Resumen Teoría INGENIERIA DE SOFTWARE 2

Clase 1

Proceso de software: conjunto de actividades y resultados que producen un producto de software

Actividades: elicitación, desarrollo, validación y evolución

Elevator pitch: mensaje corto para contar un proyecto de forma diferenciadora

Elicitacion: proceso de adquirir todo conocimiento relevante para conocer los requerimientos de un sistema

Características de un requerimiento: necesario, conciso, completo, consistente (no contradictorio con otro), no ambiguo, verificable

Dificultades de un requerimiento: no obvios, interrelacionados, suelen ser muchos, pueden cambiar en el desarrollo

Tipos de requerimientos: funcionales (definen el comportamiento, o describen tareas que debe hacer) y no funcionales (aspectos deseables para el usuario, o restricciones)

CONOPS: documento que describe un sistema desde el punto de vista del usuario

SRS (Requerimientos del Software): especificación de las funcionalidades que realiza un software

Características de un SRS: alcance, naturaleza, ambiente, correcto, no ambiguo, completo, consistente, priorizado, comprobable, modificable

Clase 2

GCS (Gestión de la Configuración del Software): proceso de identificar y controlar los elementos de un sistema y los cambios q tengan a lo largo de su ciclo de vida.

Un GCS sirve para: identificar, controlar, garantizar el funcionamiento e informar todo cambio detectado

Etapas de un GCS: identificación, control de versiones, control de cambios, auditorias, generación de informes

Para controlar un cambio, cuando se detecta la necesidad del mismo se debe hacer la petición, que la evalúe un desarrollador y generar un informe, para que luego la Autoridad de Control de Cambios (ACC) decida si se hace o no.

Para decidir el ACC evalúa como impactara el cambio en el hardware, rendimiento, cómo ve el cliente al producto, calidad, etc.

Clase 3

Proyecto de software: esfuerzo temporal para crear un producto o servicio de software

Las 4 P: Personal (RRHH), Producto, Proceso y Proyecto

Signos de mala gestión: incumplimiento de plazos, incremento de costos, entrega de productos de mala calidad

Elementos clave de la gestión de proyectos: métricas, estimaciones, calendario temporal, organización del personal, análisis de riesgos, seguimiento y control

Planificación: QUE debe hacerse, CON QUE recursos y EN QUE ORDEN

Organizaciones de equipos:

- DD: descentralizado democrático. No hay jefe, se nombran coordinadores por tareas, decisiones por consenso y comunicación horizontal. Para problemas complejos o para equipos que van a perdurar por mucho tiempo.
- DC: descentralizado controlado: Hay un jefe para ciertas tareas y subjefes para subtareas, las resoluciones son en el grupo general pero la aplicación en subgrupos. Comunicación horizontal y vertical. Mejores para proyectos grandes y para problemas complejos.
- CC: centralizado controlado: Hay un jefe que se encarga de tareas de alto nivel y coordinación interna del equipo. Comunicación vertical. Útil para tareas rápidas, problemas sencillos o para proyectos grandes.

Clase 4

Planificación temporal: actividad que distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto.

Calendarización del proyecto: distribución del esfuerzo en tareas específicas

Tarea: secuencia de acciones a hacer en un plazo

Tarea crítica: tarea que si se atrasa, atrasa todo el proyecto

Hito: tarea que se espera esté hecha para alguna fecha

Métodos de Planificación temporal: GANTT, PERT, CPM (Critical Path Method)

Camino crítico (combinación de PERT y CPM): establece una lista de tareas y fija una dependencia entre tareas y duraciones. Se puede calcular fecha temprana y tardía de cada tarea.

FORMULA DE FECHA TEMPRANA:

Fecha Temprana = Fecha temprana de la tarea anterior + duración de mi tarea

FORMULA DE FECHA TARDIA:

Fecha Tardía = Fecha tardía de tarea siguiente – duración de tarea siguiente

FORMULA DE MARGEN TOTAL:

Margen total = Fecha tardía de nodo siguiente – duración de tarea

Margen total = Fecha temprana mía – Fecha tardía mía

Ejemplo: tarea con margen total de 6 días: la tarea puede atrasarse 6 días sin retrasar al proyecto

Si una tarea tiene margen total 0 (inicia y termina en fecha temprana) es una tarea critica. **Un camino crítico es una sucesión de tareas críticas.**

Del método PERT-CPM se obtiene: camino crítico, fecha temprana y tardía de inicio y fin de una tarea, margen total.

