ISTRED
INSTITUTO SUPERIOR TÉCNICO REY DAVID

PERIODO I - 2024 - 2025

GUÍA DEL APRENDIZAJE DEL ESTUDIANTE

METODOLOGÍA DE DISEÑO DE SOFTWARE

ING. ANTONY HERNÁNDEZ LEÓN

GESTIÓN DE BASES DE DATOS
2DO. SEMESTRE

Contenido

I. DATOS GENERALES DE LA ASIGNATURA 6

Información General 6

Requisitos. 6

Descripción de la asignatura. 6

Objetivo general 6

Introducción de la Asignatura 7

I. DATOS GENERALES DE LA UNIDAD 1 8

Información General 8

Resultado de Aprendizaje de Unidad 8

Metodología de Unidad 8

DESARROLLO 9

Unidad 1: Introducción a las Metodologías de Desarrollo 9

Tema 1: Introducción al desarrollo 9

Historia 10

Conceptos Fundamentales 12

Ciclo de Desarrollo de software 14

Importancia de la metodología de desarrollo aplicada a proyectos 16

ACTIVIDADES PRÁCTICAS 18

FORO DE DEBATE 19

PRÁCTICA INVESTIGATIVA 19

PRÁCTICA TEST 20

Tema 2: Procesos de desarrollo 21

Procesos en el ciclo de desarrollo 21
Recolección y análisis de requisitos 21
Características de los procesos 23
Proceso liviano vs Procesos Pesados. 25
Proceso Liviano: 25
Proceso Pesado: 26
Roles en el desarrollo de software. 27
ACTIVIDADES PRÁCTICAS 30
FORO DE DEBATE 31
PRÁCTICA INVESTIGATIVA 31
PRÁCTICA TEST 32
II. DATOS GENERALES DE LA UNIDAD 2 33
Información General 33
Resultado de Aprendizaje de Unidad 33
Metodología 33
DESARROLLO 34
Unidad 2: Metodologías de Desarrollo Tradicionales 34
Tema 1: Metodologías 34
Metodología pesada RUP 35
Historia de RUP 37
Características principales de RUP 38
ACTIVIDADES PRÁCTICAS 41
FORO DE DEBATE 42

PRÁCTICA INVESTIGATIVA 42
PRÁCTICA TEST 43
Tema 2: Estructura del Proceso 44
Proceso Centrado en la Arquitectura 44
Proceso Iterativo e Incremental 44
Otras Prácticas RUP 45
Estructura del Proceso 46
ACTIVIDADES PRÁCTICAS 47
FORO DE DEBATE 48
PRÁCTICA INVESTIGATIVA 48
PRÁCTICA TEST 49
III. DATOS GENERALES DE LA UNIDAD 3 50
Información General 50
Resultado de Aprendizaje de Unidad 50
Metodología 50
DESARROLLO 51
Unidad N.3: Metodologías de Desarrollo Ágiles 51
Tema 1: Metodología XP 51
Introducción 51
Características Principales de Extreme Programming (XP): 52
Proceso en XP: 53
Gestión y Seguimiento en XP: 56
ACTIVIDADES PRÁCTICAS 57

FORO DE DEBATE 58
PRÁCTICA INVESTIGATIVA 58
PRÁCTICA TEST 59
Tema 2: Metodología SCRUM 60
Concepto y Características 60
Características Principales de Metodología SCRUM: 61
Requisitos y Roles: 64
Gestión y Seguimiento en Scrum: 65
ACTIVIDADES PRÁCTICAS 66
FORO DE DEBATE 67
PRÁCTICA INVESTIGATIVA 67
PRÁCTICA TEST 68
IV. DATOS GENERALES DE LA UNIDAD 4 69
Información General 69
Resultado de Aprendizaje de Unidad 69
Metodología de Unidad 69
DESARROLLO 70
Unidad 4: Aplicación de una metodología de desarrollo de software según tendencia 70
Tema 1: La Aplicabilidad 70
Preparación del Proyecto: 70
Planificar un Sprint 71
Ejemplo con Caso de Estudio: 71

ACTIVIDADES PRÁCTICAS 73
FORO DE DEBATE 74
PRÁCTICA INVESTIGATIVA 74
PRÁCTICA TEST 75
Tema 2: El Desarrollo 76
El desarrollo del sprint 76
Diagrama detallado de las fases de la metodología elegida 76
ACTIVIDADES PRÁCTICAS 78
FORO DE DEBATE 79
PRÁCTICA INVESTIGATIVA 79
PRÁCTICA TEST 80
V. Bibliografía 81
Bibliografía básica 81
Bibliografía de consulta 81
Anexo 1: Modelo de Foro 82
Anexo 2: Esquema de Ensayo 83
Anexo 3: Esquema de Taller/Investigación 84
Anexo 4: Rúbrica de Foro 85
Anexo 5: Rubrica de Ensayo 86
Anexo 6: Rúbrica de Taller/Investigación 87

I. DATOS GENERALES DE LA ASIGNATURA

Información General

1.1.	Asignatura	Metodología de Diseño de Software					
1.2.	Carrera	Gestión de Bases de Datos					
1.3.	Código de asignatura	GBD22312					
1.4.	Créditos	1,2					
1.5.	Nivel	Segundo					
1.6.	Detalle de horas	ACD	24	AA	16	APE	20

Requisitos.

