

Toda la materia en un documento, se omiten detalles

Tabla de contenido

Clase 1 – SRS.....	4
IEEE 830 como estándar.....	4
Un buen SRS es	5
Consideraciones.....	5
Clase 2 – Riesgos.....	5
Tipos de riesgo.....	5
Dentro de los tipos de riesgo	6
Gestión de Riesgos.....	6
Clase 3 – Parte 1, GCS	7
Linea base de GCS (Gestión de la Configuración de Software)	7
Proceso del GCS en 5 partes.....	7
Clase 3 – Parte 2, Planificación Organizativa	8
Participantes de la planificación organizativa.....	8
Líder de Equipo	8
Organigramas de equipos	8
Tres tipos de organigramas.....	8
Clase 4 – Planificación Temporal.....	9
Calendarización.....	9
Red de tareas de una calendarización	10
Hay 3 métodos vistos en teoría sobre planificaciones temporales	10
PERT.....	10
CPM	10
Método del camino crítico (PERT-CPM).....	10
Clase 5 – Parte 1, Métricas.....	10
Métricas del Proyecto	11
Métricas del proceso	11
Métricas del producto.....	11
Métricas postmortem	11
Líneas de Código como Métrica de producto	11

Toda la materia en un documento, se omiten detalles

GQM (Goal, Question, Metric) para medir objetivos y mejorar calidad de proyecto	11
Clase 5 – Parte 2, Estimaciones	12
Estimación de recursos	12
Estimación de costos.....	12
Tres técnicas de estimación	12
Modelos empíricos	13
COCOMO II – Como modelo empírico	13
COCOMO II – Modelo de composición de aplicación.....	13
COCOMO II – Modelo de diseño temprano	13
COCOMO II – Modelo de reutilización	13
COCOMO II – Modelo de post-arquitectura	13
Clase 6 – Conceptos de Diseño de Software.....	13
Distintos tipos de diseño.....	13
Evaluar un diseño.....	14
Conceptos de Diseño como tales	14
Abstracción	14
Arquitectura.....	14
Patrones.....	14
Modularidad	15
Ocultamiento de información	15
Independencia funcional.....	15
Refinamiento	15
Refactoring	15
Clase 7 – Diseño de interfaz (UI) y algo de UX	15
6 principios del diseño de UI	16
User Design Experience (Udx)	16
Reglas Doradas del Diseño	16
Clase 8 – Parte 1, Diseño Arquitectónico	16
La arquitectura afecta a requisitos no funcionales	17
Sobre organización del sistema.....	17
Patrones de Arquitectura.....	17

Patrón Repositorio	17
Patrón Cliente-Servidor	18
Patrón de arquitectura en capas	18
Descomposición Modular	18
Estrategias de descomposición	18
Modelo de control	18
Dos formas de control.....	18
Sistemas dirigidos por eventos.....	19
Arquitectura de sistemas distribuidos.....	19
Cuatro grandes arquitecturas distribuidas	19
Multiprocesador	19
Cliente-servidor	19
De componentes distribuidos	20
Distribuida inter-organizacional	20
Clase 8 – Parte 2 , Codificación SOLO CONCEPTOS RELEVANTES	20
Pautas generales de código.....	20
Documentación.....	21
Clase 9 – Estrategias de Pruebas	21
Verificar y Validar.....	21
Tipos de prueba convencionales	21
Prueba de unidad.....	21
Prueba de integración	22
Prueba de regresión.....	22
Prueba del sistema.....	22
Pruebas de validación	23
Pruebas de aceptación – Alfa y Beta	23
Aceptación ALFA	23
Aceptación BETA	23
Depuración	23
Clase 10 – Tipos de Prueba	23
Prueba del Software	23

Principios de la prueba.....	24
Tipos de prueba	24
Caja Blanca.....	24
Caja Negra.....	24
Clase 11 – Mantenimiento y Auditoria	24
Mantenimiento.....	24
Características de mantenimiento.....	25
Problemática de mantener	25
Ciclo de mantenimiento	25
Tareas en el desarrollo que ayudan al mantenimiento.....	25
Tipos de mantenimiento	25
Rejuvenecimiento del Software	25
Auditoria informática	26
Objetivos de la auditoria	26

Clase 1 – SRS

El documento SRS(ERS) normalizado en el estándar IEEE 830. Es la especificación de las funciones que realiza un producto de software en un determinado entorno. Puede ser desarrollado por personal que representa a los desarrolladores o de la parte cliente, es aconsejable que ambas partes intervengan.

IEEE 830 como estándar

- Alcance del documento: brindar colección de buenas prácticas para escribir especificaciones de requerimientos de software, mediante el alcance del estándar se describen contenidos y cualidades de un buen SRS.
- Naturaleza del documento: especificación para un producto de software en particular.
- Ambiente del SRS: si el sistema es parte de otro, se hará un SRS que declarará interfaces entre el sistema y el sistema que desarrollamos.

Un buen SRS es

- Correcto: cada requisito declarado está en el software.
- No ambiguo: cada requisito tiene una sola interpretación.
- Completo: se reconocen todos los requisitos externos impuestos por la especificación del sistema.
- Consistente: el SRS debe estar de acuerdo con documentos de nivel superior (una especificación de requerimientos por ejemplo).
- Priorizado.
- Comprobable: cada requisito declarado es comprobable. Un requisito es comprobable si una persona o máquina puede verificar que el producto cumple el requisito. Un requisito ambiguo no es comprobable.
- Modificable: si permite cambios de requisitos de forma fácil, completa y consistente conservando su estructura y estilo.
- Trazable.

