GOBIERNO DEL ESTADO DE MÉXICO | ESTADO DE MÉXICO ¡El poder de servir! | EDUCACIÓN
SECRETARÍA DE EDUCACIÓN, CIENCIA, TECNOLOGÍA E INNOVACIÓN

"2024. Año del Bicentenario de la Erección del Estado Libre y Soberano de México"

BACHILLERATO GENERAL
FORMATO DE DISEÑO SIMPLIFICADO DE SITUACIONES DIDÁCTICAS
ESCUELA PREPARATORIA OFICIAL No. 28
2DA EVALUACIÓN PARCIAL SEMESTRE "A" C. ESCOLAR 2024-2025

Nombre del Docente: Beatriz Morales Cruz **Materia: Sistemas de Información** **Competencia Genérica:** CG 5.2 Ordena información de acuerdo a categorías, jerarquías y relaciones CG 5.6 Utiliza las tecnologías de la información y comunicación para procesar e interpretar información CG 8.1 Propone maneras de solucionar un problema o desarrollar un proyecto en equipo, definiendo un curso de acción con pasos específicos. **Competencia Disciplinar BÁSICAS:** CPBTIC5 Propone el diseño de sistemas de información, a partir del análisis de las necesidades de los usuarios, permitiendo la solución de problemas de manera responsable e innovadora en diferentes contextos. **Núm. de Bloque/Tema del Bloque:** MÓDULO III: Desarrollo de sistemas **Propósito de Módulo:** Plantea soluciones críticas y responsables mediante la metodología de desarrollo de software para demostrar eficiencia en el manejo de base de datos y software de programación de alto nivel, que sean aplicables a necesidades de una empresa o institución para el tratamiento de información.	**Semestre:** **QUINTO** **Grado:** **TERCERO**	**Periodo de Aplicación:** **07 de octubre al 22 de noviembre** **Grupos:** **3ro. 1, 2, 3**	**No. de Sesiones:** **14** **Turno:** **Vespertino**

Nombre de la Situación Didáctica o Descripción de la Competencia

Fundamentos de sistemas:

- Diseño
- Codificación
- Pruebas
- Validación
- Mantenimiento y evaluación.

Aprendizajes esperados:

Diseño de Software
Desarrolla diseños de sistemas de software utilizando metodologías formales, como diagramas de flujo de datos y modelos entidad-relación, asegurando que el diseño cumpla con los requerimientos del sistema.
Codificación
Transforma un diseño de software en código funcional utilizando un lenguaje de programación, siguiendo buenas prácticas de programación y asegurando que el código sea legible, modular y funcional.
Pruebas de Software
Diseña y ejecuta pruebas de software que validen la funcionalidad del sistema, utilizando pruebas unitarias y de integración, identificando errores y proponiendo soluciones correctivas.
Validación
Puede validar que el software desarrollado cumple con los requerimientos del usuario final, utilizando técnicas de retroalimentación y validación para proponer mejoras basadas en las necesidades y expectativas del cliente.
Mantenimiento y evaluación
Identifica y aplica soluciones de mantenimiento al software, mejorando su rendimiento y corrigiendo errores, al mismo tiempo que desarrolla un plan de evaluación que mida la calidad del software utilizando métricas como rendimiento, usabilidad y seguridad.

Ruta de aprendizaje (Estrategias didácticas)

PERIODO COMPRENDIDO: 07 de al 11 de octubre de 2024

Secuencia de actividades según la COMPETENCIA	Recursos (Materiales Didácticos y de información)	RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)
I. INICIO **Actividades del Docente:** • La docente inicia la clase explicando el objetivo de la sesión, que consiste en que los estudiantes comprendan cómo diseñar un sistema de software partiendo de los requerimientos identificados. Explica el propósito del diseño en el ciclo de desarrollo de software. • Utiliza una presentación multimedia para introducir las metodologías de diseño (diagrama de flujo de datos, modelo entidad-relación) y las herramientas que se emplearán para documentar los requerimientos. Muestra ejemplos reales de diseños de software. • Formula preguntas abiertas para que los estudiantes reflexionen sobre la importancia de la planificación en el diseño de sistemas, incentivando la participación mediante lluvia de ideas.	*Presentación en PowerPoint o Prezi sobre metodologías de diseño de software. *Ejemplos de diagramas de flujo de datos y modelos entidad-relación.	Explicación clara de los conceptos clave del diseño de software y la relación entre requerimientos y diseño.