Clase 5

Un riesgo es un evento no deseado que tiene consecuencias negativas

Los riesgos deben evitarse de ser posibles, y sino minimizar las consecuencias

Tipos de estrategias para gestionar riesgos: reactivas (al momento) o proactivas (prepararse)

Tipos de riesgos: Genéricos o Específicos, ambos pudiendo ser conocidos, predecibles o impredecibles

Etapas de gestión: identificación, análisis, planeación, supervisión (repite constantemente)

Se suele hacer brainstorming para tratar de identificarlos

Categorías: del proyecto, del producto, del negocio.

Hay que analizar cada riesgo teniendo en cuenta su probabilidad y su impacto.

Un riesgo que tenga:

- **Gran impacto** pero **poca probabilidad**, no debería absorber un tiempo significativo.
- **Gran impacto** con **probabilidad moderada o alta**, o **poco impacto** con **gran probabilidad** deberían tomarse en cuenta.

Fórmula para ver si se justifica reducir un riesgo: (influencia debe ser alto)

Influencia = (EXPOSICION ANTES – EXPOSICION DESPUES)/ Costo de Reducción

Exposición= probabilidad de riesgo * costo total de proyecto

Siempre deben monitorizarse los riesgos, al final cada etapa del proyecto se debe revisar si cambio la probabilidad de algún riesgo, la efectividad de alguna estrategia, si ocurrió algún riesgo, etc.

Clase 6

Métricas: medidas usadas para saber cuáles son las mejores decisiones para tomar. Asegura la calidad en procesos, proyectos y productos.

Tipos: métrica de proyecto (usadas para hacer ajustes al calendario, evitar demoras, rastrear riesgos, etc.), métricas del proyecto (recopilación a través de varios proyectos para obtener indicadores para mejorar un proceso)

Las métricas de un producto pueden ser dinámicas (eficiencia, fiabilidad) o estáticas (complejidad, comprensibilidad y mantenibilidad), por ejemplo longitud de código o complejidad ciclomática.

LDC: métrica de líneas de código. Depende del lenguaje. Permite conocer productividad (KLDC/personasXmes), calidad (errores/KLDC) o costo ($/KLDC). También se puede obtener la cantidad total de líneas si se suman los comentarios, u obtener la densidad de comentarios dividiendo cant comentarios entre el total de líneas

Métrica de Punto Función: métrica que mide la cantidad de funcionalidad de un sistema descripto. Toma en cuenta todas las entradas, salidas, consultas, almacenamientos internos y externos, cada uno multiplicado por un valor acorde a su dificultad.

También permite calcular productividad, calidad y costo.

Clase 7

Estimaciones: técnicas para dar un valor aproximado a algo que se quiere medir. Se usan varias fórmulas de estimación y se hace un promedio. No son exactas y se ven muy afectadas por los cambios.

Estimación de recursos: estimar la cantidad necesaria, habilidades, etc.

Estimación de costos: esfuerzo, hardware/software y viajes (no es lo mismo que precio, costo es lo que cuesta y precio lo que lo cobro)

Estimación de tiempo: cálculo de fechas tempranas y tardías de inicio y fin

Técnicas: juicio de experto (consultas a varios expertos, que comparan, estiman y discuten), Técnica Delphi (parecida a la anterior pero las evaluaciones de cada experto son independientes, no hablan entre sí), y división de trabajo (no se pregunta a expertos, sino que va de lo más bajo de la jerarquía para arriba)

Clase 8

Diseño de software: representación de algo que se va a construir. Independiente del modelo a usar. Acá se reúnen todos los requisitos, necesidades y técnicas.

Si no se tiene buen diseño, el sistema puede ser inestable y tener grandes chances de fallar ante un cambio.

Tipos: diseño de datos (transforma lo obtenido del análisis a estructuras de datos), diseño arquitectónico (define relaciones entre las estructuras del software), diseño a nivel de componentes (transforma los componentes en una descripción procedimental de los mismos), diseño de interfaz (marca la comunicación entre sistema y algo ajeno, usuario u otro sistema).