Consideraciones

- Prepararlo conjuntamente entre ambas partes.
- Hacerlo evolutivo en conjunto con el software, registrando cambios, responsables y la aceptación de estos.
- Usar prototipos para definir requisitos.
- Los detalles particulares se anexan como documentos externos (CU, plan de proyecto, plan de aseguramiento de calidad, etc).

Clase 2 – Riesgos

Un riesgo es un evento no deseado con consecuencias negativas para el proyecto. Se deben hacer planes para evitar los riesgos o si ocurren, minimizar las consecuencias.

Un plan reactivo es cuando el problema ya ocurrió y tenemos que gestionar dicho problema para minimizar las consecuencias.

Un plan proactivo es cuando ya tenemos estrategias de tratamiento para posibles riesgos.

La probabilidad de ocurrencia de un riesgo es llamada incertidumbre, la pérdida hace referencia a aquello que se perdió cuando ocurrió el riesgo.



Tipos de riesgo

- De proyecto: amenazan al plan de proyecto y al desarrollo del mismo -> falta de presupuesto, mala elección de personal, falta de recursos, etc.
- De producto: afectan a la calidad o rendimiento del sistema terminado que entregamos.
- De negocio: afectan a la organización que crea y distribuye el software.

Dentro de los tipos de riesgo

- Riesgo genérico: pueden aparecer en cualquier tipo de proyecto.
- Riesgo específico: pueden aparecer pero son específicos al dominio del proyecto.
- Podemos seguir subdividiendo:
 - Riesgo conocido: en base a experiencia anterior se pueden determinar.
 - Predecibles: aquellos que siempre pasan en los proyectos.
 - Impredecibles: no son previsibles, pueden ocurrir porque nunca pasaron o por que no los conocemos

Gestión de Riesgos

- Tarea que consiste en 4 etapas en orden
 - Identificación de riesgos: se identifican mediante un brainstorming, se seleccionan los verdaderos riesgos que puedan ocurrir (para evitar listas largas) y se categorizan esos riesgos seleccionados.
 - Análisis del riesgo: en la etapa anterior tenemos los riesgos con la probabilidad de que ocurran y cual será su impacto si ocurren.
 - Se establece una tabla que menciona cada riesgo, la categoría de este, su probabilidad de ocurrencia y el impacto que provocaría.
 - Se toman en cuenta: riesgos de gran impacto con probabilidad moderada a alta y riesgos de poco impacto pero con gran probabilidad.
 - Entonces la probabilidad es lo importante.
 - Planeamiento: se elige estrategias para tratar los riesgos que quedan como prioridad inicial.
 - Evitar el riesgo -> consiste en diseñar el sistema de forma que el riesgo no aparezca.
 - Minimizar el riesgo -> se buscan estrategias para que se reduzca la probabilidad de un riesgo.
 - Plan de contingencia -> para riesgos que van a aparecer, se tratan de forma que se minimice su impacto.
 - Supervisión: se supervisa a lo largo del tiempo.
 - Si cambiaron las probabilidades, que tan efectivas son las estrategias, se detectan ocurrencias de riesgos previstos, se asegura que se cumplen los pasos definidos para cada riesgo, se recopila información a futuro, se determina si surgen nuevos riesgos, se reevalúan periódicamente los que ya existe.

Clase 3 – Parte 1, GCS - Gestión de la Configuración del Software

Cuando se realiza un software pueden ocurrir cambios en este, por lo tanto debemos saber controlar e informar sobre lo que pueden implicar dichos cambios.

Un elemento ECS es un elemento que debemos controlar y gestionar, son propensos al cambio y para esto usamos GCS.

GCS es el proceso de identificación y definición de elementos del sistema, de forma que nos permite controlar el cambio de dichos elementos a lo largo de su ciclo de vida, para registrar y reportar el estado de cada elemento y las solicitudes de cambios, de forma que nos permitirá verificar que los elementos sean completos y correctos.

Los cambios ocurren en cualquier momento, GCS nos ayuda para identificarlo, controlarlo y garantizar su correcta implementación.

Línea base de GCS (Gestión de la Configuración de Software)

ECS = Elementos de la gestión de Configuración de Software. (me habría gustado más EGCS)

Cuando un cambio se realiza con éxito y se arreglan posibles errores se crea una línea base que debe ser aceptada/aprobada tanto por el equipo de desarrollo como por el usuario, de forma que se genera una línea base en la cual ambas partes están de acuerdo con lo que expresa el documento.

Cuando un ECS pasa a ser una línea base, todos los cambios realizados en ese documento deben ser controlados y gestionados por medio de la GCS.

Proceso del GCS en 5 partes

1. **Identificar:** cada ECS es identificado con una cadena de caracteres no ambigua junto a una descripción.
2. **Control de versiones:** cada versión de los ECS es gestionada.
3. **Control de cambios:** se combinan procedimientos humanos y herramientas para controlar cambios. Una autoridad de control de cambios (ACC) nos dirá como impactará el cambio en el hardware, rendimiento, como alterará la percepción del cliente y como afectará la calidad y fiabilidad.
4. **Auditoría de la configuración:** Se realizan revisiones técnicas formales que verifican la correctitud del cambio.
5. **Generación de informe de estado de la configuración:** nos indica que pasó, quién hizo el cambio, cuando lo hizo, que más se vió afectado. Generar estos informes es vital para el éxito del proyecto.

