Ingeniería de Software

Diseño de Software
Clase 1

Diseño de Software

- Según Sommerville: El diseño de software es una actividad creativa donde se identifican los componentes del software y sus relaciones, con base en los requerimientos de un cliente.
- Algunas veces, hay una etapa de diseño separada y este último se modela y documenta. En otras ocasiones, un diseño se halla en la mente del programador o se bosqueja en hojas de papel.
- El diseño trata sobre cómo resolver un problema, de modo que siempre existe un proceso de diseño.

Diseño de Software

- El diseño se define como "el proceso de definición de la arquitectura, componentes, interfaces y otras características de un sistema o componente" y "el resultado de [ese] proceso".

- Como proceso → es una actividad del ciclo de vida en la cual se analizan los requisitos de software para producir una descripción interna* del software que servirá como base para su construcción.

- Como resultado → describe la arquitectura del software (cómo se descompone y se organiza en componentes) y las interfaces entre esos componentes.

Diseño de Software

- Consiste en dos actividades que se encuentran entre el análisis de requisitos del software y la construcción del software:
  - Diseño arquitectónico de software (a veces llamado diseño de alto nivel): desarrolla la estructura y la organización de alto nivel del software e identifica los diversos componentes.
  - Diseño detallado del software: especifica cada componente con detalles suficientes para facilitar su construcción.

Diseño de Software

- El diseño de un sistema es ***correcto*** si un sistema construido de acuerdo a ese diseño satisface los requerimientos del sistema
- Pero el objetivo del diseño no es encontrar **el** diseño correcto sino
  - Encontrar el mejor diseño posible
  - dentro de las limitaciones impuestas por
    - los requerimientos,
    - el ambiente físico del sistema
    - y el ambiente social donde el mismo va a operar
    - ¿otras limitaciones?

Diseño de Software

- Un diseño claramente debe ser:
  - Verificable
  - Completo (implementa toda la especificación)
  - “Rastreable”. Se puede rastrear hacia los requerimientos que diseña (“*traceable”*)
- Sin embargo, las dos cosas más importantes concernientes a los diseñadores es que el diseño sea:
  - Eficiente
  - Simple
  - ¿Por qué?

Diseño de Software

- Crear un diseño simple y eficiente de un sistema “grande” puede ser una tarea extremadamente compleja
  - Actividad básicamente creativa
  - No puede reducirse a una serie de pasos a seguir
  - Sin embargo, se pueden dar guías
- Si se logra manejar la complejidad
  - Se reducen los costos del diseño
  - Se reduce la posibilidad de introducir defectos durante el diseño

Proceso de Diseño de Software

- El proceso es muy creativo
- Algunos autores plantean las siguientes actividades:
  - Postular una solución*
  - Construir un modelo de la solución*
  - Evaluar el modelo contra los requisitos originales*
  - Elaborar el modelo para producir una especificación detallada de la solución.
  - (*) Esto se hace hasta encontrar una solución adecuada.

Principios de Diseño

- Abstracción

  La abstracción es "una vista de un objeto que se enfoca en la información relevante para un propósito particular e ignora el resto de la información".

  Permite al diseñador considerar un componente sin preocuparse por los detalles de implementación (por ej. describiendo comportamiento exterior sin describir los detalles internos).

Principios de Diseño

- Acoplamiento

  "una medida de la interdependencia entre módulos en un programa informático"

- Más acoplamiento → más díficil comprender y modificar
- Depende de → cuánta información se necesita para entender un módulo, y qué tan compleja y explícita es esa información.
- Cohesión

  "una medida de la fuerza de asociación de los elementos dentro de un módulo"

- Más cohesión → más fáciles de comprender y modificar

Principios de Diseño

- Descomposición y modularización

  Descomponer y modularizar significa que el software grande se divide en una serie de componentes nombrados más pequeños que tienen interfaces bien definidas que describen las interacciones de los componentes. Usualmente el objetivo es colocar diferentes funcionalidades y responsabilidades en diferentes componentes.

- Un sistema se considera modular si consiste de componentes que se pueden implementar separadamente y el cambio en un componente tiene mínimos impactos en otros componentes.

Principios de Diseño

- Encapsulación y ocultamiento de información

  Significa agrupar y empaquetar los detalles internos de una abstracción y hacer que esos detalles sean inaccesibles para las entidades externas.

- Separación de la interfaz y la implementación

  La separación de la interfaz y la implementación implica la definición de un componente especificando una interfaz pública (conocida por los clientes) que es independiente de los detalles de cómo se realiza el componente.