II. DESARROLLO **Actividades del Docente:** • Divide a los estudiantes en equipos y les proporciona un caso práctico que requiere el diseño de un sistema de software. Cada equipo recibe una descripción del problema y los requerimientos del sistema. • Mientras los equipos trabajan en los diagramas de flujo y modelos entidad-relación, el docente circula entre los grupos, brindando retroalimentación y aclarando dudas. Les recuerda que deben justificar cada paso del diseño en función de los requerimientos dados. • Realiza una demostración en tiempo real utilizando un software de modelado (como LucidChart), mostrando cómo convertir un conjunto de requerimientos en un diseño visualmente claro y funcional. Explica cada paso y las decisiones que debe tomar el diseñador.	*Software de diagramación (LucidChart, Draw.io) para mostrar ejemplos en tiempo real. *Caso práctico para diseñar un sistema de software.	Supervisión y retroalimentación a los estudiantes durante la elaboración de los diagramas de flujo de datos y modelos entidad-relación.
III. CIERRE **Actividades del Docente:** • Reúne al grupo para que cada equipo presente su diseño, pidiendo que expliquen cómo llegaron a su solución y cómo sus decisiones impactarán el desarrollo del sistema. • El docente facilita una discusión grupal en la que se comparan los diferentes enfoques de diseño presentados por los equipos, destacando las fortalezas y áreas de mejora de cada uno. Resalta la importancia de seguir metodologías estándar. • Asigna una tarea individual donde los estudiantes deben investigar una metodología adicional de diseño de software, proporcionando un ejemplo de su aplicación.	*Pintarrón para facilitar la discusión grupal. *Ejemplos de mejores prácticas de diseño de software.	Facilita la discusión grupal y proporciona retroalimentación final sobre los diseños presentados.
NÚMERO DE HORAS: 2		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**
IV. TRABAJO ÁULICO **Actividad del Estudiante:**		

Inicio: • Los estudiantes escuchan atentamente la explicación sobre los objetivos de la sesión y la importancia del diseño de software. Realizan preguntas para aclarar cualquier duda sobre el proceso de diseño. • Participan activamente en la lluvia de ideas organizada por la docente, aportando su comprensión sobre la importancia del diseño como fase crítica en el desarrollo de software. **Desarrollo:** • En equipos, los estudiantes analizan el caso práctico proporcionado, identificando los requerimientos y empezando a dibujar los primeros esbozos de los diagramas de flujo de datos y modelos entidad-relación. • Mientras trabajan en el diseño, discuten entre ellos la mejor manera de estructurar el sistema, consultando a la docente cuando surgen dudas técnicas. Utilizan software especializado para crear sus diseños y aseguran que todos los miembros del equipo comprendan el proceso. • Documentan cada decisión que toman, explicando cómo sus elecciones impactarán la funcionalidad del sistema y garantizando que el diseño sea fácilmente implementable en fases posteriores. **Cierre:** • Presentan su diseño ante el resto del grupo, explicando los componentes clave y cómo su solución cumple con los requerimientos del caso práctico. • Participan en la discusión grupal posterior, comparando su diseño con los de los otros equipos y tomando nota de los comentarios del docente y sus compañeros para mejorar su comprensión del proceso de diseño. • Realizan la investigación individual asignada para la siguiente sesión.	*Libreta o cuaderno de apuntes para tomar notas. *Material de lectura (documento digital o impreso) sobre conceptos de diseño. *Computadora con acceso a internet y software de diagramación. *Documentación del caso práctico para analizar y desarrollar el diseño. *Computadora para preparar la presentación del diseño. *Material de soporte (notas y diagramas) para la presentación grupal.	Participación en la discusión inicial, preguntas sobre los conceptos y apuntes sobre la presentación: 10% Desarrollo de un diagrama de flujo de datos y modelo entidad-relación para el caso práctico asignado: 40% Presentación del diseño del sistema, explicando las decisiones tomadas y las justificaciones técnicas:50%
Ruta de aprendizaje (Estrategias didácticas)		
PERIODO COMPRENDIDO: 14 al 18 de octubre de 2024		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**