Prerrequisito		Correquisito	
Asignatura	Código	Asignatura	Código
Lógica de Sistemas	GBD1233	N/A	N/A

Descripción de la asignatura.

En esta asignatura se exploran diversos enfoques y técnicas para el diseño de software, así como la importancia de seguir un proceso metodológico sólido. Los estudiantes aprenden a identificar requerimientos, definir arquitecturas, seleccionar herramientas y tecnologías apropiadas, y gestionar proyectos de desarrollo de software. Además, se enfatiza la importancia de la colaboración en equipos y la comunicación efectiva en todo el ciclo de desarrollo. Esta asignatura prepara a los estudiantes para convertirse en profesionales capaces de abordar proyectos de desarrollo de software de manera estructurada y exitosa.

Objetivo general

Desarrollar en el estudiante de Gestión de Bases de Datos el conocimiento general de los diversos tipos de metodologías de diseño de software, asociados a las soluciones de tipos sistematizadas que ayuden a la resolución de problemas comunes y no comunes.

Introducción de la Asignatura

La tecnología de la información y la informática desempeñan un papel fundamental en nuestro mundo contemporáneo. En el corazón de esta revolución se encuentra el desarrollo de software, un proceso complejo que requiere una metodología estructurada y efectiva para asegurar el éxito de los proyectos. En esta asignatura, exploraremos las diversas metodologías de desarrollo de software que han evolucionado a lo largo del tiempo. Comenzando con una sólida base en la Unidad N.1, "Introducción a las Metodologías de Desarrollo", adentrándonos en la historia, conceptos principales y el ciclo de desarrollo de software, nuestros estudiantes comprenderán la importancia de aplicar metodologías a proyectos de software.

Se presentarán los procesos de desarrollo y se analizarán las características que los definen. Además, se explorará la diferencia entre procesos livianos y pesados, y se destacarán los roles fundamentales en el desarrollo de software. Todo ello con un fuerte énfasis en los fundamentos teóricos de la Metodología de Diseño de Software (MDS). A medida que avancemos a lo largo del curso, profundizaremos en metodologías tanto tradicionales como ágiles, como RUP, XP y Scrum, con el objetivo de comprender sus conceptos, características, procesos y aplicaciones prácticas.

Aplicaremos este conocimiento a la realidad, preparándonos para proyectos reales. Aprenderemos a planificar sprints y a gestionar el desarrollo en consonancia con la metodología elegida. La asignatura "Metodología de Diseño de Software" será una valiosa introducción al mundo del desarrollo de software, equipando a nuestros estudiantes con las herramientas y el conocimiento necesarios para abordar proyectos de software de manera exitosa.

I. DATOS GENERALES DE LA UNIDAD 1

Información General

1.1.	Nivel	Segundo					
1.2.	Nombre Unidad	Introducción a las Metodologías de Desarrollo					
1.3.	Tema 1 – Unidad	Introducción al desarrollo					
1.4.	Tema 2 – Unidad	Procesos de desarrollo					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	1					
1.7.	Total, Horas Unidad	15					
1.8.	Detalle de horas Unidad	ACD	6	AA	4	APE	5

Resultado de Aprendizaje de Unidad

En la unidad "Introducción a las Metodologías de Desarrollo", los estudiantes lograrán un entendimiento fundamental de las diversas metodologías utilizadas en el desarrollo de software. Al finalizar la unidad, los estudiantes serán capaces de identificar y describir las características clave de metodologías como el modelo en cascada, el desarrollo ágil, el desarrollo iterativo y la programación extrema (XP). Además, serán capaces de explicar cómo y cuándo se aplican estas metodologías en el ciclo de vida del desarrollo de software, destacando las ventajas y desventajas de cada enfoque. Este conocimiento permitirá a los estudiantes tener una visión global de las estrategias disponibles para abordar diferentes tipos de proyectos de desarrollo.

Metodología de Unidad

La Metodología de se basa en un enfoque activo y colaborativo. Incluye sesiones de clase estructuradas en torno a temas específicos, lecturas y tareas, estudio de casos, proyectos prácticos y debates en clase. Los estudiantes también se involucran en trabajos en grupo y mantienen diarios de reflexión. Se fomenta la participación en debates y preguntas y respuestas, y se proporcionan recursos en línea para una comprensión más profunda. La evaluación se realiza mediante pruebas, proyectos y una evaluación final, con retroalimentación continua para guiar el aprendizaje.

