implementación de cualquier representación del software.
    - ✓ Verificar que el software bajo revisiones alcanza sus requisitos.
    - ✓ Garantizar que el software ha sido representado de acuerdo con ciertos estándares predefinidos.
    - ✓ Conseguir un software desarrollado de forma uniforme.
    - ✓ Hacer que los proyectos sean más manejables.

- Calidad del Software y Garantía de calidad – Técnicas (Cont.)
- Revisión Técnica Formal (RTF) – Reunión RTF
  - La reunión RTF debe ajustarse a las siguientes restricciones:
    - ✓ Deben convocarse para la revisión entre 3 y 5 personas.
    - ✓ Se debe preparar por adelantado pero sin que requiera más de dos horas de trabajo por cada persona.
    - ✓ La reunión no debe durar más de 2 horas.
    - ✓ Con las anteriores limitaciones, cada RTF se centra en una parte específica del SW total. Al limitar el centro de atención de la RTF, la probabilidad de descubrir errores es mayor.

  - Al final de la reunión los participantes deben decidir si:

1. **Aceptar el producto** sin posteriores modificaciones.
2. **Rechazar el producto** debido a los serios errores encontrados. Una vez corregidos debe hacerse otra reunión.
3. **Aceptar el producto provisionalmente**. Se han encontrado errores menores que deben ser corregidos sin necesidad de otra reunión.

- Calidad del Software y Garantía de calidad – Técnicas (Cont.)
- Revisión Técnica Formal (RTF) – Registro e informe
  - Durante la reunión se van registrando todas las dudas y errores que vayan surgiendo.
  - Al final de la reunión se resumen todas las incidencias y se genera una lista de sucesos de revisión.
  - Además se prepara un informe de la RTF que debe responder a 3 preguntas:

1. ¿Qué fue revisado?
2. ¿Quién lo revisó?
3. ¿Qué se descubrió y cuáles son las conclusiones?

  - La lista de sucesos de revisión tiene dos propósitos:
    - ✓ Identifica las áreas problemáticas dentro del producto.
    - ✓ Sirve como lista de comprobación de puntos de acción que guían al programador para realizar correcciones.

- Calidad del Software y Garantía de calidad – Técnicas (Cont.)
- Revisión Técnica Formal (RTF) – Directrices para la RTF
  - Se deben establecer de antemano directrices para conducir la RTF. Las más comunes son:
    - ✓ Revisar el producto, no al productor.
    - ✓ Fijar una agenda y mantenerla. El jefe de revisión es el encargado de mantener el plan de la reunión.
    - ✓ Limitar el debate y las impugnaciones.
    - ✓ Enunciar áreas de problemas, pero no intentar resolver ningún problema que se ponga de manifiesto.
    - ✓ Tomar notas escritas. De preferencia, estas notas deber ser escritas en un pizarrón de forma que puedan ser comprobadas por el resto de los revisores.
    - ✓ Limitar el número de participantes e insistir en la preparación anticipada.
    - ✓ Desarrollar una lista de comprobaciones para cada producto que vaya a ser revisado.
    - ✓ Disponer recursos y una agenda para las RTF.
    - ✓ Llevar a cabo un buen entrenamiento de los revisores.
    - ✓ Repasar las revisiones anteriores. Pueden ser beneficiosas para descubrir problemas en el propio proceso de revisión. El primer producto que se haya revisado puede establecer las propias directivas de revisión.

- Calidad del Software - Software Quality Assurance Plan (SQAP)
  - Una de las principales fases dentro de la elaboración de un proyecto es el Aseguramiento de la Calidad del Software (SQA), es decir, un modelo sistemático y planeado de todas las acciones necesarias para proveer la confianza adecuada, según los requerimientos técnicos establecidos, de cada producto e ítem del proyecto. Un sinónimo del aseguramiento de la calidad del software es aseguramiento del producto de software.
  - La actividad del SQA es el proceso de verificación de que los estándares sean aplicados. En proyectos pequeños esto se puede realizar por el equipo de desarrollo, pero en proyectos grandes, un grupo específico se debe dedicar a este rol.
  - El **plan de aseguramiento de la calidad del software** (SQAP) define cuan adherido a estos estándares se debe monitorear.
  - El SQAP contiene una lista de comprobación para las actividades que se deben llevar a cabo para asegurar la calidad del producto.
  - Para cada actividad, en las que tiene responsabilidad el SQA, se debe crear un plan para su monitoreo.
  - En el SQAP se recogen una serie de medidas que permiten establecer el nivel de calidad de los desarrollos en cualquier momento en relación a los parámetros de calidad establecidos en el mismo, de modo que los gestores de proyecto puedan dar respuesta adecuada a las acciones a tomar de acuerdo a las medidas que se recogen en el plan.