Clase 3 – Parte 2, Planificación Organizativa

El personal que trabaja en una organización de software es el activo más grande e importante de una empresa y una mala administración de dicho personal es un factor principal de fracaso en proyectos.

Participantes de la planificación organizativa

- Gerentes ejecutivos: definen temas empresariales, aspectos de negocio del proyecto.
- Gerentes de proyecto: planifican, motivan, organizan y controlan a los profesionales.
- Profesionales: aportan las habilidades técnicas.
- Clientes: dan requerimientos.
- Usuarios finales: interactúan con el software.

Líder de Equipo

El líder de equipo organizará el equipo de software de forma que se maximicen las habilidades y capacidades de cada persona.

MOI de Liderazgo:

- **Motivación al personal** para sacar lo mejor.
- **Organización del equipo.**
- **Incentivación de ideas** para alentar a las personas.

Rasgos de un buen líder:

- **Resolución de problemas:** capacidad de diagnóstico de conflictos relevantes, estructurar soluciones, motivar a otros profesionales, aplicar lecciones aprendidas y flexibilidad si los intentos de resolución son malos.
- **Identidad administrativa:** es confiado para asumir el control de ser necesario, ayuda al personal técnico a seguir sus instintos.
- **Logro:** recompensará la iniciativa y logros de su equipo, para mantener un equipo optimizado y productivo, no debe castigar a nadie por correr riesgos controlados.
- **Influencia y construcción de equipo:** capaz de leer a la gente, conocer señales verbales y no verbales, proporcionar reacción ante ellas. Permanecer bajo control en situaciones de estrés.

Organigramas de equipos


Comunicación vertical: con gente de jerarquía superior o inferior.

Comunicación horizontal: con gente de misma jerarquía.


Tres tipos de organigramas

- **Descentralizado democrático** (para problemas difíciles y/o complejos, para equipos que estarán juntos mucho tiempo)

Para el cual no existe un jefe permanente, sino que hay coordinadores de tareas a corto plazo, las decisiones se toman por consenso y la comunicación es horizontal.

 **Descentralizado controlado** (para problemas complejos, proyectos muy grandes)

Hay un jefe definido que coordina tareas y jefes secundarios para subtareas, la implementación de soluciones se reparte en subgrupos. La comunicación es vertical/horizontal.

 **Centralizado controlado** (para tareas rápidas y problemas sencillos, proyectos muy grandes)

El jefe del equipo se encarga de resolver problemas a alto nivel y de coordinar internamente al equipo, la comunicación entre jefe-miembros de equipo es vertical.

La elección del organigrama depende del proyecto, su tamaño, tiempo para desarrollar, cantidad de personas en el equipo, etc.

Clase 4 – Planificación Temporal

La planificación temporal es el cómo vamos a planificar nuestro proyecto a lo largo del tiempo. Es una actividad que distribuye el esfuerzo estimado durante la duración prevista del proyecto, una mala estimación genera problemas de entregas y subida de costos.

Calendarización

Se compone de...

Tareas: secuencia de acciones que se realizarán en un plazo determinado de tiempo.

Tienen parámetros:

- **Precursor:** evento o conjunto de eventos que deben ocurrir antes de empezar la tarea.
- **Duración:** cuanto tiempo nos tomará completarla.
- **Fecha de entrega:** para la cual la actividad ya debe estar completa.
- **Resultado:** hito o componente listo.

Tarea crítica: el retraso de estas tareas generan retrasos generales en todo el proyecto.

Hito: algo que se espera ya completo para una fecha determinada.

Red de tareas de una calendarización

- Es una representación gráfica del flujo de las tareas desde el inicio hasta el fin del proyecto.
- Muestra la secuencia de tareas.
- Si hay actividades que se puedan realizar en paralelo, se deja visto.
- Muestra interdependencia que pueda existir entre tareas.

Hay 3 métodos vistos en teoría sobre planificaciones temporales

PERT

- Utilizado para controlar la ejecución de proyectos con un gran número de tareas que implican investigación, desarrollo y pruebas.
- Es un método probabilístico ya que la red de tareas es implementada con tiempos más probables que pueden ser optimistas o pesimistas.

CPM

- Utilizado en proyectos donde hay poca incertidumbre en estimaciones (más certezas, sirve para proyectos más pequeños).
- Inicio temprano y tardío.
- Es un método determinístico.

Método del camino crítico (PERT-CPM)

- Toma lo mejor de los dos anteriores.
- Se siguen pasos:
 1. Establecer la lista de tareas.
 2. Fijar dependencia entre tareas y duración de las tareas.
 3. Construir la red de tareas.
 4. Numerar los nodos (nodo = tarea).
 5. Se calcula la fecha temprana y tardía de cada nodo.
 - a. TE_i = Fecha temprana del nodo i.
 - b. TA_i = Fecha tardía del nodo i.
 6. Se calcula el camino crítico que une las tareas críticas.
 - a. $TE_i - TA_i$

Clase 5 – Parte 1, Métricas

Una métrica nos permite realizar una medición para obtener como resultado una indicación cuantitativa de un algo (una medida). También tenemos las combinaciones de métricas llamadas indicadores, que nos definen si estamos bien encaminados en un proceso o no.

Las métricas se usan por profesionales e investigadores para tomar mejores decisiones, a nivel software podemos dividirlos en 3.