DESARROLLO

Unidad 1: Introducción a las Metodologías de Desarrollo

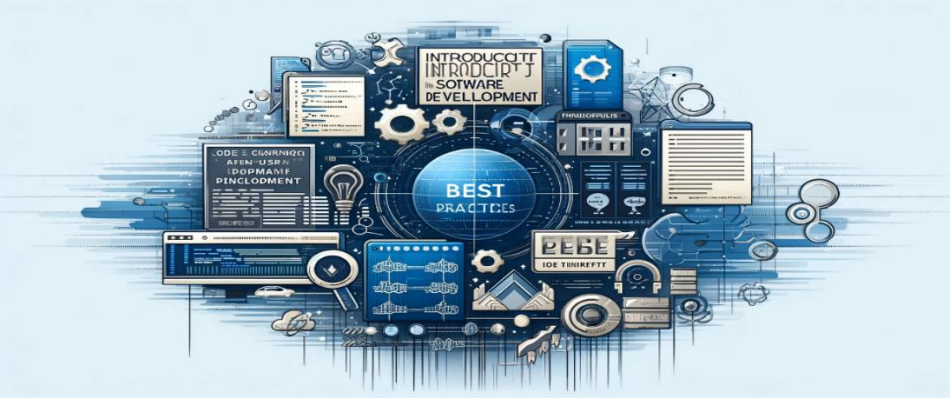
Tema 1: Introducción al desarrollo

Objetivo: proporcionar a los estudiantes una comprensión sólida y fundamentada de los conceptos esenciales relacionados con el desarrollo de software y las metodologías que se utilizan en este campo

El desarrollo de software es un campo que ha evolucionado significativamente a lo largo de las décadas. En sus inicios, la programación de computadoras implicaba la escritura de código máquina, un proceso laborioso y propenso a errores. Con el tiempo, se desarrollaron lenguajes de programación de alto nivel que permitían a los programadores expresar instrucciones de manera más comprensible y eficiente. Esta evolución marcó el comienzo del desarrollo de software moderno.

Los conceptos clave en el desarrollo de software incluyen la programación, la arquitectura de software, la gestión de proyectos, la calidad del software y las metodologías de desarrollo. Estos conceptos son fundamentales para comprender cómo se construyen aplicaciones informáticas de manera efectiva y eficiente.

Imagen1: Metodología del desarrollo de Software



Tomado de: https://lovtechnology.com/introduccion-al-desarrollo-de-software-principios-y-mejores-practicas/

A lo largo de las décadas, el desarrollo de software ha pasado por varias etapas clave, desde los primeros días de la programación hasta la era actual de metodologías ágiles y la nube. A continuación, se presenta una historia amplia del desarrollo de software:

Historia

Década de 1940: Los Inicios

La historia del desarrollo de software comienza con la creación de las primeras computadoras electrónicas durante la Segunda Guerra Mundial. Los primeros programadores trabajaban directamente con el hardware, escribiendo código de máquina para ejecutar tareas específicas. Uno de los primeros lenguajes de programación, el "Short Code", permitía a los programadores usar abreviaciones para escribir instrucciones, allanando el camino para lenguajes de alto nivel.

Década de 1950: Lenguajes de Programación

La década de 1950 marcó el inicio de los lenguajes de programación de alto nivel, como el Fortran y el LISP. Estos lenguajes permitieron a los programadores escribir código más legible y reutilizable, lo que revolucionó la programación. La idea de compilar código fuente en código máquina se convirtió en una práctica común.

Década de 1960: Desarrollo Estructurado

La década de 1960 vio el nacimiento del desarrollo estructurado, que promovía la organización lógica del código a través de técnicas como el diseño modular y la programación orientada a objetos. Se crearon los primeros sistemas operativos y bases de datos.

Década de 1970: Los Microprocesadores

La disponibilidad de microprocesadores permitió la creación de computadoras personales. Se desarrollaron los primeros sistemas operativos de tiempo compartido y se popularizaron los lenguajes de programación como el C.

Década de 1980: GUI y Computadoras Personales

La década de 1980 fue testigo del auge de las interfaces gráficas de usuario (GUI) con la introducción de computadoras personales como la IBM PC y el sistema operativo Windows. Los lenguajes de programación como C++ y el desarrollo orientado a objetos se convirtieron en tendencia.

Década de 1990: Internet y la Web

La década de 1990 fue revolucionaria con la proliferación de Internet y la World Wide Web. El desarrollo de aplicaciones web y lenguajes como Java y JavaScript cambió la forma en que las personas interactúan con el software. Las metodologías de desarrollo ágil comenzaron a emerger.