PRINCIPALES Conceptos: *ABSTRACCION* (permite concentrarse en un problema a nivel general en lugar de detalle. Puede ser de tipo **procedimental** (secuencia nombrada de instrucciones con funcionalidad especifica) o **de datos** (colección nombrada de datos que definen un objeto real)), *INDEPENDENCIA FUNCIONAL (*se busca que el modelo tenga alta cohesión y bajo acoplamiento.)

COHESION: relación funcional entre un grupo de sentencias de un mismo modulo. Mucho es bueno, poco es malo. Tipos de mayor a menor: funcional, comunicacional, procedimental, temporal, lógica y coinciden tal)

ACOPLAMIENTO: dependencia entre módulos. Bajo es bueno, alto es malo. Tipos de mayor a menor: de contenido, externo, común, de control, de marca y de datos.

Si un módulo tiene alta cohesión y alto acoplamiento o baja cohesión y bajo acoplamiento, no podemos decir si es bueno o malo.

Clase 9

Diseño de interfaz: diseño de computadores, apps, etc., desde el punto de vista del usuario para mantener la interacción con los mismos de forma más atractiva. Intenta que los usuarios aprendan el funcionamiento lo más rápido posible. LA TECNOLOGIA SE DEBE ADAPTAR A LOS SERES HUMANOS.

Reglas de diseño de Theo Mandel: *dar control al usuario* (definir interfaz que evite acciones innecesarias, agregar opciones de “deshacer” o “cancelar”, mostrar solo los objetos de interacción directa), *reducir la carga cognitiva del usuario* (definir valores x defecto, accesos intuitivos, no desglosar toda la información de una), *lograr interfaz consistente* (el usuario debe saber dónde está, de donde viene y a donde va)

La usabilidad no tiene que ver con la estética sino de que la arquitectura se ajuste a las necesidades de los usuarios.

Principios de la usabilidad de Jacob Nielsen:

- Dialogo simple y natural (escritura correcta y sin errores, sin abreviaturas)
- Lenguaje del usuario (no usar lenguaje técnico)
- Minimizar carga de memoria del usuario (dar información de contexto)
- Consistencia (que no haya ambigüedades ni de diseño ni de comportamiento)
- Feedback (mantener informado al usuario de lo que está sucediendo)
- Salidas evidentes (opciones visibles de salida o abandono del contexto actual)
- Mensajes de error (ante un error informar porque fue y alternativas posibles)
- Prevención de errores (evitar que el usuario pueda llegar a error con rangos, valores x defecto)
- Atajos (debería tener manejo rápido para usuarios nuevos o experimentados)
- Ayudas (asistencia al usuario)

Estilos de interfaz:

- De comandos (solo contexto, poderoso pero difícil de aprender, mala administración de errores)
- Selección de menú (selección entre opciones, evita errores pero lento para experimentados)
- Grafica de usuarios (usa medios visuales para interactuar, fácil de aprender y usar)
- Relleno de formularios (datos sencillos en campos marcados, fácil de aprender pero ocupa mucho espacio)
- Manipulación directa (hardware especifico o táctil)
- Reconocimiento de voz
- Inteligente

Una interfaz accesible es la que respeta las normas de diseño universales para mejor acceso de cualquier usuario.

Clase 10

Diseño arquitectónico: relación entre los elementos estructurales para cumplir con los requisitos de un sistema. Identificar subsistemas dentro del sistema.

La arquitectura afecta directamente a los requisitos NO funcionales como rendimiento, seguridad, protección, mantenibilidad, etc.

Principales estilos de organización (Patrones arquitectónicos):

- Repositorio: al usar gran cantidad de datos se organizan como una gran de datos compartida. Es eficiente para compartir grandes datos y hacer backups, pero a veces tiene mal rendimiento, y migrar datos.
- Cliente-servidor: son servidores que aportan servicios y clientes los cuales acceden a los mismos. Los clientes conocen los servidores pero no al revés.
- Arquitectura en capas: cada capa presenta un conjunto de servicios a las subyacentes. Soporta el desarrollo incremental y es tolerante a cambios, pero es difícil de estructurar, además de que si el servicio está en una capa interna hay q atravesar todas las previas.