Métricas del Proyecto

Usada para: ajustar el calendario, valorar el estado de un proyecto, rastrear riesgos, descubrir áreas problemáticas, ajustar el flujo de tareas, evaluar la habilidad del equipo.

Métricas del proceso

La finalidad es dar un conjunto de indicadores para mejorar el proceso, recopilando información a través de los proyectos. Usada para retroalimentaciones, para establecer metas claras, para dar al grupo de trabajo un sentido común y una sensibilidad organizacional.

Métricas del producto

Divididas en dos:

- **Dinámicas:** de forma que se hacen en un programa en ejecución para valorar eficiencia y fiabilidad de un programa.
- **Estáticas:** se hacen sobre representaciones del sistema para valorar complejidad, comprensibilidad y mantenibilidad.

Métricas postmortem

Usadas cuando el producto está terminado. La utilidad de estas métricas es conformar una línea base para futuras métricas. Dicha línea base la otorga el primer proyecto exitoso del grupo. Estas métricas postmortem ayudan al mantenimiento del sistema conociendo complejidad lógica, tamaño, flujo de información, identificación de módulos críticos y ayuda en procesos de reingeniería.

Líneas de Código como Métrica de producto

Es la métrica más común para el tamaño de un producto, es una métrica postmortem (se aplica sobre el proyecto terminado ya que se sabe cuantas líneas de código tiene este).

Nos da el tiempo usado y las personas involucradas. Esta métrica es aplicable y calculable si la organización de software mantiene registros de proyectos anteriores similares.

GQM (Goal, Question, Metric) para medir objetivos y mejorar calidad de proyecto

El método GQM tiene la siguiente estructura:

Conceptual (Goal):

- Se define un objetivo.

Operativo (Question):



- Se hace un conjunto de preguntas para ese objetivo, para verificar cumplimiento.

Cuantitativo (Metrica):

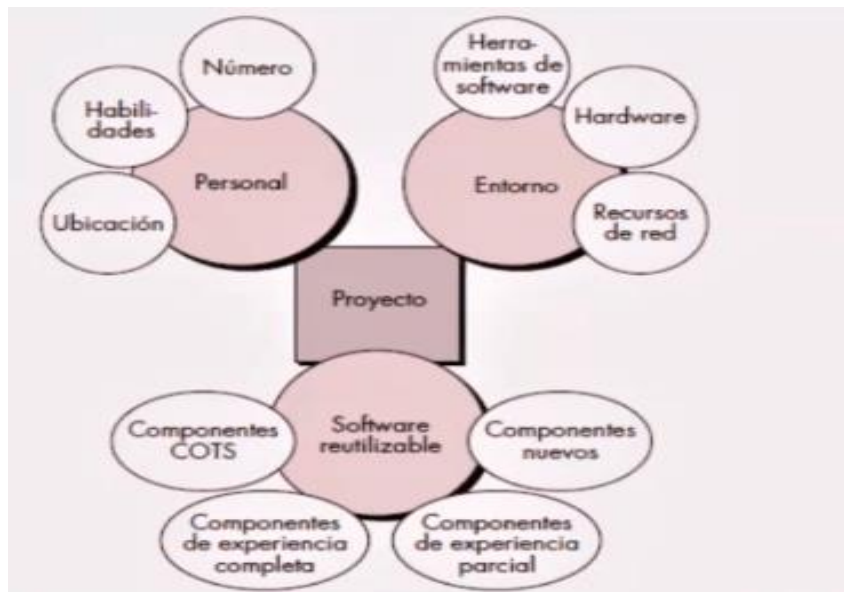
- Un conjunto de métricas se asocia a cada pregunta, para responder a cada pregunta de un modo cuantitativo.

Clase 5 – Parte 2, Estimaciones

Las estimaciones son técnicas que se pueden aplicar para dar valor aproximado a recursos necesarios, costos y tiempos del proyecto. Surge cuando deseamos medir algo pero este algo no nos dará un valor exacto.

Se usan con formulas de estimación y a tener en cuenta es que una estimación puede ser o no certera, por lo tanto se necesita experiencia para estimar. La complejidad, tamaño de producto y estructura de proyecto son cruciales para las estimaciones.

Estimación de recursos



Estimación de costos

Usa tres parámetros para calcular el costo (lo que nos cuesta) del proyecto. Cuanto se le paga a los developers (esfuerzo). Mantenimiento, costo de la plataforma y costo de internet (hardware y software). Los viajes que se puedan realizar.

Para fijar el precio que pagará el cliente debemos pensar en los intereses de la empresa, que riesgos implica el proyecto y el tipo de contrato.

Tres técnicas de estimación

Juicio experto: consiste en juntar varios expertos para que estimen, comparen y discutan.

Técnica Delphi: se consultará de forma independiente y anónima a varios expertos hasta que se llegue a un consenso.

División de trabajo: se consulta en una jerarquía que va de abajo hacia arriba.

Modelos empíricos

Utilizan fórmulas derivadas empíricamente para predecir costos o esfuerzos requeridos

COCOMO II – Como modelo empírico

COCOMO II – Modelo de composición de aplicación

Modela el esfuerzo que se requiere para desarrollar sistemas que se crean a partir de componentes reutilizables y proyectos de creación de prototipos. Basado en estimar puntos de aplicación (numero de pantallas desplegadas, numero de informes, numero de módulos del código, etc).