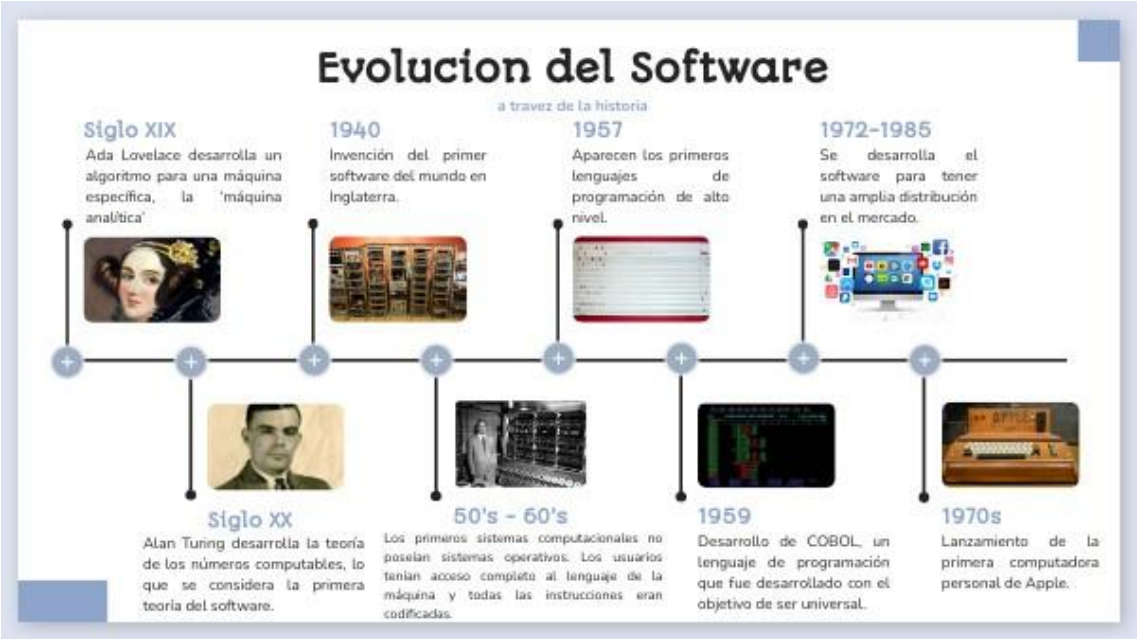
Década de 2000: Software Empresarial y Móvil

El auge del software empresarial y las aplicaciones móviles definió la década de 2000. Se desarrollaron sistemas ERP (Enterprise Resource Planning) y CRM (Customer Relationship Management). La popularización de los teléfonos inteligentes impulsó el desarrollo de aplicaciones móviles.

Década de 2010: La Nube y Metodologías Ágiles

La computación en la nube y las metodologías ágiles como Scrum y DevOps ganaron protagonismo. Las organizaciones adoptaron servicios en la nube para hospedar aplicaciones y datos. El desarrollo de software se volvió más colaborativo y centrado en la entrega continua.

Imagen2: Historia del Software



Tomado de: https://view.genially.com/65b6a3734dce970014889874/interactive-content-linea-del-tiempo-del-software

Década Actual: IA y Desarrollo Sostenible

En la actualidad, el desarrollo de software está en constante evolución con enfoques en inteligencia artificial, aprendizaje automático y sostenibilidad. Se continúa innovando en la forma en que se construyen y entregan aplicaciones, y se abordan desafíos globales como la seguridad y la privacidad de los datos.

La historia del desarrollo de software es una muestra del rápido avance tecnológico y cómo la programación ha pasado de ser un arte oscuro a una disciplina altamente estructurada y colaborativa que impulsa la revolución digital. Cada etapa ha dejado su huella en la forma en que interactuamos con la tecnología y ha dado forma al mundo tal como lo conocemos hoy en día.

Conceptos Fundamentales

En el desarrollo de software, existen conceptos importantes que sirven como pilares fundamentales para comprender y llevar a cabo proyectos de manera exitosa. Aquí te proporciono algunas conceptualizaciones clave:

Requerimientos de Software: Los requerimientos son especificaciones detalladas de las funcionalidades y características que debe tener el software. Comprender y documentar los requerimientos de manera precisa es esencial para el éxito del proyecto.

Ciclo de Vida del Software: El ciclo de vida del software se refiere al proceso completo de desarrollo, desde la concepción de la idea hasta su despliegue, mantenimiento y posible retirada. Comprender este ciclo es crucial para planificar y gestionar el proyecto.

Metodologías de Desarrollo: Las metodologías son enfoques estructurados para el desarrollo de software. Ejemplos incluyen el modelo en cascada, las metodologías ágiles (como Scrum y Kanban), y otras. Cada metodología tiene sus propias reglas y prácticas.

Diseño de Software: El diseño de software se refiere a la fase en la que se crean estructuras y arquitecturas para el software. Esto incluye el diseño de la interfaz de usuario, la arquitectura del sistema y la planificación de la base de datos.

Desarrollo Ágil: El desarrollo ágil es un enfoque colaborativo y flexible que se centra en la entrega continua de software funcional en ciclos cortos (sprints). Se valora la adaptabilidad y la comunicación constante con los interesados.

Control de Calidad de Software: El control de calidad se refiere a las prácticas que garantizan que el software cumpla con estándares y requisitos de calidad. Esto incluye pruebas, revisión de código y aseguramiento de la calidad.

Gestión de Proyectos de Software: La gestión de proyectos de software implica la planificación, programación, seguimiento y control de todas las actividades relacionadas con el desarrollo de software. Las metodologías como el enfoque PMI y Agile PM son comunes en este ámbito.

Documentación de Software: La documentación es esencial para comprender y mantener el software a lo largo del tiempo. Esto incluye manuales de usuario, documentación técnica y comentarios en el código.

Seguridad de Software: La seguridad del software implica la identificación y mitigación de riesgos relacionados con la seguridad, como vulnerabilidades y amenazas. La seguridad es esencial en el mundo digital actual.

