Modelo

Thursday, October 24, 2019 7:24 PM

① Requerimiento | ③ Diseño | ⑤ Pruebas
② Análisis | ④ Desarrollo | ⑥ Mantenimiento

- Se crean para entender mejor la entidad real que se va a construir
- 2 tipos de modelos:

① De requerimientos (Modelo de análisis) ← ¿A quién va dirigido?
- Información
- Funcional (Lo que quiere)
- Comportamiento (Cómo)

⟹ Prototipos

② De diseño
- Arquitectura
- Interfaz de Usuario
- Detalle en el nivel de comportamiento

• Improntar

Modelado de requerimientos

Cada método de análisis tiene un punto de vista único. Sin embargo, todos están relacionados por ciertos principios operacionales:

Principio 1. **Debe representarse y entenderse el dominio de información de un problema.** El dominio de información incluye:

- ✓ los datos que fluyen hacia el sistema (usuarios finales, otros sistemas o dispositivos externos)
- ✓ los datos que fluyen fuera del sistema (por la interfaz de usuario, interfaces de red, reportes, gráficas y otros medios)
- ✓ los almacenamientos de datos que recaban y organizan objetos persistentes de datos (por ejemplo, aquellos que se conservan en forma permanente)

Principio 2. **Deben definirse las funciones que realizará el software.** Las funciones del software dan un beneficio directo a los usuarios finales y también brindan apoyo interno para las características que son visibles para aquéllos.

Principio 2. **Deben definirse las funciones que realizará el software.** Las funciones del software dan un beneficio directo a los usuarios finales y también brindan apoyo interno para las características que son visibles para aquéllos.

✓ *Funciones que transforman* los datos que fluyen hacia el sistema.

✓ *Funciones que activan* algún nivel de control sobre el procesamiento interno del software o sobre los elementos externos del sistema.

Las funciones se describen en muchos y distintos niveles de abstracción, que van de un enunciado de propósito general a la descripción detallada de los elementos del procesamiento que deben invocarse.

Principio 3. **Debe representarse el comportamiento del software.** El comportamiento del software de computadora está determinado por su interacción con el ambiente externo. Las *entradas* que dan los usuarios finales, el control de los datos efectuado por un sistema externo o la vigilancia de datos reunidos en una red son el motivo por el que el software se comporta en una forma específica.

Principio 4. **Los modelos que representen información, función y comportamiento deben dividirse de manera que revelen los detalles en forma estratificada (o jerárquica).** El modelado de los requerimientos es el primer paso para resolver un problema de ingeniería de software.

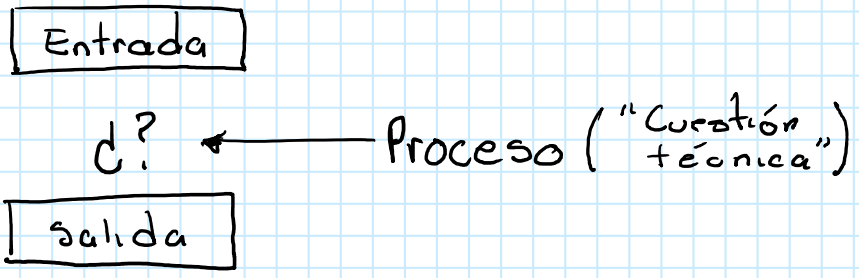
✓ *permite* entender mejor el problema

✓ *proporciona* una base para la solución (diseño).

Los problemas complejos son difíciles de resolver por completo, debiéndose usar la estrategia de divide y vencerás. Un problema grande y complejo se divide en sub-problemas hasta que cada uno de éstos sea relativamente fácil de entender: *partición o separación de entidades*.

Principio 5. **El trabajo de análisis debe avanzar de la información esencial hacia la implementación en detalle.** El modelado de requerimientos comienza con la descripción del problema desde la perspectiva del usuario final. Se describe la "esencia" del problema sin considerar la forma en la que se implementará la solución. La implementación detallada (normalmente descrita como parte del modelo del diseño) indica cómo se desarrollará la esencia.

