RESUMEN TRIMESTRE 1

Este resumen es un apoyo para repasar una vez ya se han estudiado las unidades más detenidamente. Es decir, esto es para echaros un mano, pero el el examen se hace sobre los contenidos de las unidades que tenéis subidas en el repositorio de la asignatura. Es decir, podría haber contenidos que no vengan aquí reflejados que siguen pudiendo aparecer en la prueba.

De cara a esa misma prueba, si pregunto (por ejemplo) por el ciclo de vida de los archivos en Git, no bastaría con responder únicamente lo que os he dejado 2 párrafos más abajo, ya que sería demasiado breve y estaría incompleto. Aclaro de nuevo que con este documento te señalo que esto es importante, no que esa sea la respuesta completa.

Unidad Didáctica 1

Conceptos básicos de Git y GitHub

- **Git**: Sistema de control de versiones distribuido; permite trabajo colaborativo, seguimiento de cambios y revertir errores.
- **GitHub**: Plataforma en línea basada en Git; facilita colaboración global, revisión de código y automatización.

Fundamentos de Git

- **Ciclo de vida de archivos**: Untracked → Staged → Committed.
- **Comandos clave**:
  - `git init`: Inicializa un repositorio.
  - `git add`: Prepara archivos para commit.
  - `git commit -m "mensaje"`: Confirma cambios.
  - `git status` y `git log`: Ver estado y historial.

Trabajo con GitHub

- Crear repositorio en GitHub y enlazarlo con `git remote add origin URL`.
- Subir cambios con `git push origin main` y actualizar con `git pull`.

Colaboración

- **Ramas**:
  - Crear: `git branch nombre_rama`.
  - Cambiar: `git checkout nombre_rama`.
  - Fusionar: `git merge nombre_rama`.

- **Pull Requests**: En GitHub, herramienta para revisión y aprobación de cambios.
- **Buenas prácticas**:
    - Commits frecuentes y descriptivos.
    - Uso de ramas para nuevas funcionalidades.
    - Revisión de código entre pares.

Conceptos avanzados

- **Deshacer cambios**:
    - `git reset`: Elimina commits recientes.
    - `git revert`: Revierte un commit específico sin alterar el historial.
- **Stash**:
    - Guardar temporalmente: `git stash`.
    - Recuperar: `git stash pop`.
- **Tags y releases**:
    - Crear tag: `git tag v1.0.0`.
    - Subir tag: `git push origin v1.0.0`.

Unidad Didáctica 2

Problemas bien definidos: Claros objetivos y parámetros específicos.
Problemas mal definidos: Objetivos ambiguos, requieren creatividad y análisis iterativo.

Estrategias generales

- **Definir el problema**: Identificar entradas, salidas, restricciones y aclarar ambigüedades.
- **Dividir en subproblemas**: Resolver por partes más pequeñas y manejables.
- **Prueba y error**: Experimentar diferentes soluciones y aprender de los errores.
- **Pensamiento algorítmico**: Planificar soluciones mediante pseudocódigo o diagramas.
- **Búsqueda de patrones**: Usar soluciones previas y analogías.

Algoritmos básicos

- **Búsqueda lineal**: Recorrer todos los elementos, complejidad **O(n)**.
- **Ordenación básica**:
    - **Bubble Sort**: Ineficiente para grandes listas (**$O(n^2)$**).
    - **Selection Sort**: Busca el menor en cada iteración (**$O(n^2)$**).
    - **Insertion Sort**: Eficiente para listas pequeñas o casi ordenadas.
- **Búsqueda binaria**: Divide y conquista en listas ordenadas, complejidad **O(log n)**.
- **Máximos y mínimos**: Buscar en listas desordenadas con complejidad **O(n)**.

Técnicas específicas

- **Depuración**:
  - Uso de depuradores y mensajes de seguimiento.
  - Documentar y aislar problemas.
- **Análisis de enunciados**:
  - Identificar entradas, procesos y salidas.
  - Crear casos de prueba adicionales.
- **Planificación**:
  - Diagramas de flujo y pseudocódigo para visualizar soluciones.
- **Pruebas**:
  - Caja negra: Verificar resultados sin conocer la implementación.
  - Caja blanca: Evaluar rutas internas del código.

Código limpio: legibilidad, mantenibilidad y simplicidad del código.

- **Nombres descriptivos**: Claridad y contexto en variables y funciones.
- **Funciones pequeñas**: Una sola responsabilidad y modularidad.
- **Eliminar redundancia**:
  - DRY: No repetir código innecesariamente.
  - YAGNI: No tener código por si acaso, sólo el que se necesita.
- **Organización lógica**: Estructurar funciones de lo general a lo específico.
- **Reglas clave**:
  - KISS: Mantener la simplicidad.
  - Boy Scout Rule: Mejorar el código constantemente.

Unidad Didáctica 3

UML (Lenguaje Unificado de Modelado): Herramienta estándar para modelar sistemas de software, facilitando su diseño, visualización, construcción y documentación.
Tipos de diagramas UML:

- **Estructurales**: Diagrama de clases, componentes, objetos, despliegue.
- **De comportamiento**: Diagramas de secuencia, estados, actividades, casos de uso.

Importancia: Mejora la comunicación entre equipos, identifica problemas tempranos y proporciona documentación clara para el mantenimiento.

Diagramas de estado UML

- Representan el **comportamiento dinámico** de un sistema: cómo los objetos pasan de un estado a otro mediante transiciones y eventos.

- **Elementos clave**:
    - Estados (condición temporal de un objeto).
    - Transiciones (cambio entre estados).
    - Eventos (desencadenantes de las transiciones).
    - Acciones (tareas realizadas al cambiar de estado o permanecer en él).
- **Aplicaciones**: Modelar el ciclo de vida de objetos (pedidos, usuarios), lógica de sistemas interactivos (máquinas expendedoras) y sistemas embebidos.

Diagramas de flujo

- Representan procesos mediante símbolos estandarizados:
    - **Óvalos**: Inicio/fin.
    - **Rectángulos**: Procesos.
    - **Rombos**: Decisiones.
    - **Paralelogramos**: Entrada/salida de datos.
- **Ventajas**: Claridad visual, detección temprana de errores, comprensión para equipos no técnicos.
- **Limitaciones**: Dificultad en sistemas complejos, mantenimiento constante en procesos cambiantes.
- **Usos**: Diseño de algoritmos, planificación previa al código, mapeo de procesos empresariales.

Diagramas de secuencia UML

- Muestran **interacciones entre objetos o componentes** a lo largo del tiempo.
- **Elementos principales**:
    - Actores (entidades externas como usuarios).
    - Objetos (instancias internas del sistema).
    - Mensajes (comunicaciones entre actores y objetos).
    - Líneas de vida (existencia temporal de un objeto/actor en la interacción).
- **Beneficios**:
    - Modelar escenarios dinámicos en sistemas distribuidos o basados en microservicios.
    - Documentar lógica de interacción para mantenimiento.
    - Identificar cuellos de botella o fallos en la comunicación.

Recomendaciones al crear diagramas:

- Mantén la simplicidad y claridad.
- Usa nombres descriptivos en actores y objetos.
- Divide diagramas complejos en partes más manejables.
- Verifica que reflejen correctamente los requisitos del sistema.