I. INICIO **Actividades del Docente:** • La docente introduce el tema explicando la transición del diseño al código, destacando la importancia de traducir correctamente los diagramas en código funcional. Presenta los conceptos clave como las estructuras de control (condicionales, ciclos) y variables. • Muestra en tiempo real cómo un diseño se traduce en código usando un lenguaje de programación (por ejemplo, Python o Java). Explica cada parte del código y cómo este refleja el diseño previo. • Formula preguntas orientadas a verificar si los estudiantes han comprendido la relación entre el diseño y la codificación.	*Presentación sobre buenas prácticas de codificación (estructuras de control, variables, modularidad). *Ejemplo en tiempo real de codificación basada en un diseño previo.	Explicación clara de los conceptos de codificación y su relación con el diseño de software.
II. DESARROLLO: **Actividades del Docente:** • Divide a los estudiantes en equipos y les asigna la tarea de implementar una parte del diseño que realizaron en la sesión anterior utilizando un lenguaje de programación. • Mientras los estudiantes trabajan en la codificación, el docente supervisa su progreso, ofreciendo asesoría cuando encuentran errores o dificultades técnicas. Fomenta el trabajo en equipo para que resuelvan problemas juntos. • Realiza pequeñas pausas para explicar con mayor detalle algunos conceptos clave, como la depuración de errores o la optimización del código. Usa ejemplos concretos para reforzar el aprendizaje.	*Entorno de desarrollo integrado (IDE) para mostrar ejemplos en tiempo real. *Ejemplo de código basado en el diseño proporcionado.	Supervisión del proceso de codificación, ofreciendo retroalimentación técnica y sugerencias de optimización.
III. CIERRE **Actividades del Docente:** • Los estudiantes presentan su código, mostrando cómo implementaron la lógica del diseño. La docente evalúa las presentaciones, enfocándose en la claridad del código, su funcionalidad y su adherencia al diseño original. • Facilita una discusión sobre los desafíos encontrados durante la codificación y las posibles soluciones a estos problemas. Da retroalimentación individualizada a cada equipo. • Asigna una tarea para que los estudiantes revisen y optimicen su código antes de la próxima sesión, aplicando buenas prácticas de programación.	*Herramienta digital para revisar los resultados de la codificación y los desafíos enfrentados.	Facilita la revisión del código y ofrece retroalimentación sobre la implementación.

NÚMERO DE HORAS: 2		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**
IV. TRABAJO ÁULICO **Actividades del Estudiante:** **Inicio:** • Los estudiantes participan en la introducción haciendo preguntas para entender cómo transformar el diseño en código. Toman notas sobre las estructuras de control y otras características del lenguaje de programación. • Observan atentamente la demostración en tiempo real del docente, tomando apuntes sobre cómo el diseño influye en la estructura del código. **Desarrollo:** • Trabajan en equipos para escribir el código basándose en el diseño que crearon en la sesión anterior. A medida que codifican, discuten la mejor manera de implementar la lógica del sistema y cómo evitar errores. • Cuando encuentran problemas técnicos, colaboran para resolverlos y piden asistencia al docente cuando es necesario. Usan el entorno de desarrollo integrado (IDE) para probar y depurar su código. • Documentan su progreso y las decisiones que tomaron para asegurarse de que el código sea legible y mantenible. **Cierre:** • Presentan su código al grupo, explicando cómo implementaron cada parte del diseño y qué desafíos encontraron durante la codificación. Escuchan la retroalimentación del docente y de sus compañeros. • Participan en la discusión grupal, reflexionando sobre cómo mejorar su código en el futuro y qué lecciones aprendieron durante el proceso de implementación.	*Libreta para tomar notas sobre las estructuras de control y buenas prácticas de programación. *Computadora con un IDE instalado (Visual Studio Code, PyCharm). *Código fuente de un proyecto para implementar. *Computadora para la presentación del código y las decisiones técnicas tomadas.	Preguntas y participación activa para aclarar conceptos sobre la relación entre diseño y codificación: 10% Implementación del código funcional basado en el diseño proporcionado: 50% Presentación grupal del código implementado, explicando cómo se basó en el diseño previo: 40%
Ruta de aprendizaje (Estrategias didácticas)		
PERIODO COMPRENDIDO: 21 al 31 de octubre de 2024		

Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**
I. INICIO **Actividades del Docente:** • El docente presenta los conceptos clave de pruebas de software, explicando la importancia de validar que el código desarrollado cumpla con los requerimientos y no tenga errores. Explica las diferentes fases de pruebas: unitarias, de integración, y de aceptación. • Muestra ejemplos de casos en los que se ha implementado correctamente un plan de pruebas. Utiliza una presentación multimedia para explicar cómo se diseñan casos de prueba y la diferencia entre pruebas manuales y automáticas. • Realiza preguntas a los estudiantes para verificar si comprenden las implicaciones de no realizar pruebas adecuadas en el ciclo de vida del software.	*Presentación sobre los conceptos de pruebas de software (unitarias, integración, aceptación). *Ejemplos de casos de prueba de software. *Pintarrón para explicar la importancia de las pruebas.	Explicación de las fases de pruebas de software y participación en la introducción.
II. DESARROLLO: **Actividades del Docente:** • Divide a los estudiantes en equipos y les asigna un conjunto de funciones previamente codificadas para que realicen pruebas unitarias. Explica paso a paso cómo documentar un caso de prueba, la ejecución de las pruebas y el reporte de los resultados. • El docente supervisa los equipos mientras ejecutan sus pruebas, resolviendo dudas sobre cómo simular entradas y salidas esperadas, y cómo utilizar herramientas automáticas de testing (como JUnit en Java o PyTest en Python). • Facilita un taller donde los estudiantes puedan ejecutar diferentes pruebas sobre el mismo código y discutir sus hallazgos. Explica cómo interpretar los errores encontrados y cómo documentar las soluciones propuestas.	*Herramientas de pruebas automatizadas (JUnit, PyTest). *Código de software para aplicar pruebas. *Pizarra para mostrar ejemplos de cómo se documentan las pruebas.	*Supervisión de la ejecución de pruebas unitarias e integración.