COCOMO II – Modelo de diseño temprano

Tiene como meta elaborar una estimación rápida y aproximada de los costos de forma que se puede usar en las primeras etapas de un proyecto para el cual todavía no está disponible el diseño detallado del sistema. Este modelo supone que se acordaron los requisitos del usuario y que está en marcha la etapa inicial de diseñado del sistema.

COCOMO II – Modelo de reutilización

Se usa para estimar el esfuerzo que requiere integrar código reutilizado o generado de forma automática (para este último se estima el número de personas que se necesitan por más para integrar el código).

COCOMO II – Modelo de post-arquitectura

Se usa cuando se encuentra disponible un diseño arquitectónico inicial para el cual podremos estimar cada parte del sistema.

Clase 6 – Conceptos de Diseño de Software

Un diseño de software es una representación de algo que será construido, donde los requisitos, necesidades y consideraciones técnicas se unen. También es la primer actividad técnica después de la etapa de análisis.

Distintos tipos de diseño

De datos

El modelo del dominio se transforma en estructuras de datos, objetos de datos, relaciones, etc.

Arquitectónico

Se eligen relaciones entre los elementos más importantes del software, se establecen estilos de arquitectura a usar, se eligen patrones de diseño, etc.

A nivel de componentes

Los elementos de la arquitectura se transforman en una descripción procedimental de los componentes del software.

De interface

La forma de comunicación dentro del sistema, con otros sistemas y con las personas.

Evaluar un diseño

Tiene que cumplir ciertos puntos

- Implementar todos los requisitos explícitos/implícitos que desea el cliente.
- Ser una guía legible y comprensiva para programadores y aquellos que dan soporte al software.
- Debe dar una imagen completa del software.

Conceptos de Diseño como tales

Abstracción

Concentrarse en un problema de forma generalizada obviando detalles de bajo nivel. Hay dos tipos de abstracción: procedimental -> secuencia de instrucciones con una funcionalidad específica ; datos -> una colección de datos que representan un objeto real.

Arquitectura

La estructura organizacional de los componentes del programa y ver como dichos componentes interactúan entre sí y con las estructuras de datos que utilizan.

Patrones

Solución para un problema recurrente dentro de un contexto dado. Es una solución demostrada y validada.

La solución describe una estructura de diseño que resuelve el problema. Proporcionan una descripción que indica si es aplicable

al proyecto, si es reutilizable o si puede servir de guía para crear un patrón similar.

Modularidad

Cuando el software se divide en componentes (módulos) nombrados y separados. Los módulos se integran para satisfacer los requisitos.

- Código monolítico: una función que hace todo (mal).
- Modularizar excesivamente: un módulo por instrucción (mal).

Ocultamiento de información

La información dentro de un módulo es inaccesible a otros módulos que no la necesiten.

Independencia funcional

Surge de unir Modularidad + Abstracción + ocultamiento. Se busca una alta cohesión y un bajo acoplamiento.

Cohesión

Un módulo altamente cohesivo es aquel que lleva a cabo una sola tarea y necesita poca interacción con otros módulos.

Un módulo es bajamente cohesivo si lleva a cabo varias tareas diferentes que no se relacionan entre sí.

Acoplamiento

Es la medida de interconexión entre módulos. Si un módulo depende de otros en una alta medida, entonces tenemos un acoplamiento alto. Deseamos tener un bajo acoplamiento.

Refinamiento

Ayuda a revelar los detalles de grado menor mientras se realiza el diseño, es una tarea sucesiva.

Refactoring

Técnica que sirve para reorganizar el diseño de un componente sin cambiar su comportamiento.

Clase 7 – Diseño de interfaz (UI) y algo de UX

Actividad multidisciplinaria que se encuentra en un amplio rango de proyectos. El objetivo de la UI es mantener la interacción con los usuarios de una forma atractiva y centrando el diseño en estos.

La UI es una categoría de diseño que **crea un medio de comunicación entre hombre y máquina**. Interfaces difíciles de usar provocan errores de parte de los usuarios o que estos rechacen usar el sistema. Se acepta que hay diversidad de percepciones, comprensiones entre un grupo de personas diferentes.

La interfaz debe brindar acceso rápido al contenido y conocimiento sin perder la comprensión del usuario.

6 principios del diseño de UI

1. **Análisis y comprensión del usuario**: para saber a donde apuntar el diseño.
2. **Prototipado en papel**: bosquejo hecho en papel que muestra como se vería la interfaz del sistema.
3. **Evaluación del prototipado**: comprueba el paso 2 y dice si es correcto (ir a paso 4) o no.
4. **Creación del diseño dinámico del prototipo**: hacerla aplicación incompleta con la parte gráfica
5. **Evaluar el diseño dinámico**: los usuarios finales lo prueban y dan un feedback.
6. **Implementación de interfaz definitiva**: se crea la aplicación más completa agregando BD.

User Design Experience (Udx)

Es un conjunto de métodos aplicados al proceso de diseño buscando satisfacer las necesidades del cliente para generar una buena experiencia para usuarios finales.

Reglas Doradas del Diseño

1. **Dar control al usuario**: el sistema reacciona a lo que necesita y lo ayuda a hacer sus tareas.
2. **Reducir la carga de memoria cerebral del usuario**: para que no piense tanto.
3. **Lograr una interfaz consistente**: la parte gráfica debe presentar consistencia.
4. **Pensar y adecuarse a factores humanos**.