Apartado 4



Actividad	Técnica o Prácticas	Herramientas	Productos
Definición del alcance del sistema	Entrevistas alto nivel	• Formato requisitos • Herramienta CASE	• Requisito a alto nivel
Identificación de requisitos	• Entrevistas detalladas • Análisis de procesos asincrónos	• Formato de requisitos • Herramientas CASE	• Catálogo de procesos asincrónicos • Catálogo de requisitos a nivel detalle
Elaboración	Modelo de	• Herramientas	• Espec. Sistemas

Elaboración Modelo de comportamiento del sistema	Modelo de caso de uso	Herramientas CASE	• Espec. Sistemas • Notación a alto nivel • Notación detallada • Diagrama de Ca. Uso.

Requerimiento → Requisitos

↳ Abstracto

↳ sentimientos

Requisitos funcionales y no funcionales.

Los requisitos se clasifican en funcionales y no funcionales.

✓ Los requisitos funcionales describen los servicios que el sistema debe suministrar y de las restricciones en su operación e implementación.

- *Modelos de caso de uso.* El caso de uso es la narración textual de la secuencia de eventos de un actor al utilizar un proceso de principio a fin que genera.

1. Identificación de procesos asíncronos
2. Descripción textual del caso de uso
3. Descripción gráfica del caso de uso
   - Asociaciones de tipo <<include>> y <<extend>>
     - Una asociación <<include>> representa una asociación de dependencia entre un caso de uso influyente (verificar cuenta) y otro dependiente (retirar dinero).
     - Una asociación <<extend>> representa un conjunto de actividades que se ejecutan de forma opcional y constituye un escenario alternativo
4. Procesos CRUD (Crear, leer, actualizar y borrar) y el uso de la asociación <<include>>

Requisitos funcionales y no funcionales.

Los requisitos se clasifican en funcionales y no funcionales.

✓ Los requisitos no funcionales se refieren a propiedades del sistema. Entre las propiedades más referenciadas:

- la presentación de la información
- la frabilidad
- tiempos de respuestas
- accesorios o dispositivos de entrada o salida
- aspectos de seguridad de acceso
- respaldo a los datos

Requisitos → ¿Necesario? → ¿Qué sería lo peor de no incluir?

→ Factible (Realista)

- - - - → Modificable → ¿Qué tan fácil es hacer cambios al requerimiento?

- - - - → Priorizado → Grado de necesidad del mismo esencial/crítico, deseado, opcional

- - - - → Verificable → ¿Cómo se sabe que si se cumplió con él? (Criterio de aceptación)

- - - → Rastreable → Cada función del sistema se puede rastrear

- - - - → Claro → Fácil de leer y entender para un futuro

Modelos: Requerimientos vs Diseño

El modelo de requerimientos describe el dominio de información del problema, las funciones visibles para el usuario, el comportamiento del sistema y un conjunto de clases de requerimientos que agrupa los objetos del negocio con los métodos que les dan servicio.

El modelo de diseño traduce esta información en una arquitectura, un conjunto de subsistemas que implementan las funciones principales y un conjunto de componentes que son la realización de las clases de requerimientos.

Requerimientos vs Diseño

Tuesday, October 29, 2019 8:31 PM

Modelos: Requerimientos vs Diseño

El modelo de requerimientos describe el dominio de información del problema, las funciones visibles para el usuario, el comportamiento del sistema y un conjunto de clases de requerimientos que agrupa los objetos del negocio con los métodos que les dan servicio.

El modelo de diseño traduce esta información en una arquitectura, un conjunto de subsistemas que implementan las funciones principales y un conjunto de componentes que son la realización de las clases de requerimientos.

Modelado de diseño

Es análogo a los planos arquitectónicos de una casa. Se comienza por representar la totalidad de lo que se va a construir (por ejemplo, un croquis tridimensional de la casa) que se refina poco a poco para que guíe la construcción de cada detalle (por ejemplo, la distribución de la plomeria).