III. CIERRE **Actividades del Docente:** • Cada equipo presenta los resultados de las pruebas realizadas, describiendo qué errores detectaron, cómo los documentaron y qué soluciones recomiendan para corregirlos. • El docente da retroalimentación sobre la calidad de los casos de prueba creados y cómo los estudiantes podrían mejorar su enfoque en futuras sesiones. Reflexiona con ellos sobre la importancia de las pruebas en proyectos reales. • Asigna una tarea individual en la que los estudiantes deberán diseñar un conjunto de pruebas adicionales para otro código proporcionado por el docente.	*Pintarron para revisar los resultados de las pruebas y errores detectados. *Informe modelo para guiar la redacción de los hallazgos.	Facilitar la presentación de resultados y proporcionar retroalimentación final.
NÚMERO DE HORAS: 4		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**
IV. TRABAJO Áulico **Actividades del Estudiante:** **Inicio:** • Los estudiantes toman nota sobre los tipos de pruebas que se utilizan en el ciclo de vida del software y las fases que estas comprenden. Realizan preguntas para entender mejor los tipos de errores que se pueden encontrar durante las pruebas. • Participan activamente en la discusión sobre la importancia de las pruebas en el desarrollo de software, reflexionando sobre los riesgos de no probar adecuadamente un sistema antes de su entrega. **Desarrollo:** • En equipos, los estudiantes diseñan y ejecutan pruebas unitarias sobre funciones específicas del código proporcionado. Utilizan herramientas de prueba automatizadas y documentan cada prueba, indicando las entradas, salidas esperadas y resultados reales. • Analizan los resultados obtenidos, identificando qué funciones fallaron y proponiendo correcciones. Interactúan con sus compañeros para discutir posibles soluciones, y realizan preguntas al docente para aclarar cualquier duda.	*Libreta para tomar notas. *Material de lectura o ejemplos de pruebas de software *Computadora con acceso a las herramientas de pruebas.	Participación en la discusión inicial sobre la importancia de las pruebas: 10% *Desarrollo de casos de prueba y ejecución de pruebas en el software asignado: 50%

• Colaboran en la creación de un reporte detallado que describe los errores encontrados, el comportamiento del software y las recomendaciones para mejorar su funcionamiento. **Cierre:** • Presentan sus hallazgos al resto del grupo, explicando qué errores detectaron y cómo podrían corregirse. Reflexionan sobre cómo podrían mejorar sus casos de prueba en futuras sesiones. • Reciben retroalimentación docente y sus compañeros, y toman nota de cómo mejorar sus pruebas. • Realizan la tarea de diseñar un conjunto de pruebas adicionales para otro fragmento de código.	*Código fuente del software para realizar las pruebas. *Computadora para preparar la presentación de los resultados de las pruebas. *Informe de resultados y correcciones propuestas.	*Presentación de los resultados de las pruebas, con errores detectados y correcciones sugeridas: 40%
Ruta de aprendizaje (Estrategias didácticas)		
PERIODO COMPRENDIDO: 04 al 08 de noviembre de 2024		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**
I. INICIO **Actividades del Docente:** • La docente introduce el concepto de validación de software, diferenciándolo de las pruebas. Explica que la validación se enfoca en verificar que el software cumple con las necesidades del usuario y los requerimientos iniciales. • Utiliza un ejemplo práctico de validación en un entorno real para mostrar cómo se recolecta retroalimentación del usuario final y cómo se realizan ajustes en el software antes de su entrega final. • Formula preguntas para verificar que los estudiantes comprendan la diferencia entre validación y pruebas, y la importancia de involucrar al cliente o usuario final en este proceso.	*Presentación sobre validación de software y la importancia de la retroalimentación del usuario. *Ejemplos de validación en software real.	Explicación clara del proceso de validación y participación activa en la introducción.