Principios de Diseño

- Suficiencia, integridad y primitivismo

  Lograr la suficiencia y la integridad significa garantizar que un componente de software capture todas las características importantes de una abstracción y nada más. Primitividad significa que el diseño debe basarse en patrones fáciles de implementar.

- Separación de intereses

  Permite enfrentarse a los distintos aspectos individuales de un problema de forma de concentrarse en cada uno por separado. Por ejemplo: según el tiempo (ciclo de vida), cualidades (correctitud y eficiencia).

Principios de Diseño

- Dividir y Conquistar
- Debido a la complejidad de los grandes problemas y las limitaciones de la mente humana estos no se pueden atacar como una unidad monolítica
  - Aplicar dividir y conquistar - Dividir en piezas que pueden ser “conquistadas” por separado.
  - Las piezas están relacionadas. Juntas forman el sistema.
  - Existe comunicación y cooperación entre ellas - esto agrega complejidad, que surge de la propia partición.
  - Cuando el costo de particionar sumado la complejidad agregada supera los ahorros logrados por particionar se debe detener el proceso

Principios de Diseño

- Reuso
- Utilizar nuevamente algo (o construir pensado en),
- Hay varios niveles.
  - Nivel de abstracción - Se utiliza el conocimiento de abstracciones exitosas en el diseño del software.
  - Nivel del objeto - Se reutilizan objetos disponibles en lugar de escribir el código nuevamente
  - Nivel de componente - Se reutilizan colecciones de objetos
  - Nivel del sistema - Se reutilizan los sistemas de aplicación enteros.
- Costos de pensar en reuso a futuro, costo de reusar algo.
- Beneficios del reuso

Reuso

Beneficio	Descripción
Mayor confianza	El software reutilizado ha sido probado y testeado en producción, debería mas confiable que el software nuevo.
Reduccion del riesgo del proceso	El costo del software existente es conocido, reduce el margen de error en la estimación de costo del proyecto.
El uso eficaz de los especialistas	Especialistas de aplicaciones se pueden desarrollar software reutilizable que encapsula su conocimiento.
Conformidad de los estándares	Algunos estándares tales como los estándares de interfaz de usuario pueden ser implementados como un conjunto de componentes reutilizables.
Desarrollo acelerado	Puede acelerar la velocidad con que se pone el sistema en producción porque el tiempo de desarrollo y validación puede ser reducido.

Reuso

Problema	Descripción
Aumento de los costos de mantenimiento	Si el código fuente de un sistema de software o un componente no esta disponible, los costos de mantenimiento pueden ser mas altos porque los elementos reutilizados en el sistema pueden llegar a ser cada vez mas incompatibles con los cambios del sistema.
"Crear, mantener y usar" o "encontrar, comprender y adaptar" componentes reutilizables	Desarrollar componentes reutilizable y asegurarse de que los desarrolladores de software puedan usar esta librería puede ser costoso
Síndrome de no inventado aquí	Algunos ingenieros de software prefieren reescribir los componentes porque creen que ellos lo pueden mejorar. Esto tiene que ver en parte con la confianza y en parte con el hecho de que escribir software original es visto como un desafío mayor que la reutilización de software de otros.

Aspectos importantes en el Diseño

- Concurrencia
- Control y manejo de eventos
- Persistencia de datos
- Distribución de componentes
- Manejo de excepciones y errores, tolerancia a fallas
- Interacción y presentación
- Seguridad

Diseño - un problema malvado (*wicked*)

- El proceso de diseño carece de cualquier forma analítica, con una consecuencia importante es que bien puede haber una serie de soluciones aceptables para cualquier problema. Debido a esto, el proceso de diseño rara vez será "convergente", en el sentido de poder dirigir al diseñador hacia una única solución preferida.
- Un problema "malvado" demuestra algunas propiedades interesantes.
- Se puede caracterizar como un problema cuya forma es tal que una solución para uno de sus aspectos simplemente cambia el problema.

Diseño - un problema malvado (*wicked*)

- No hay una formulación definitiva de un problema malvado.
- Los problemas malvados no tienen una regla de detención.
- Sus soluciones no son verdaderas o falsas, sino buenas o malas.
- No hay una prueba inmediata ni definitiva de una solución.
- Cada solución a un problema malvado es una "operación de una sola vez", porque no hay oportunidad de aprender por ensayo y error, cada intento cuenta de manera significativa.
- No tienen un conjunto enumerable (o exhaustivamente descriptible) de soluciones potenciales.
- Cada problema malvado es esencialmente único.
- Cada problema malvado puede considerarse como un síntoma de otro problema.