Integración Continua y Entrega Continua (CI/CD): CI/CD son prácticas que automatizan las pruebas y la entrega de software, lo que permite una implementación más rápida y confiable de las nuevas funcionalidades.

Mantenimiento de Software: El mantenimiento del software se refiere a la fase en la que se corrigen errores, se aplican actualizaciones y se realizan mejoras en el software ya en funcionamiento.

Arquitectura de Software: La arquitectura del software se refiere a la estructura general y la organización de los componentes del software. Una arquitectura sólida es esencial para la escalabilidad y el rendimiento.

Desarrollo Sostenible de Software: Este concepto implica considerar la sostenibilidad ambiental y ética en el desarrollo de software, minimizando el impacto negativo en el medio ambiente y promoviendo prácticas responsables.

Ciclo de Desarrollo de software

El ciclo de desarrollo de software es un proceso que describe las fases y actividades que se realizan para crear, diseñar, construir, probar, implementar y mantener un sistema de software. Este ciclo guía el desarrollo de software desde su concepción hasta su puesta en producción y su posterior mantenimiento. A continuación, se detallan las principales fases y conceptos que componen el ciclo de desarrollo de software:

Concepción: Esta fase marca el inicio del ciclo de desarrollo. En esta etapa, se identifica la necesidad de un nuevo sistema de software o una mejora en uno existente. Se recopilan los primeros requerimientos y se realiza una evaluación inicial de la viabilidad del proyecto.

Requerimientos: En esta fase, se definen y documentan los requerimientos del software. Esto incluye la identificación de las funcionalidades, características y restricciones del sistema. Los requerimientos pueden ser funcionales (lo que el sistema debe hacer) y no funcionales (como el rendimiento, la seguridad y la escalabilidad).

Diseño: La fase de diseño se enfoca en la creación de una arquitectura y una estructura para el software. Se planifica la distribución de componentes, la interfaz de usuario y la base de datos. El diseño se basa en los requerimientos recopilados en la fase anterior.

Implementación: En esta etapa, se traduce el diseño en código fuente. Los programadores escriben el código necesario siguiendo las especificaciones del diseño. Se aplican mejores prácticas de programación y se enfatiza la legibilidad y la mantenibilidad del código.

Pruebas: La fase de pruebas es crucial para asegurarse de que el software funcione según lo previsto. Se crean casos de prueba que evalúan las funcionalidades del software y se buscan errores y problemas. La depuración y la corrección de defectos son parte integral de esta fase.

Implementación y Despliegue: Una vez que el software ha pasado las pruebas satisfactoriamente, se implementa en el entorno de producción. Esto implica la instalación en servidores, la configuración, la migración de datos y la puesta en funcionamiento del sistema para su uso real.

Mantenimiento: El ciclo de desarrollo no termina con la implementación. La fase de mantenimiento implica la corrección de errores, la aplicación de actualizaciones y mejoras, y la gestión continua del sistema a lo largo de su vida útil.

Documentación: En cada fase del ciclo de desarrollo, se debe mantener una documentación adecuada. Esto incluye la documentación de requerimientos, el diseño, el código fuente, los manuales de usuario y cualquier información relevante para entender y mantener el sistema.

Gestión de Proyecto: A lo largo de todas las fases, se debe llevar a cabo una gestión de proyecto eficaz. Esto incluye la planificación, la programación, la asignación de recursos y el seguimiento del progreso. Metodologías como el enfoque PMI (Project Management Institute) y Agile PM son comunes en la gestión de proyectos de software.

Calidad del Software: La calidad del software es un enfoque que se aplica en todas las fases del ciclo de desarrollo. Implica asegurarse de que el software cumpla con estándares y requisitos de calidad, lo que incluye la realización de pruebas y la revisión de código.

El ciclo de desarrollo de software puede variar según la metodología y el enfoque específico utilizados, como el modelo en cascada, el desarrollo ágil, el modelo en espiral, entre otros. La elección de la metodología depende de las necesidades del proyecto y de las preferencias del equipo de desarrollo. En todos los casos, el ciclo de desarrollo de software proporciona una estructura lógica y sistemática para la creación exitosa de software.

Imagen 3: Fases del desarrollo de Software



Tomado de: https://www.viewnext.com/el-ciclo-sdlc-en-7-fases/

Importancia de la metodología de desarrollo aplicada a proyectos.

Organización y Estructura: Las metodologías proporcionan un marco de trabajo estructurado que ayuda a organizar y gestionar el proyecto de manera eficiente. Esto incluye la planificación, programación de tareas, asignación de recursos y definición de roles y responsabilidades.

Gestión de Requerimientos: Las metodologías enfatizan la recopilación y documentación adecuada de requerimientos. Esto garantiza que todas las funcionalidades necesarias se identifiquen y se tengan en cuenta en el desarrollo.

Control de Calidad: Las metodologías definen estándares de calidad y prácticas para asegurar que el software cumpla con los requerimientos y las expectativas del cliente. Las pruebas y la revisión de código son parte integral de este proceso.