II. DESARROLLO: **Actividades del Docente:** • Los estudiantes trabajan en equipos para realizar un ejercicio de validación basado en un caso de estudio proporcionado por la docente. Simulan ser tanto usuarios finales como desarrolladores, revisando si el software cumple con los requerimientos iniciales. • La docente circula entre los equipos, guiando la simulación de validación y explicando cómo deben recopilar la retroalimentación del usuario. Ayuda a los estudiantes a entender cómo documentar cualquier discrepancia entre el sistema entregado y los requerimientos iniciales. • Organiza un panel de discusión donde los equipos actúan como "clientes" y "desarrolladores" para discutir las observaciones sobre el sistema y los ajustes que se deben realizar antes de la entrega final.	*Guía para recolectar retroalimentación del usuario final. *Documentación de requerimientos iniciales del sistema para validar su cumplimiento.	Supervisión del proceso de validación y recolección de retroalimentación.
III. CIERRE **Actividades del Docente:** • Cada equipo presenta sus hallazgos sobre la validación del sistema, explicando si cumple o no con las expectativas del usuario. Discuten los ajustes recomendados y cómo los implementarían. • La docente reflexiona con el grupo sobre los resultados de la validación y la importancia de este paso en el ciclo de desarrollo del software. Da retroalimentación sobre cómo mejorar la recolección de retroalimentación del usuario. • Asigna una tarea en la que los estudiantes deben crear un plan de validación para un nuevo sistema, detallando los pasos a seguir y cómo se recolectará la retroalimentación del usuario.	*Ejemplos de mejoras en software basado en retroalimentación del usuario. *Pizarra para discutir los resultados de la validación.	Facilitar la discusión sobre los resultados de la validación.
NÚMERO DE HORAS: 2		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**
IV. TRABAJO AULÍCO **Actividades del Estudiante:** **Inicio:**		Preguntas y participación

• Los estudiantes escuchan atentamente la explicación de la docente sobre el proceso de validación, haciendo preguntas para aclarar las diferencias entre pruebas y validación. • Reflexionan sobre la importancia de validar que el sistema cumpla con los requerimientos del usuario final y las consecuencias de no hacer este paso adecuadamente. **Desarrollo:** • En equipos, los estudiantes simulan el proceso de validación, actuando como usuarios finales y evaluando si el software cumple con sus expectativas. Documentan cualquier diferencia entre los requerimientos y el software entregado. • Colaboran para proponer mejoras y ajustes al sistema basado en la retroalimentación obtenida durante la validación. Elaboran un informe detallado de los resultados de la validación, destacando las áreas a mejorar. • Participan en la simulación del panel de discusión, actuando como "clientes" o "desarrolladores" y defendiendo sus observaciones y recomendaciones. **Cierre:** • Presentan sus resultados al grupo, explicando si el software cumple con los requerimientos y qué ajustes proponen antes de la entrega final. Reflexionan sobre las lecciones aprendidas durante la validación. • Reciben retroalimentación del docente y de sus compañeros, y reflexionan sobre cómo mejorar el proceso de validación. • Desarrollan un plan de validación como tarea individual para un sistema nuevo.	*Material de lectura sobre validación de software. Libreta para tomar notas. *Computadora para la creación del informe de validación. *Documentación de pruebas y retroalimentación del usuario final. *Computadora para preparar la presentación de los resultados de la validación. *Informe final con los hallazgos y recomendaciones.	en la discusión inicial: 10% Informe de validación detallando los resultados y las observaciones del usuario: 50% Presentación de los hallazgos y recomendaciones para mejorar el software: 40%

Ruta de aprendizaje (Estrategias didácticas)		
PERIODO COMPRENDIDO: 11 al 22 de noviembre de 2024		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**