Clase 11

Una vez que se organizó el sistema, se lo subdivide en módulos, que deben ser controlados para garantizar que los servicios vayan a un lugar específico en un momento específico.

Modelos de control:

Centralizado (un subsistema inicia y detiene a otro) puede ser de llamada y retorno (modelo secuencial, aplicable a subrutinas descendentes) o de gestor (un gestor coordina el inicio y fin con el resto de los procesos, aplica a modelos concurrentes)

Basado en eventos (responde ante eventos ajenos al sistema) puede ser de transmisión (transmite a todos y el que este programado para manejarlo lo atiende) o dirigido por interrupciones (usado en sistemas de tiempo real, hay un manejador de interrupciones que capta y redirecciona a un componente para ser tratada)

Sistema distribuido: el procesamiento se distribuye en varias computadoras.

Tipos genéricos: cliente-servidor, componentes distribuidos

Características: compartir recursos, son abiertos, concurrentes, tolerantes a fallos, y buena capacidad de escalabilidad.

Desventajas: complejos, menos seguros (el tráfico de un componente a otro puede ser interrumpido), impredecibles

TIPOS DE ARQUITECTURAS:

- **Multiprocesador** (formado por varios procesos que pueden, o no, ejecutarse en distintos procesadores, de forma predeterminada o determinada por un dispatcher)
- **Cliente-servidor** (servidores que aportan servicios y clientes que los usan)
    - Dos niveles: cliente ligero (el procesamiento y gestión de datos es en servidor), y cliente pesado (el procesamiento es en cliente y gestión de datos en servidor)
    - Multinivel: presentación, procesamiento y gestión están separados y pueden hacerse en diferentes procesadores.
- **Objetos distribuidos** (conjunto de objetos que brindan servicios que ellos mismos suministran. Clientes y servidores son lo mismo)
- **Computación distribuida inter-organizacional** (organización de servidores que reparte la carga equitativamente)
    - P2P: el cálculo se hace en cualquier nodo. Puede ser descentralizado (cada uno dirige los suyo) o semi-centralizado (un servidor guía paquetes de un nodo a otro)
    - Orientada a servicios. (servicio: representación de recurso o información para que pueda ser usado por otro programa)

CODIFICACION

El código debe ser comprensible y buscar que sea reutilizable. Se debe contar con documentación interna (escrita para un programador o alguien que vera el código fuente) y externa (para alguien que no vera el código, como un diseñador)

Clase 12

Estrategia de pruebas: guía que describe pasos a seguir sobre las pruebas (cuando se planean, hacen, como, cuanto tiempo, con que)

PRUEBA: conjunto de actividades planeadas con anticipación y hechas de manera sistemática.

La **verificación** apunta a chequear que se cumplan los requerimientos del SRS, mientras que la **validación** apunta más a chequear si dichos requerimientos verificados cumplen con las expectativas del cliente.

Tipos de prueba: de unidad (chequean q un componente funcione bien con distintos ingresos), de integración (verifica que trabajen bien entre sí), de validación (chequeo extra que se cumplen los requisitos funcionales y no funcionales), de sistema (chequea que todo ande bien y se logre la funcionalidad y rendimiento final deseados)

Prueba de unidad: chequea q la interfaz ande bien. Prueban los casos límites. Suelen descubrirse errores de cálculo, o comparaciones incorrectas. Si el modulo tiene alto grado de cohesión es más fácil.

Prueba de integración: se combinan los módulos aprobados x unidad de acuerdo al diseño. Errores comunes son los efectos adversos de un módulo a otro. Puede ser ascendente (de lo más atómico a lo más general, por lo que no se necesitan resguardos) o descendente (del programa principal hacia abajo)

Prueba de regresión: vuelve a ejecutar pruebas sobre módulos ya testeados para chequear que no se haya roto nada con los cambios hechos al software. Hacen foco en los MODULOS CRITICOS.

> Módulos críticos: módulos que abordan muchos requerimientos, tienen alto nivel de control, son complejos o tienden al error.

Clase 13

Prueba de sistema: serie de pruebas que en conjunto verifican que todos los elementos del sistema se integraron bien. Usa varios tipos de pruebas: recuperación de fallas, seguridad, stress, rendimiento.