Gestión de Riesgos: Las metodologías permiten la identificación y mitigación de riesgos durante el ciclo de desarrollo. Esto ayuda a anticipar problemas y a tomar medidas para evitar retrasos y costos adicionales.

Comunicación Efectiva: Las metodologías promueven la comunicación efectiva entre los miembros del equipo y los interesados en el proyecto. Esto es crucial para asegurarse de que todos estén en la misma página y para abordar problemas de manera oportuna.

Eficiencia y Productividad: Las metodologías buscan optimizar el uso de recursos y tiempo. Esto resulta en un desarrollo más eficiente y en la entrega de software en plazos razonables.

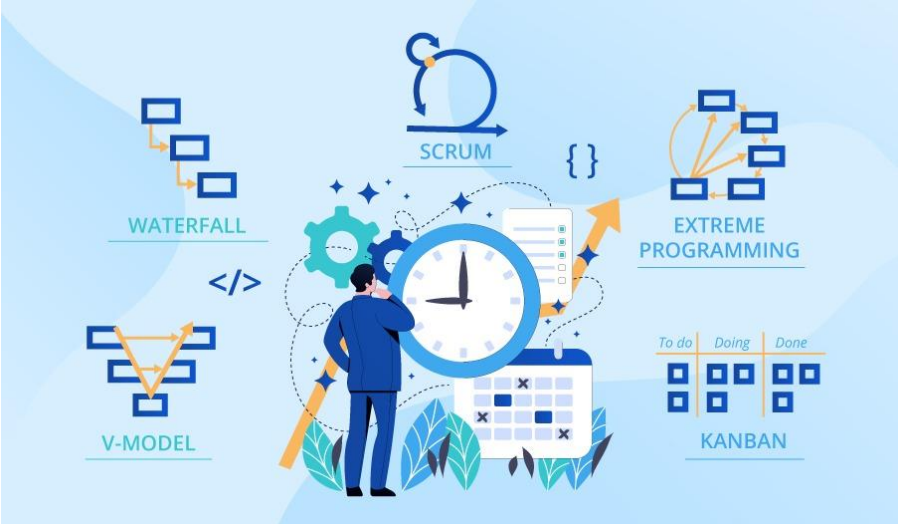
Mejora Continua: Las metodologías a menudo incorporan ciclos de retroalimentación y mejora continua. Esto significa que se aprende de la experiencia y se aplican mejoras en futuros proyectos.

Cumplimiento de Plazos y Presupuesto: Las metodologías ayudan a mantener el control sobre el progreso del proyecto, lo que facilita la entrega del software en el plazo establecido y dentro del presupuesto.

Facilitación de Colaboración: En entornos de desarrollo de software, donde se trabaja en equipos multidisciplinarios, las metodologías fomentan la colaboración y la toma de decisiones conjuntas.

Documentación y Mantenimiento: Las metodologías requieren la creación de documentación adecuada, lo que facilita la comprensión y el mantenimiento del software a lo largo del tiempo.

Imagen 4: Importancia de la metodología de desarrollo de Software



Tomado de: https://emprendecontuweb.com/cuales-son-las-metodologias-de-desarrollo-de-software-mas-eficaces/

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: ¿Es importante utilizar una metodología antes de desarrollar?

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Ensayo: Importancia de la metodología de desarrollo aplicada a proyectos.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: Procesos de desarrollo

Objetivo: Analizar los procesos de desarrollo dentro de las metodologías de ingeniería de software, comprendiendo sus etapas, enfoques y buenas prácticas, con el propósito de optimizar la planificación, ejecución y mantenimiento de proyectos tecnológicos eficientes y sostenibles.

La aplicación de una metodología de desarrollo de software es esencial para garantizar la calidad, eficiencia y éxito en proyectos de software. Ayuda a minimizar riesgos, a optimizar recursos y a mejorar la comunicación, lo que es crucial en un campo en constante evolución como el desarrollo de software. Las metodologías proporcionan una estructura y un enfoque que aseguran que los proyectos se desarrollen de manera ordenada y efectiva.

Procesos en el ciclo de desarrollo

Los procesos en el ciclo de desarrollo de software se refieren a las actividades y etapas específicas que se realizan a lo largo del ciclo para llevar a cabo la creación, diseño, construcción, prueba, implementación y mantenimiento del software. Cada proceso tiene un propósito y un conjunto de tareas asociadas que contribuyen al desarrollo exitoso del software. Aquí se proporciona una explicación detallada de los procesos comunes en el ciclo de desarrollo de software:

Recolección y análisis de requisitos

- **Objetivo:** Entender qué necesita el usuario y qué requisitos debe cumplir el software.
- **Actividades:** Reuniones con stakeholders, entrevistas, cuestionarios, análisis de documentos existentes.
- **Resultados:** Documento de requisitos del sistema (SRS) que especifica qué hará el sistema.

Planificación

- **Objetivo**: Establecer un plan de acción para el proyecto.
- **Actividades**: Definición del alcance del proyecto, estimación de tiempo y costos, asignación de recursos, creación del cronograma.

- **Resultados**: Plan de proyecto, cronograma, presupuesto.