Clase 8 – Parte 1, Diseño Arquitectónico

Define la relación entre los elementos estructurales del sistema de forma que se cumplan los requisitos. Es desarrollado en base a componentes y sus vínculos ya que los grandes sistemas se subdividen en subsistemas. **La idea es identificar dichos subsistemas para establecer cuál es el control y la comunicación entre estos.**

La arquitectura afecta a requisitos no funcionales

Rendimiento: operaciones críticas son agrupadas en un grupo pequeño de sub-sistemas con una baja comunicación.

Seguridad: el uso de arquitectura en capas protege recursos críticos de capas internas.

Protección: la arquitectura se debe diseñar para que las operaciones de protección estén en un único sub-sistema, esto para reducir costos y problemas de validación.

Disponibilidad: se debe diseñar con componentes redundantes que permite el reemplazo sin detener el sistema, genera tolerancia a fallas.

Mantenibilidad: se debe diseñar el sistema usando una alta cantidad de componentes pequeños (grano fino) para que sea más fácil la modificación.

Organización del sistema

Representa la estrategia que se usa para estructurar el sistema. Donde el intercambio de información entre subsistemas se logra teniendo datos compartidos en una BD central y haciendo que cada subsistema mantenga su información.

Patrones de Arquitectura

Patrón Repositorio

Patrón que consiste en una BD central compartida por muchos subsistemas, los datos de esa BD los genera un subsistema en particular. Usado en sistemas que manejan grandes cantidades de información.

Ventajas: Tenemos una forma eficiente de compartir grandes cantidades de datos, de esta manera no hay comunicación entre subsistemas, independizándolos. El backup, protección y control de acceso se centralizan en el repositorio de información y las nuevas herramientas se integran de forma directa.

Desventaja: Es difícil evolucionar estos sistemas ya que al generar mucha información nos va a costar más poder migrarla en un futuro. Los distintos subsistemas pueden tener distintas políticas de seguridad y el repositorio impone las mismas para todos.

Patrón Cliente-Servidor

El sistema se organiza en tres componentes:

- Un conjunto de servidores que ofrecen servicios.
- Un conjunto de clientes que llaman a los servicios.
- La red que conecta el cliente con dichos servicios.

El servidor NO necesita conocer al cliente.

Patrón de arquitectura en capas

El sistema se divide en capas, cada capa otorga servicios a sus capas adyacentes.

La ventaja es que permite un desarrollo incremental, ser portable y resistente a cambios, también una capa puede ser reemplazada si se mantiene la interfaz (si no se usa un adaptador), también permite generar sistemas multiplataforma.

Es difícil dividir en capas, las capas más bajas dan servicios que usan todos los niveles, es muy probable navegar de más entre capas para hallar un servicio que desee el usuario.

Descomposición Modular

Es dividir un subsistema en módulos.

Estrategias de descomposición

Orientada a flujo de funciones

Se ingresan datos que fluyen de una función a otra de forma que se transforman hasta llegar a datos de salida.

Orientada a objetos

El sistema se estructura como un conjunto de objetos débilmente acoplados y con interfaces bien definidas.

Modelo de control

Es controlar subsistemas para que los servicios sean entregados en el lugar correcto al momento preciso (forma de ejecutar).

Dos formas de control

Control Centralizado

Se diseña un subsistema como controlador que se responsabiliza de gestionar la ejecución de otros subsistemas. Aplicable a modelos secuenciales de subrutinas que descienden (como el top-down).

Control basado en eventos

Los subsistemas responden a eventos externos, de forma que un gestor controla el inicio y la parada de cada uno. Aplicable a modelos concurrentes.

Sistemas dirigidos por eventos

Aquellos que se rigen por eventos externos al proceso.

Varios tipos de sistemas dirigidos por eventos (modelos)

Broadcast

Se transmite un evento a todos los subsistemas, el subsistema programado para manejar ese evento lo atenderá.

Interrupciones

Usados en sistemas de tiempo real, se generan interrupciones detectadas por un manejador para enviarlas a algún componente que las procese.

Arquitectura de sistemas distribuidos

Es un sistema donde el procesamiento es realizado sobre varias computadoras. De forma que se comparten recursos, se manejan aspectos de concurrencia, son sistemas escalables y tolerables a fallos.

El problema es que son sistemas más complejos (comunicación y sincronización de PC), el tráfico que generan las PC puede ser intervenido, el hecho de tener PC de distinto tipo y S.O genera dificultades de gestión y también son sistemas impredecibles ya que la respuesta a una petición es variable entre peticiones.

Cuatro grandes arquitecturas distribuidas

Multiprocesador

Sistema que posee varios procesadores distintos que reciben procesos mediante forma predeterminada o por un dispatcher. Es normal ver estos sistemas en grandes sistemas de tiempo real que recolectan información y toman decisiones o envían señales.

Cliente-servidor

Ya explicado antes. Pero esta arquitectura se clasifican en niveles

Dos niveles

Cliente ligero: el procesamiento y gestión de datos se lleva a cabo en el servidor.

Cliente pesado: el proceso cliente implementa la lógica de la app y el servidor solo gestiona datos.

Multinivel

La presentación, procesamiento y gestión de datos son procesos separados lógicamente y pueden ser ejecutados en procesadores distintos.

De componentes distribuidos

El sistema es un conjunto de componentes que brindan una interfaz de los servicios que otorgan que serán usados por otros componentes de forma que no se distingue cliente y servidor. Los componentes pueden usarse en otras máquinas a través de la red usando un middleware que intermedie peticiones.

Distribuida inter-organizacional

Una organización tiene varios servidores donde se reparte la carga computacional.