Principio 1. El diseño debe poderse rastrear hasta el modelo de requerimientos. Los elementos del modelo de diseño deben poder rastrearse en el modelo de requerimientos.

Modelado de diseño

Principio 2. Siempre tomar en cuenta la arquitectura del sistema que se va a construir. El diseño debe comenzar con consideraciones de la arquitectura, entendiéndola como el esqueleto del sistema que se va a construir. Sólo después de establecida ésta deben considerarse los aspectos en el nivel de los componentes. Afecta:

- interfaces
- estructuras de datos
- flujo de control
- comportamiento del programa
- manera en la que se realizarán las pruebas
- la susceptibilidad del sistema resultante a recibir mantenimiento Por todas estas razones,

También la forma en que se probará

Desarrollar para mantener

Principio 3. El diseño de los datos es tan importante como el de las funciones de procesamiento. El diseño de los datos es un elemento esencial del diseño de la arquitectura. Un diseño de datos bien estructurado ayuda a simplificar el flujo del programa, hace más fácil el diseño e implementación de componentes de software y más eficiente el procesamiento conjunto.

↳ UTF8 / UNICODE / ETC...

Modelado de diseño

- **Principio 4. Las interfaces (tanto internas como externas) deben diseñarse con cuidado.** La manera en la que los datos fluyen entre los componentes de un sistema tiene mucho que ver con la eficiencia del procesamiento, la propagación del error y la simplicidad del diseño.

- **Principio 5. El diseño de la interfaz de usuario debe ajustarse a las necesidades del usuario final.** "La interfaz de usuario es la manifestación visible del software." No importa cuán sofisticadas sean sus funciones internas, ni lo incluyentes que sean sus estructuras de datos, ni lo bien diseñada que esté su arquitectura, un mal diseño de la interfaz con frecuencia conduce a la percepción de que el software es "malo".

Check this

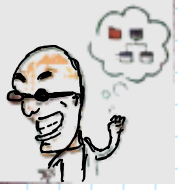
Modelado de diseño

Principio 6. El diseño en el nivel de componentes debe tener independencia funcional. La independencia funcional es una medida de la "mentalidad única" de un componente de software. La funcionalidad que entrega un componente debe ser cohesiva, es decir, debe centrarse en una y sólo una función o sub-función.

Principio 7. Los componentes deben estar acoplados con holgura entre sí y con el ambiente externo. A medida que se incrementa el nivel de acoplamiento, también aumenta la probabilidad de propagación del error y disminuye la facilidad general de dar mantenimiento al software. Entonces, el acoplamiento de componentes debe mantenerse tan bajo como sea razonable.

Tipos de acoplamiento:

- con mensajería
- datos globales



Modelado de diseño

- **Principio 8. Las representaciones del diseño (modelos) deben entenderse con facilidad.** El propósito del diseño es comunicar información a los profesionales que generarán el código, a los que probarán el software y a otros que le darán mantenimiento en el futuro. Si el diseño es difícil de entender, no servirá como medio de comunicación eficaz.
- **Principio 9. El diseño debe desarrollarse en forma iterativa.** El diseñador debe buscar más sencillez en cada iteración. Igual que ocurre con casi todas las actividades creativas, el diseño ocurre de manera iterativa. Las primeras iteraciones sirven para mejorar el diseño y corregir errores, pero las posteriores deben buscar un diseño tan sencillo como sea posible.

Modelado de diseño

- **Principio 8. Las representaciones del diseño (modelos) deben entenderse con facilidad.** El propósito del diseño es comunicar información a los profesionales que generarán el código, a los que probarán el software y a otros que le darán mantenimiento en el futuro. Si el diseño es difícil de entender, no servirá como medio de comunicación eficaz.
- **Principio 9. El diseño debe desarrollarse en forma iterativa.** El diseñador debe buscar más sencillez en cada iteración. Igual que ocurre con casi todas las actividades creativas, el diseño ocurre de manera iterativa. Las primeras iteraciones sirven para mejorar el diseño y corregir errores, pero las posteriores deben buscar un diseño tan sencillo como sea posible.