I. INICIO **Actividades del Docente:** • La docente introduce la relación entre el **mantenimiento** y la **evaluación** de software, explicando que ambas fases son cruciales para garantizar que el software funcione adecuadamente a largo plazo y cumpla con los requerimientos establecidos. • Explica que el **mantenimiento** de software incluye correcciones de errores (correctivo), adaptaciones a nuevos entornos (adaptativo), prevención de futuros problemas (preventivo), y mejoras continuas (perfectivo), mientras que la **evaluación** permite medir la calidad y efectividad del software a través de métricas específicas como el rendimiento, la seguridad, y la satisfacción del usuario. • Muestra ejemplos prácticos de software que ha pasado por varios ciclos de mantenimiento y evaluación, explicando cómo estas fases son clave para la mejora continua del software. Se usan casos reales para ilustrar cómo el mantenimiento y la evaluación trabajan en conjunto. • Formula preguntas para que los estudiantes reflexionen sobre cómo el mantenimiento ayuda a que un software continúe siendo funcional y eficiente, y cómo la evaluación permite identificar las áreas que requieren mantenimiento. •	*Presentación sobre tipos de mantenimiento y métricas de evaluación. *Ejemplos de software que ha pasado por procesos de mantenimiento y evaluación.	Explicación de las diferencias entre los tipos de mantenimiento y la importancia de la evaluación continua.
II. DESARROLLO **Actividades del Docente:** • Los estudiantes trabajan en equipos para realizar tanto el mantenimiento como la evaluación de un software previamente desarrollado. El docente proporciona un código con áreas que necesitan mantenimiento (por ejemplo, correcciones de errores o mejoras en el rendimiento) y también métricas para evaluar la calidad del software. • Cada equipo debe identificar los aspectos del software que requieren mantenimiento. El docente guía a los estudiantes en cómo priorizar las tareas de mantenimiento (correctivo, preventivo, adaptativo, perfectivo), brindando retroalimentación sobre sus decisiones. • Simultáneamente, la docente introduce un **plan de evaluación** que los estudiantes deben seguir. El plan incluye la medición de métricas como tiempo de respuesta, eficiencia de recursos, usabilidad, y satisfacción del usuario. Los equipos deben realizar pruebas y medir el software utilizando herramientas	*Herramientas de control de versiones (Git) para realizar el mantenimiento. *Herramientas de medición de rendimiento y usabilidad (JMeter, LoadRunner).	Supervisión de la implementación de soluciones de mantenimiento y la ejecución del plan de evaluación.

automáticas (como JMeter para pruebas de rendimiento o herramientas de análisis de usabilidad). • La docente proporciona una demostración de cómo utilizar herramientas de control de versiones (como Git) para gestionar los cambios durante el mantenimiento y la evaluación del software. Esto incluye registrar cambios en el código, realizar pruebas antes y después de las modificaciones, y generar reportes con los resultados de las evaluaciones.		
III. CIERRE **Actividades del Docente:** • Los equipos presentan sus resultados, describiendo las tareas de mantenimiento que realizaron y cómo impactaron el rendimiento del software. Explican las métricas utilizadas en la evaluación, mostrando las áreas donde el software cumple con los estándares y aquellas que requieren mejoras. • La docente facilita una discusión final en la que se reflexiona sobre los retos del mantenimiento y la evaluación en proyectos reales de software, destacando cómo estos procesos son interdependientes y esenciales para la mejora continua. • Se asigna una tarea para que los estudiantes diseñen un plan de mantenimiento y evaluación para otro software, detallando los tipos de mantenimiento que aplicarían, las métricas de evaluación, y las herramientas que usarían.	*Pizarra para revisar los resultados del mantenimiento y la evaluación. *Ejemplos de software mejorado a través del mantenimiento.	Facilita la discusión final sobre los resultados del mantenimiento y la evaluación.
NÚMERO DE HORAS: 4		
Secuencia de actividades según la COMPETENCIA	**Recursos (Materiales Didácticos y de información)**	**RÚBRICA: Evidencias / Criterios de Evaluación (Indicar % de Eval.)**
IV. TRABAJO AULÍCO **Actividades del Estudiante:** **Inicio:** • Los estudiantes toman notas sobre las fases del mantenimiento y evaluación de software, formulando preguntas para clarificar cualquier duda. Reflexionan sobre cómo ambas fases están conectadas y cómo se complementan para mejorar la calidad del software a largo plazo.	Libreta para notas. Material de lectura	Participación en la discusión sobre los diferentes tipos de mantenimiento: 10%