Arquitectura Peer-to-Peer (P2P)

Sistema descentralizado donde el cálculo puede hacerse en cualquier nodo de la red de servidores. Aprovechando la ventaja de la potencia computacional y el almacenamiento de red.

Usa alguna de las dos arquitecturas

Descentralizada: cada nodo rutea los paquetes a sus vecinos hasta hallar el destino.

Semi-centralizada: un servidor ayuda a conectar los nodos o coordinar resultados.

Arquitectura Orientada a servicios

Un servicio es una representación de un recurso computacional (código) de forma que es independiente. Existe un proveedor de servicios que otorga servicios definiendo interfaz y funcionalidad. Un solicitante enlazará dicho servicio a su aplicación de forma que hace el código para invocar el servicio y procesar el resultado del mismo.

Clase 8 – Parte 2 , Codificación SOLO CONCEPTOS RELEVANTES

Pautas generales de código

- Las entradas y salidas deben estar separadas del resto del código.

- El pseudocódigo nos ayuda a avanzar en el diseño.
- Revisar y reescribir borradores.
- Reutilizar
 - **Reutilización productiva:** crear componentes que se reutilizarán por otra app.
 - **Reutilización consumidora:** se usan componentes desarrollados para otros proyectos.

Documentación

Conjunto de descripciones escritas que explican al lector qué hace el programa y como lo hace.

Se divide en:

- **Documentación interna:** concisa, escrita a nivel programador ya que se dirige a quienes leerán el fuente. Incluyendo información de algoritmia, estructuras de control y flujos de control.
- **Documentación externa:** se leerá por quienes tal vez jamás vean el código real.

Clase 9 – Estrategias de Pruebas

Una estrategia de pruebas nos otorga una guía con pasos a seguir, de forma que se pueda planificar, diseñar casos de prueba, ejecutar y recolectar/evaluar resultados. Son actividades planeadas con anticipación e incluye pruebas de bajo/alto nivel. Estas actividades forman parte de la verificación y validación de la calidad del software.

Verificar y Validar

Verificar es un conjunto de actividades que asegura que el software haga una función específica de forma correcta.

Validar es un conjunto de actividades que aseguran que el software es lo que pidió el cliente.

Tipos de prueba convencionales

Prueba de unidad

Se verifica que el módulo funciona correctamente de forma que se asegure que la información viaja de manera adecuada. Se examinan datos locales, se prueban condiciones límite y se ejercitan caminos de ejecución del módulo. La idea es esto de

detectar cálculos incorrectos, comparaciones erróneas o flujos de control inapropiados.

Para cada prueba de unidad se desarrolla un controlador o resguardo. El controlador es un programa principal que acepta los datos de los casos de prueba y pasa estos al módulo a ser testeado. El resguardo sirve para reemplazar módulos que utiliza el componente a probar. Tanto controlador y resguardo son trabajo extra por lo que hay que realizarlos de forma sencilla. Un módulo altamente cohesivo simplifica esto.

Prueba de integración

Se combinan aquellos componentes que hayan pasado la prueba de la unidad y se los prueba.

Puede ocurrir que haya datos que se pierdan, que un componente rompa otro, que no surja el resultado deseado. La idea de integrar es construir y probar en segmentos pequeños para aislar errores y corregirlos más fácil.

La integración se puede dar de forma descendente (desde prog principal, pero necesita resguardos) o ascendente (nivel más bajo, el programa entero no existe hasta agregar el último módulo), aunque un enfoque combinado (sandwich) es mejor.

Prueba de regresión

Consiste en volver a ejecutar un subconjunto de pruebas llevadas a cabo anteriormente para asegurar que los cambios no han provocado efectos colaterales no deseados. Se pueden probar todas las funciones del software, se pueden probar funciones del software que pueden ser afectadas por el cambio o se pueden probar los componentes que han cambiado.

Los módulos críticos son aquellos que:

- Abordan muchos requisitos.
- Alto nivel de control.
- Complejo o propenso a errores.
- Tiene requerimientos de rendimiento definidos.
- Deben ser probados ASAP, la prueba de regresión nos ayuda en esto.

Prueba del sistema

Conjunto de pruebas diferentes con propósitos distintos de forma que se verifica la correcta integración de todos los componentes.

Prueba de recuperación: se comprueba la reacción ante fallas forzando el fallo.

Prueba de seguridad: se comprueban mecanismos de protección.

Pruebas de Resistencia (stress): se diseñan para enfrentar situaciones anómalas.

Pruebas de rendimiento: se testea el sistema en tiempo de ejecución.

Pruebas de validación

Comienzan cuando terminan las pruebas de integración, la idea es validar el software de forma que se demuestre la conformidad del cliente con los requisitos. Puede ocurrir que las características se acepten o que se descubran desviaciones de especificaciones haciendo que la característica vaya a una lista de deficiencias.

Pruebas de aceptación – Alfa y Beta

Estas pruebas son realizadas por el usuario final.

Aceptación ALFA

Llevadas a cabo por el cliente en el lugar de desarrollo del sistema, en el entorno controlado, se produce después de las pruebas del sistema. El desarrollador está de espectador registrando errores o problemas de uso.

Aceptación BETA

Llevadas a cabo por usuarios finales del software en lugar de trabajo del cliente. El desarrollador no está presente sino que el cliente registrará los problemas encontrados e informará al desarrollador. La prueba beta se hace utilizando técnicas de caja negra.