CODIFICACIÓN Y PRUEBAS

Tuesday, 19 November 2019 19:22

Codificación y pruebas

Hace referencia a la construcción correcta de los elementos que se han diseñado

Incluye las tareas de

- Generación del código: codificación en lenguaje fuente (ya sea manualmente o automáticamente), compilación, enlazado, e instalación en el entorno de ejecución.
- La prueba de que el código opera correctamente

Los principales productos a que da lugar la fase de traducción y elaboración son:

- El codigo fuente de los elementos definidos en el diseño.
- Un plan de prueba , de las asociaciones, de las operaciones y de las interacciones.
- Un informe sobre los módulos elaborados y su forma de utilización.
- Los componentes de software compilados.

Codificación

- ✓ El código fuente de cada programa realizado para el sistema, incluyendo nombre de archivo, función que realiza, autor y fecha de creación.
- ✓ Diccionario de datos, donde se incluya la descripción de cada estructura de las tablas o bases de datos que interactuarán con el sistema. Debe llevar, nombre del archivo, fecha de creación, nombre de campo, tipo de campo, longitud del campo, relaciones con otras bases de datos y nombrar los campos clave.
- ✓ Si la aplicación va a tener interacciones con otros sistemas.
- ✓ Para seleccionar la plataforma para el desarrollo de la aplicación debemos tomar en cuenta las funciones que se van a realizar, equipo con el que contamos, sistema operativo, conectividad con la que se cuenta, plataformas de datos con las que cuentan los sistemas actuales (en el dado caso que la aplicación vaya a interactuar con otros sistemas)
  - ✓ tomar en cuenta las bondades que ofrece el lenguaje de programación
    - en cuanto a manejo de datos
    - capacidad de ejecución de los programas (tiempo de respuesta para los usuarios)

Malas prácticas

1. No asegurarse de que los parámetros recibidos son los que esperamos.
2. No comprobar si un elemento tiene valor antes de acceder a él.
3. No asegurarse de que un array o colección tenga valor y contenga elementos antes de acceder a ellos.
4. No comentar. (Comentar el código también es programar)
5. No comprobar si el índice del elemento existe antes de acceder a él en una colección indizada.
6. División por 0 / NaN.
7. Convertir cadenas a números directamente.
8. No liberar los recursos adecuadamente.
9. El método recursivo infinito.
10. Utilizar incorrectamente técnicas conocidas.

- **Inspección** : Determina si el código esta completo y correcto, como también las especificaciones.
- **Walkthrough**: Interrelación informal entre testers, creadores y usuarios del sistema
- **Verificación estática**: Compara los valores generados por el programa con los rangos de valores predefinidos haciendo una descripción del funcionamiento de los procedimientos en términos
- **Ejecución simbólica**: Hace un seguimiento de la comunicación entre funciones, módulos, aplicaciones, luego de que todas las partes hayan sido verificadas por separado

↳ Pruebas de escritorio

Buenos hábitos - KISS

Acrónimo de "**K**eep **I**t **S**hort and **S**imple".

- **Particionado.** Se basa en el principio de dividir secciones grandes de código en partes más pequeñas, lo que hace cada una de las secciones más comprensible.
- **Simplificando.** Se basa en el principio de hacer el código más comprensible eliminando todo aquello que no sea útil o modificándolo para hacerlo más sencillo.