• Participan en la discusión inicial, aportando ejemplos de software que usan regularmente y cómo este ha sido actualizado o evaluado para mantenerse funcional. **Desarrollo:** • En equipos, los estudiantes identifican áreas del software que necesitan mantenimiento. Discuten entre ellos qué tipo de mantenimiento es más adecuado para cada situación (correctivo, preventivo, adaptativo, perfectivo) y aplican las soluciones necesarias. • Simultáneamente, desarrollan un plan de evaluación para medir el rendimiento y la usabilidad del software utilizando métricas específicas. Implementan herramientas de medición para recolectar datos sobre el tiempo de respuesta, uso de recursos, y satisfacción del usuario. • Los estudiantes documentan el proceso de mantenimiento y evaluación, detallando los cambios realizados y los resultados de las métricas obtenidas. Utilizan herramientas de control de versiones (Git) para registrar los cambios y asegurar que el trabajo colaborativo sea organizado y efectivo. **Cierre:** • Presentan sus resultados al grupo, explicando las soluciones de mantenimiento implementadas y cómo estas afectaron la calidad del software. Comparan los resultados de sus evaluaciones, destacando las áreas de mejora y discutiendo cómo mejorar el proceso de mantenimiento y evaluación en futuros proyectos. • Participan en la discusión final, reflexionando sobre cómo el mantenimiento y la evaluación son procesos continuos que permiten que el software se adapte a nuevos requerimientos y siga siendo útil a lo largo del tiempo. • Realizan la tarea de diseñar un plan de mantenimiento y evaluación para otro software, aplicando los conocimientos adquiridos en la sesión.	sobre mantenimiento y evaluación de software. Computadora con acceso a herramientas de control de versiones y medición. Código fuente para realizar mantenimiento y evaluación. Computadora para la presentación final de los resultados del mantenimiento y la evaluación. Informe final con los hallazgos y propuestas de mejora.	Implementación de soluciones de mantenimiento y ejecución de las métricas de evaluación: 50% Presentación grupal de los hallazgos y resultados del mantenimiento y evaluación: 40%

Descripción del Trabajo por Colegio de Grado (Indicar % de Evaluación):

(EJEMPLO)

Trabajo o Producto Final Integrador de la o las Competencias del o los Bloques:

Aplicación del Formato de Requerimientos Basado en el Análisis de Sistemas

Bibliografía/Cibergrafía Recomendada

Bibliografía para el Alumno	Bibliografía para el Docente
Ceballos Sierra, A. (2016). *Desarrollo de software basado en componentes*. Alfaomega. **Deitel, P., & Deitel, H.** (2017). *Cómo programar en Java* (10ª ed.). Pearson Educación. **Gómez, G. A.**, & Blanco, F. C. (2017). *Diseño y programación de software*. Alfaomega. **Jiménez, M.** (2015). *Pruebas de software: Cómo mejorar la calidad del software mediante pruebas eficaces*. Alfaomega. **Joyanes Aguilar, L.** (2014). *Fundamentos de Programación: Algoritmos, estructuras de datos y objetos* (5ª ed.). McGraw-Hill. **Pressman, R. S.** (2010). *Ingeniería de software: Un enfoque práctico* (7ª ed.). McGraw-Hill. **Rubio, R.** (2018). *Metodologías ágiles: Desarrollo de software en tiempos de cambio*. Ra-Ma Editorial.	**Arcega, M., & Prieto, J.** (2019). *Mantenimiento de Software: Co aplicaciones*. Alfaomega. **Gómez, G. A.**, & Blanco, F. C. (2017). *Diseño y programación de* Alfaomega. **Hernández, R., Fernández, C., & Baptista, P.** (2014). *Metodolog investigación* (6ª ed.). McGraw-Hill. **Jiménez, M.** (2018). *Pruebas de software: Un enfoque práctico*. **Pressman, R. S., & Maxim, B. R.** (2018). *Ingeniería de Software: práctico*. McGraw-Hill. **Sommerville, I.** (2016). *Ingeniería de Software* (10ª ed.). Pearso

Atentamente: Docente del Grupo(s): Beatriz Morales Cruz

Observaciones:	Vo. Bo. Subdirección Académica ADRIÁN ANDRADE ALMANZA Autorizado

Cd. Nezahualcóyotl, Estado de México, a 3 de Octubre de 2024.

ANEXOS

Tema 1: Diseño de Software

Instrumento 1: Rúbrica para el Diagrama de Flujo de Datos y Modelo Entidad-Relación

Criterio	Excelente (4 pts.)	Bueno (3 pts.)	Satisfactorio (2 pts.)	Insuficiente (1 pt.)
Claridad y organización del diseño	El diagrama es claro, bien organizado y sigue una secuencia lógica.	El diagrama está bien organizado, con pequeños errores en la secuencia lógica.	El diagrama tiene varios errores de organización y secuencia.	El diagrama es confuso, con muchas fallas en la secuencia lógica.
Precisión técnica	El diseño cubre todos los requerimientos del sistema de manera precisa.	El diseño cubre la mayoría de los requerimientos con algunos errores menores.	El diseño cubre pocos requerimientos y tiene errores técnicos significativos.	El diseño no cubre los requerimientos y presenta graves errores técnicos.
Creatividad e innovación	El diseño presenta soluciones creativas e innovadoras.	El diseño es funcional con algunas propuestas innovadoras.	El diseño es básico y funcional, sin elementos innovadores.	El diseño es deficiente y no propone soluciones funcionales o innovadoras.
Documentación del proceso	El diseño está completamente documentado, explicando todas las decisiones tomadas.	El diseño está bien documentado, con algunas decisiones no explicadas.	El diseño tiene poca documentación y no explica muchas decisiones.	El diseño carece de documentación que justifique las decisiones.