Diseño del sistema

- **Objetivo**: Crear la arquitectura del software y su diseño detallado.
- **Actividades**: Diseño de la arquitectura del sistema, modelado de datos, diseño de la interfaz de usuario, diagramas de flujo y diagramas UML.
- **Resultados**: Documentos de diseño, maquetas y prototipos.

Desarrollo (Codificación)

- **Objetivo**: Convertir el diseño en código ejecutable.
- **Actividades**: Programación según los estándares y las especificaciones definidas, revisiones de código.
- **Resultados**: Código fuente, componentes del software.

Pruebas (Testing)

- **Objetivo**: Verificar y validar que el software funcione según los requisitos especificados y esté libre de defectos.
- **Actividades**: Pruebas unitarias, pruebas de integración, pruebas del sistema, pruebas de aceptación del usuario.
- **Resultados**: Informes de pruebas, software libre de errores (o con errores conocidos y gestionados).

Implementación (Deployment)

- **Objetivo**: Poner el software en un entorno de producción donde los usuarios finales puedan utilizarlo.
- **Actividades**: Instalación y configuración del software en servidores de producción, migración de datos, formación de usuarios.
- **Resultados**: Software en funcionamiento, manuales de usuario, formación completada.

Mantenimiento

- **Objetivo**: Garantizar que el software siga funcionando correctamente y cumplir con los nuevos requisitos.
- **Actividades**: Corrección de errores, mejoras y actualizaciones, soporte técnico.
- **Resultados**: Versiones actualizadas del software, documentación actualizada.

Retiro (Retirement)

- **Objetivo**: Desactivar el software cuando ya no es necesario o se reemplaza por una nueva versión/sistema.
- **Actividades**: Notificación a los usuarios, migración de datos si es necesario, desinstalación del software.
- **Resultados**: Software retirado de producción, documentación del proceso de retiro.

Características de los procesos.

Las características de los procesos en el ciclo de desarrollo de software son atributos distintivos que describen cómo se llevan a cabo estas etapas específicas en el desarrollo de un sistema de software. Estas características son fundamentales para comprender y gestionar eficazmente el proceso de desarrollo.

1. **Secuencia Lógica:** Los procesos siguen una secuencia lógica de actividades, lo que significa que cada proceso tiene una relación de dependencia con los procesos anteriores y posteriores. Por ejemplo, la recopilación de requerimientos generalmente precede al diseño y la implementación.
2. **Iterativo e Incremental:** Los procesos pueden ser iterativos e incrementales, lo que permite ajustar y mejorar el software a medida que se avanza en el desarrollo. Esto implica ciclos de retroalimentación y mejoras constantes.
3. **Documentación:** La documentación es una característica importante de los procesos. Cada fase y actividad del desarrollo debe documentarse adecuadamente, lo que incluye la documentación de requerimientos, diseño, código fuente, pruebas y manuales de usuario.

4. **Planificación y Gestión de Proyectos:** La planificación y la gestión de proyectos son características intrínsecas de los procesos. Esto implica la programación de actividades, la asignación de recursos, la definición de hitos y la supervisión del progreso del proyecto.

5. **Colaboración y Comunicación:** Los procesos fomentan la colaboración y la comunicación efectiva entre los miembros del equipo y los interesados en el proyecto. Esto asegura que todos estén alineados en cuanto a los objetivos y las expectativas.

6. **Calidad y Control de Calidad:** La calidad del software es un enfoque clave. Los procesos incluyen prácticas y actividades de control de calidad, como pruebas, revisión de código y aseguramiento de la calidad.

7. **Flexibilidad y Adaptabilidad:** Los procesos deben ser flexibles y adaptables para afrontar cambios en los requerimientos o en la dirección del proyecto. Esto es especialmente importante en el desarrollo ágil, donde la adaptabilidad es un principio fundamental.

8. **Gestión de Riesgos:** La identificación y la gestión de riesgos son características esenciales en el ciclo de desarrollo. Los procesos incluyen evaluación de riesgos, planificación de mitigación y seguimiento de problemas.

9. **Cumplimiento de Estándares:** Los procesos a menudo se adhieren a estándares de la industria y buenas prácticas. Esto garantiza la consistencia y la calidad en el desarrollo.

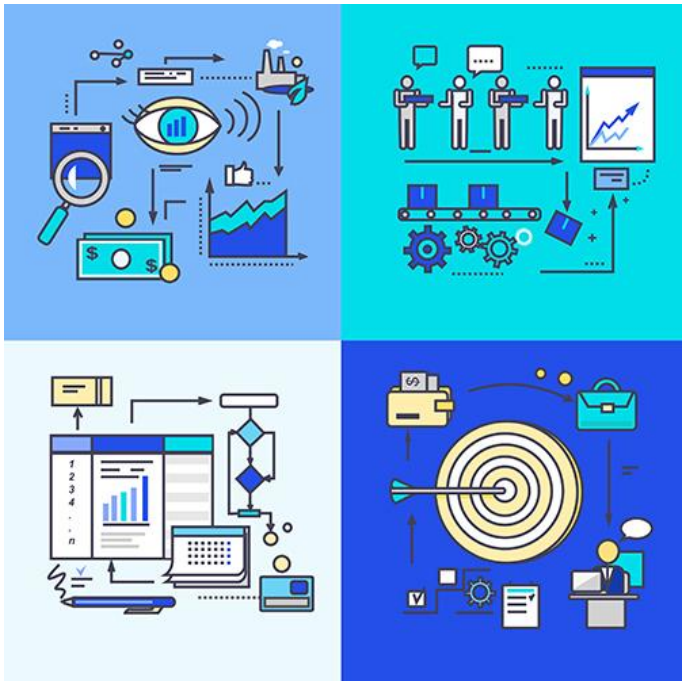
10. **Evaluación y Mejora Continua:** Los procesos suelen incluir la evaluación y la mejora continua, lo que significa que se aprende de la experiencia y se aplican mejoras en futuros proyectos.