Prueba de validación: serie de pruebas que demuestran la conformidad del cliente con los requisitos.

> ALFA: hechas por el cliente en un ambiente controlado y con el desarrollador tomando nota.
>
> BETA: hechas por los usuarios finales en un entorno no controlado. El cliente registra errores y los comunica al desarrollador. Se utiliza técnicas de CAJA NEGRA

Depuración: proceso de corrección de errores. No es una prueba pero ocurre como consecuencia de una.

Una vez terminado el proceso de depuración se hacen pruebas de regresión.

A medida que el software se complejiza, también lo hacen los métodos de prueba:

Pruebas de arquitectura cliente-servidor: prueban que la funcionalidad sea correcta, el buen manejo de datos y desempeño del servidor, la seguridad y exactitud de la base de datos, la fiabiabilidad de las transacciones y de la comunicación entre nodos.

Pruebas de la documentación ayuda: se revisa tanto la claridad de los documentos como que efectivamente sirvan para ser usados sobre el software

Pruebas de sistema en tiempo real: serie de pruebas para verificar el buen manejo de eventos, paralelismo, entre otras cosa.

Clase 14

Si el software falla es porque tiene defectos.

Clasificación ortogonal de defectos: por omisión (falta parte de código, como inicializar una variable) o de cometido (no falta nada pero hay error, como una inicialización con valor incorrecto)

Tipo de defectos:

- Función: afecta la capacidad e interfaces
- Interfaz: afecta a la interacción con otros componentes
- Comprobación: afecta la lógica del programa
- Asignación: afecta la estructura de datos
- Sincronización: involucra sincronización de recursos compartidos y de tiempo real
- Construcción: pasa por problemas en repositos, gestión de cambios o control de versiones
- Documentación: afecta a publicaciones
- Algoritmo: involucra la eficiencia o exactitud de un algoritmo

Pruebas de software: buscan descubrir errores antes de que el software salga del entorno de desarrollo, además de bajar costos de corrección en etapa de mantenimiento.

Tipos: CAJA BLANCA (pruebas del "interior", analizan los procedimientos y su buen funcionamiento. Se analizan si o si al menos una vez todos los caminos lógicos) o CAJA NEGRA (pruebas del "exterior", analizan el sistema desde la interfaz)

Prueba de participación equivalente: prueban los casos de estados válidos y no válidos para una determinada condición de entrada.

Análisis de los valores límite (AVL): revisan las condiciones límites de entrada. Es buena en conjunto con las pruebas de participación equivalente.

La **complejidad ciclomatica** es una métrica que permite conocer una medición cuantitativa de la complejidad del software. Define el número de caminos independientes del conjunto básico de un programa y nos dice el numero límite máximo de pruebas necesarias para poder testear al menos una vez cada uno de ellos.

Clase 15

Mantenimiento: proceso de solución de errores, añadir mejoras, optimización, entre otros, que se hace durante la etapa de Evolución del Sistema (post entrega). Todo esto puede generar costos adicionales.

Puede causar disminución de otros desarrollos, puede haber efectos secundarios en el código, baja la calidad general, generalmente provoca que se reinicien las fases de análisis, diseño e implementación, puede provocar insatisfacción del cliente, además de involucrar **entre un 40% y un 70%** del costo del desarrollo.

Tipos: correctivo (corrige errores), adaptativo (modificación para mejor interacción con el entorno), perfectivo (mejoras) y preventivo (se hacen cambios anticipando errores)

Rejuvenecimiento del software: intentar aumentar la calidad global de un sistema ya existente.

Tipos de rejuvenecimiento:

- Re-documentación: producción de documentación a partir de análisis de código
- Re-estructuración: rearmar al software para mejor facilidad de entenderlo
- Ingeniería inversa: recupera el diseño y a veces la especificación a partir del código
- Re-ingeniería: extensión de ingeniería inversa, donde además se produce nuevo código correctamente estructurado.

Auditoria: examen crítico sobre un sistema u organismo para evaluar la eficiencia y eficacia y determinar el curso de acción para mejorar los mismos. Actividad **preventiva**. Busca comprobar la integridad de datos, efectividad de los sistemas, eficiencia de los mismos y seguridad y confidencialidad.