Tema 2: Codificación de Software

Instrumento 2: Lista de Cotejo para la Codificación

Criterio	Sí (1 pt.)	No (0 pts.)
El código cumple con los requerimientos funcionales.		
El código está correctamente estructurado y es legible.		
El código incluye comentarios que explican cada parte.		
El código utiliza buenas prácticas de programación (modularidad, reutilización).		
El código ha sido probado y funciona correctamente.		
Se documentan correctamente las bibliotecas y herramientas usadas.		

Tema 3: Pruebas de Software

Instrumento 3: Rúbrica para el Diseño y Ejecución de Pruebas

Criterio	Excelente (4 pts.)	Bueno (3 pts.)	Satisfactorio (2 pts.)	Insuficiente (1 pt.)
Diseño de casos de prueba	Los casos de prueba cubren todos los aspectos funcionales del software de manera detallada.	La mayoría de los aspectos funcionales están cubiertos, pero algunos detalles están ausentes.	Los casos de prueba cubren sólo algunos aspectos funcionales de manera superficial.	Los casos de prueba son insuficientes y no cubren aspectos importantes.
Ejecución de las pruebas	Las pruebas se ejecutan correctamente y se documentan todos los resultados con precisión.	La ejecución de pruebas es buena, pero algunos resultados no están bien documentados.	La ejecución de pruebas tiene errores y la documentación es incompleta.	Las pruebas no se ejecutan correctamente y la documentación es deficiente.
Identificación de errores	Todos los errores son detectados y documentados claramente.	La mayoría de los errores son detectados y documentados, con algunas omisiones.	Se detectan pocos errores y la documentación es incompleta.	No se detectan errores o la documentación es muy deficiente.
Propuestas de corrección	Las propuestas de corrección son claras y viables, alineadas con los resultados obtenidos.	Las propuestas son claras, pero no siempre viables o alineadas con los resultados.	Las propuestas son básicas y no abordan completamente los problemas detectados.	No se proponen soluciones adecuadas o viables.

Tema 4: Validación de Software

Instrumento 4: Lista de Cotejo para la Validación de Software

Criterio	Sí (1 pt.)	No (0 pts.)
Se recopila retroalimentación detallada del usuario final.		
La retroalimentación cubre todas las funcionalidades y aspectos clave del software.		
Se comparan los resultados del software con los requerimientos iniciales.		
Las observaciones del usuario final son documentadas correctamente.		
Las propuestas de mejora están alineadas con las observaciones del usuario.		
El informe de validación está bien estructurado y documentado.		

Temas 5 y 6: Mantenimiento y Evaluación de Software

Instrumento 5: Rúbrica para el Mantenimiento y Evaluación del Software

Criterio	Excelente (4 pts.)	Bueno (3 pts.)	Satisfactorio (2 pts.)	Insuficiente (1 pt.)
Análisis de mantenimiento	El análisis es detallado, identificando todas las áreas que requieren mantenimiento.	El análisis identifica la mayoría de las áreas, pero omite algunos aspectos menores.	El análisis cubre algunas áreas, pero es superficial.	El análisis es incompleto y no identifica las áreas clave para mantenimiento.
Implementación de soluciones de mantenimiento	Las soluciones implementadas corrigen todos los problemas y mejoran el rendimiento.	Las soluciones son buenas, pero no corrigen todos los problemas.	Las soluciones implementadas son funcionales pero tienen muchas limitaciones.	Las soluciones no corrigen los problemas o empeoran el rendimiento del software.
Plan de evaluación	El plan de evaluación cubre todas las métricas clave y está bien estructurado.	El plan de evaluación es bueno, pero omite algunas métricas importantes.	El plan de evaluación cubre solo las métricas básicas de forma superficial.	El plan de evaluación está incompleto y mal estructurado.
Documentación de resultados de la evaluación	Los resultados de la evaluación están bien documentados, y se proponen mejoras claras.	Los resultados están documentados, pero las propuestas de mejora no son claras.	Los resultados son vagos y la documentación es insuficiente.	Los resultados no están bien documentados ni se proponen mejoras claras.
Uso de herramientas de control de versiones	Se utiliza correctamente el control de versiones, documentando todos los cambios.	Se usa el control de versiones, pero la documentación es incompleta.	Se usa poco el control de versiones, con una documentación limitada.	No se usa adecuadamente el control de versiones o no hay documentación.