Buenos hábitos - YAGNI

Acrónimo de **Y**ou **A**in't **G**onna **N**eed **I**t

- ✓ Uno de los fallos comunes en un proyecto de software es el exceso de programación u over-programming
- ✓ Cuando afrontamos cualquier proyecto uno de los primeros pasos es realizar un análisis.
  - ✓ Realizar el análisis de los requisitos del proyecto sin tener en mente el objetivo final, produce siempre como resultado un exceso de funcionalidad.
  - ✓ ...tener en mente el objetivo final

"10 minutos despues de poner de redactar ésta diapositiva y su título, todavia seguia buscando la mejor redacción"

DRY - Don't repeat Yourself

- Cuando se aplica de forma eficiente, los cambios necesarios sólo se necesita hacerlos en un sólo lugar.

Costes de errores

Costes de errores

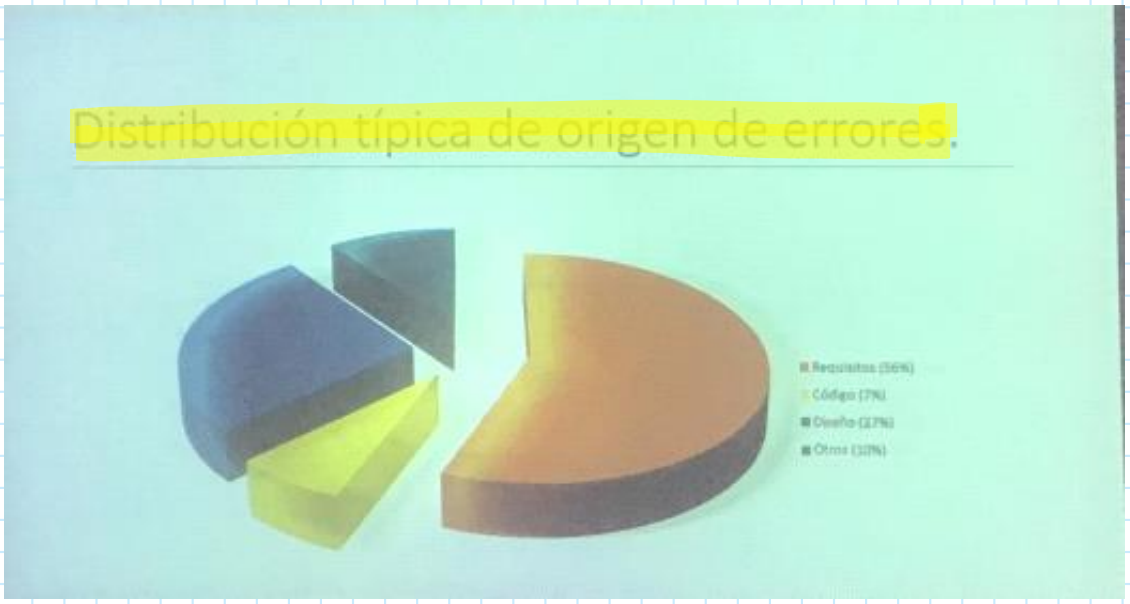
- Daños reales
- Tiempo
- Imagen y confianza
- Motivación

} Errores

- Encontrar errores, acotando situaciones en donde algo pasa cuando no debe de pasar y viceversa.

Errores

Thursday, 21 November 2019 7:54 PM



- Error $\triangleq$ Es la causa raíz
- Defecto $\triangleq$ Es la consecuencia del error
- Falla $\triangleq$ Es la parte visual del error, lo que ve el usuario

Tipos de fallos.

Fallos iniciales. Esta etapa se caracteriza por tener una elevada tasa de fallos que desciende rápidamente con el tiempo. Estos fallos pueden deberse a diferentes razones como equipos defectuosos, instalaciones incorrectas, errores de diseño del equipo, desconocimiento del equipo por parte de los operarios o desconocimiento del procedimiento adecuado.

Fallos normales. Etapa con una tasa de errores menor y constante. Los fallos no se producen debido a causas inherentes al equipo, sino por causas aleatorias externas. Estas causas pueden ser accidentes fortuitos, mala operación, condiciones inadecuadas u otros.