- Calidad del Software - Software Quality Assurance Plan (Cont.)
  - Para poder realizar una buena adherencia con los estándares se debe medir cuantitativamente, donde sea posible, los aspectos de calidad (P. e. complejidad, confiabilidad, mantenimiento, seguridad, defectos, número de problemas) utilizando métricas bien establecidas.
  - Para cumplir con esto, se deben realizar las siguientes actividades:

    ✓ Administración
    ✓ Documentación
    ✓ Estándares, prácticas, convenciones y métricas
    ✓ Revisiones e intervenciones
    ✓ Actividades de testeo
    ✓ Reporte de errores y acciones correctivas
    ✓ Herramientas, técnicas y métodos
    ✓ Código y control de media
    ✓ Control:
      - Colección de registros, mantenimiento y retención
      - Entrenamiento
      - Administración del riesgo

- Calidad del Software - Software Quality Assurance Plan (Cont.)
  - Luego, todas estas actividades se deben documentar en el Plan de Aseguramiento de la Calidad del Software, el cual irá evolucionado es las sucesivas fases, es decir:

    - ✓ **Fase UR**: Definición de requerimientos de usuario (User Requirements).
    - ✓ **Fase SR**: Requerimientos de Software (Software Requirements).
    - ✓ **Fase AD**: Diseño arquitectónico (Architectural Design) del software.
    - ✓ **Fase DD**: Diseño detallado del software y producción de código (Detailed Design).
    - ✓ **Fase TR**: Transferencia del software a operaciones (Transfer)
    - ✓ **Fase OM**: Fase de operaciones y mantenimiento del ciclo de vida del software (Operations and Maintenance).

- Calidad del Software - Software Quality Assurance Plan (Cont.)
  - La garantía de calidad del software es una **actividad de protección** que se aplica a cada paso del proceso del software.
  - SQA comprende:
    - ✓ Procedimientos para la aplicación efectiva de métodos y herramientas
    - ✓ Revisiones técnicas formales
    - ✓ Técnicas y estrategias de prueba
    - ✓ Dispositivos *poku-yoke* (prueba de errores)
    - ✓ Procedimientos de control de cambios
    - ✓ Procedimientos de garantía de ajuste a los estándares
    - ✓ Mecanismos de medida e información
  - SQA es complicada por la compleja naturaleza de la calidad del software → un atributo de los programas de computadora que se define como **concordancia con los requisitos definidos explícita e implícitamente.**
  - Cuando se considera de forma más general, la calidad del software engloba muchos factores de proceso y de producto diferentes con sus métricas asociadas. Las revisiones del software son una de las actividades más importantes de la SQA.

- Calidad del Software - Resumen
  - Las **revisiones sirven como filtros** durante todas las actividades de ingeniería del software, eliminando defectos mientras que no son relativamente costosos de encontrar y corregir.
  - La **RTF es una revisión** específica que se ha mostrado extremadamente efectiva en el descubrimiento de errores.
  - Para llevar a cabo adecuadamente una **SQA**, se deben recopilar, evaluar y distribuir todos los datos relacionados con el **proceso de ingeniería del software**.
  - La **SQA estadística** ayuda a mejorar la calidad del producto y la del proceso de software. Los modelos de fiabilidad del software amplían las medidas, permitiendo extrapolar los datos recogidos sobre los defectos, a predicciones de tasas de fallo y de fiabilidad.
  - **La capacidad para garantizar la calidad es la medida de madurez de la disciplina de ingeniería**. Cuando se sigue de forma adecuada esa guía anteriormente mencionada, lo que se obtiene es la madurez de la ingeniería del software.

- Bibliografía

  - Ingeniería de software – Un enfoque práctico – Cuarta Edición de Roger S. Pressman - Editorial Mc Graw Hill

  - The practical Guide to Structured Systems Design – second edition de Meilir Page-Jones – Editorial Prentice Hall

  - Análisis y Diseño de Sistemas – Tercera Edición de Kendall & Kendall – Editorial: Prentice Hall