Depuración

Es el proceso de identificar y corregir errores en software. Tiene dos resultados posibles: hallar y corregir el error; no hallar el error, podemos sospechar que causa el error.

Clase 10 – Tipos de Prueba

Prueba del Software

Tienen como objetivo primario diseñar pruebas que saquen a la luz distintos tipos de errores en una menor cantidad de tiempo/esfuerzo.

Una prueba tiene éxito al descubrir errores. Lo bueno de estas pruebas es poder descubrir errores o detectarlos antes de que el software salga del

entorno de desarrollo, bajando así costos de corrección de errores en etapa de mantenimiento.

Principios de la prueba

- Toda prueba debe poder ser seguida hasta los requisitos.
- Deberán ser planificadas mucho antes de aplicarlas.
- “El 80% de errores se genera por un 20% del código, el 80% de código restante genera solo el 20% de errores restantes.
- Las pruebas empiezan de lo pequeño para progresar a lo grande.
- Asegurarse de probar todas las condiciones a nivel componente.
- Las pruebas las debe realizar un equipo independiente al desarrollo para más eficacia (equipo de testers).
 - Evita conflictos, permite realizar pruebas en concurrencia con desarrollo y los desarrolladores colaboran.

Tipos de prueba

Caja Blanca

Es un examen minucioso de cada procedimiento, de forma que se ejercitan los caminos de ejecución del software proponiendo casos de prueba que ejercitan conjuntos específicos de condiciones y/o búcles de cada módulo.

Caja Negra

Es una prueba de comportamiento centrada en requisitos funcionales, menos minuciosa que la anterior y se enfoca en la interfaz del software. Buscando funciones incorrectas, errores de interfaz, de estructuras de datos o acceso a BD, de rendimiento, de inicialización y terminación, etc.

Clase 11 – Mantenimiento y Auditoria

Mantenimiento

Atender al sistema después de que haya sido entregado, es la fase llamada “Evolución”. Solucionar errores, agregar mejorar y optimizar el software provoca altos costos adicionales.

A veces es necesario mantener sistemas heredados que son viejos, pueden no tener metodología, documentación o modularidad por lo tanto tenemos que evaluar cuando nos conviene cerrar el ciclo de vida de un sistema viejo y reemplazarlo por otro. Dicha decisión se toma en función de cuanto nos cuesta el ciclo de vida del viejo proyecto y la estimación de uno nuevo.

Características de mantenimiento

- Mantener puede implicar la existencia de efectos secundarios sobre código, datos y documentación.
- Las modificaciones pueden disminuir la calidad del producto.
- Las tareas de mantenimiento provocan un reinicio de fases de análisis, diseño e implementación.
- **Implica un 40% a 70% del costo total del desarrollo.**
- Los errores provocan clientes insatisfechos.

Problemática de mantener

No es un trabajo atractivo, no siempre en el diseño se prevén cambios a futuro y el código ajeno es difícil de entender, más si está mal documentado o no documentado.

Ciclo de mantenimiento

1. **Análisis: comprender alcance y efectos que implican la modificación.**
2. **Diseño: rediseño para incorporar cambios.**
3. **Implementación: recodificar y actualizar documentación interna del código.**
4. **Prueba: revalidar el software.**
5. **Documentación de apoyo: actualizarla.**
6. **Distribuir las nuevas versiones.**

Tareas en el desarrollo que ayudan al mantenimiento

- Análisis: especificar controles de calidad, identificar posibles mejoras a futuro, estimar recursos para mantenimiento.
- Diseño arquitectónico: claro, modular, modificable y con notaciones estandarizadas.
- Diseño detallado: notaciones para algoritmos y estructuras de datos, especificación de interfaces, manejo de excepciones, efectos colaterales.
- Implementación: buena indentación, código comentado, codificación simple y clara.
- Verificación: lotes de prueba y resultados.

Tipos de mantenimiento

- **Correctivo: para diagnosticar y corregir errores. (El tercero)**
- **Adaptativo: modificar el software para que interaccione correctamente con el entorno. (El segundo)**
- **Perfectivo: mejora al sistema. (El más común)**
- **Preventivo: se hace antes de que se petitionen cambios, para facilitar el mantenimiento futuro. (El menos común)**

Rejuvenecimiento del Software

- Intenta aumentar la calidad global de un sistema existente.

Tipos de rejuvenecimiento

- Re-documentar: analizar código para producir la documentación del sistema.
- Re-estructurar: reestructurar el software para hacerlo más entendible.
- Ingeniería inversa: para sistemas indocumentados, se parte desde el fuente para recuperar el diseño y en ocasiones la especificación.
- Re-ingeniería: extensión de la ingeniería inversa, produce un nuevo código fuente correctamente estructurado de forma que se mejora la calidad sin cambiar la funcionalidad del sistema.

Auditoria informática

Es un examen crítico que se hace con el objetivo de evaluar eficiencia y eficacia de un organismo/sección para decidir cursos alternativos de acción de forma que se mejore la organización y se logren objetivos propuestos. NO es una actividad mecánica. Se revisan/evalúan sistemas y procedimientos informáticos, equipos computacionales y la organización que participa en el procesamiento de la información.

La finalidad es definir estrategias para prevenir delitos y problemas legales, por lo tanto es una actividad de prevención.

Objetivos de la auditoria

- » Salvaguardar los activos.
- » Integridad de datos.
- » Efectividad de sistemas.
- » Eficiencia de los sistemas.
- » Seguridad y confidencialidad.