Fallos de desgaste. Etapa caracterizada por una tasa de errores rápidamente creciente. Los fallos se producen por desgaste natural del equipo debido al transcurso del tiempo

Errores de programación

- Errores en tiempo de diseño
  1. Compilación
- Errores en ejecución
  1. Lógica
  2. Procesamiento

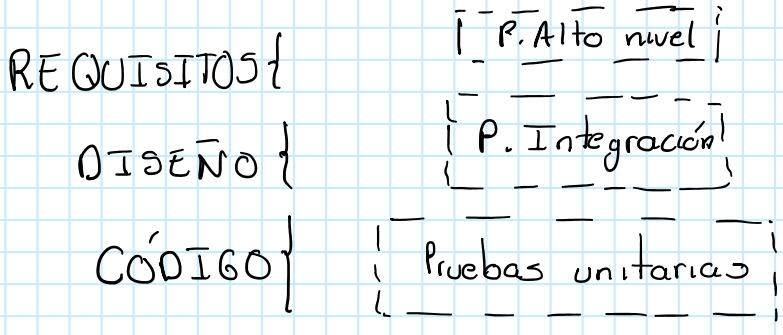
Tipos de prueba

Las pruebas unitarias o de componente, se implementan habitualmente por el equipo de desarrollo, consisten en la ejecución de actividades que le permitan verificar al desarrollador que los componentes unitarios están codificados soportando el ingreso de datos erróneos o inesperados y demostrando así la capacidad de tratar errores de manera controlada, también con la finalidad de poder identificar estos errores y así buscarles una solución.

La prueba de Integración, tiene como objetivo presentar el software de manera completa ya que se lo puede realizar por módulos.

Las pruebas del sistema, tienen como propósito ejercitar profundamente el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema (hardware, otro software, etc.) y que realizan las funciones adecuadas.

Las pruebas de aceptación, es cuando el producto esta idóneo para ser utilizado por los cliente, por lo tanto no debe de presentar ningún tipo de inconveniente en el entorno donde tenga que ser implementado.



Métodos básicos

Caja negra (o funcionales), **no se conoce ni su contenido ni su estructura interna.** ***Si bien no conocemos la estructura interna del sistema, sabemos cómo se espera que se comporte.***

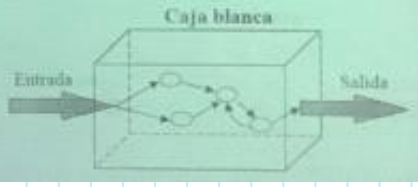
Para la ejecución de pruebas de caja negra:

- ✓ se suministran datos de entrada al sistema
- ✓ se observa la salida obtenida y se compara con la salida esperada.

Métodos básicos

Caja blanca (o estructurales), **se piensan y diseñan casos de prueba conociendo el código del software.** Se seleccionan los caminos del programa y el flujo de datos a ejercitar durante las pruebas.

Es importante destacar que las pruebas de caja blanca están enfocadas a identificar debilidades en el diseño del código y no a detectar discrepancias entre los requerimientos y su implementación.



Métodos básicos

Pruebas de regresión, verifican que no ocurrió una regresión en la calidad del producto luego de un cambio.

Implican la re ejecución de alguna o todas las pruebas ejecutadas anteriormente.

No es suficiente con probar la modificación solamente, porque ésta pudo haber tenido un impacto en otras "zonas" del producto.

Caso de prueba

Está dado por un conjunto de entradas, condiciones de ejecución y las salidas esperadas, permite revelar fallas.

- ✓ **Planeación,** fija las metas y una estrategia general de pruebas
- ✓ **Preparación,** se describe el procedimiento general de pruebas y se generan los casos de prueba específicos
- ✓ **Ejecución,** incluye la observación y medición del comportamiento del producto
- ✓ **Análisis,** incluye verificación y análisis de resultados para determinar si se observaron fallas
- ✓ **Seguimiento,** si se detectaron fallas, se inicia un monitoreo para asegurar que se remueva el origen de éstas