11. **Enfoque en el Cliente:** Los procesos deben ser orientados al cliente, lo que implica una comprensión profunda de las necesidades y expectativas del cliente para el software.

12. **Eficiencia y Productividad:** Los procesos buscan la optimización de recursos y la mejora de la eficiencia en el desarrollo.

Estas características son aplicables a diversos modelos de ciclo de desarrollo, incluyendo el modelo en cascada, el modelo en espiral, las metodologías ágiles, y otros enfoques.

Imagen 1: Proceso del desarrollo de software



Tomado de: https://software-b.com/sb25/soporte/metodologia-dessarrollo.html

Proceso liviano vs Procesos Pesados.

Los términos "proceso liviano" y "proceso pesado" se refieren a enfoques diferentes para el desarrollo de software y se utilizan para describir la cantidad de documentación, planificación y formalidad involucrada en un proyecto. Aquí se detallan las diferencias clave entre ambos:

Proceso Liviano:

1. **Enfoque Ágil:** Los procesos livianos a menudo se asocian con metodologías ágiles, como Scrum y Kanban. Estas metodologías se centran en la flexibilidad, la colaboración y la entrega continua de software funcional en ciclos cortos (sprints).

2. **Documentación Mínima:** Los procesos livianos tienden a requerir menos documentación detallada en comparación con los procesos pesados. Se valora la comunicación cara a cara y la documentación justa y necesaria.

3. **Mayor Adaptabilidad:** Los procesos livianos son altamente adaptables a los cambios en los requerimientos del cliente. Pueden acomodar modificaciones a lo largo del ciclo de desarrollo.

4. **Iterativo e Incremental:** Los procesos livianos a menudo siguen un enfoque iterativo e incremental, lo que significa que se realizan entregas parciales del software y se mejoran con el tiempo.

5. **Menos Planificación Detallada:** La planificación en los procesos livianos suele ser de alto nivel, lo que permite una planificación más flexible en el proyecto.

6. **Colaboración Continua:** Se enfatiza la colaboración continua con los interesados y el equipo de desarrollo. La comunicación constante es esencial.

Proceso Pesado:

1. **Enfoque Tradicional:** Los procesos pesados, a menudo asociados con el modelo en cascada, el modelo en espiral y otros enfoques más tradicionales, tienden a ser más estructurados y planificados.

2. **Documentación Completa:** Los procesos pesados requieren una documentación exhaustiva que incluye especificaciones detalladas, planes de proyecto extensos y documentación técnica.

3. **Menor Adaptabilidad:** Los procesos pesados son menos flexibles en respuesta a cambios en los requerimientos. Los cambios suelen ser costosos y difíciles de implementar.

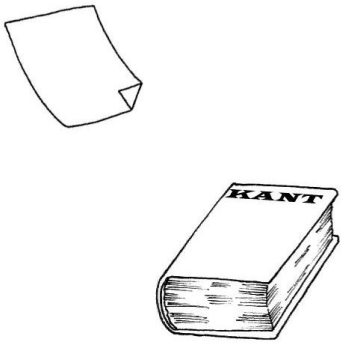
4. **Secuencia Fija:** En los procesos pesados, las fases siguen una secuencia fija, lo que significa que una fase debe completarse antes de pasar a la siguiente.

5. **Planificación Detallada:** La planificación en los procesos pesados es rigurosa y detallada. Se define una extensa planificación antes de comenzar el desarrollo.

6. **Enfoque en Documentación:** La documentación tiene un papel fundamental en los procesos pesados para garantizar que cada paso del proceso se documente adecuadamente.

La elección entre un proceso liviano y uno pesado depende de las necesidades y la naturaleza del proyecto. Los procesos livianos son ideales para proyectos donde la flexibilidad y la adaptabilidad son críticas, especialmente en entornos donde los requerimientos del cliente pueden cambiar con frecuencia. Por otro lado, los procesos pesados son más adecuados para proyectos donde se requiere una documentación detallada y una planificación exhaustiva, como en proyectos de ingeniería de software críticos o proyectos gubernamentales.

Imagen 2: Proceso liviano vs pesado



Tomado de: https://www.pinterest.es/pin/492018328016862026/

En muchos casos, las organizaciones optan por un enfoque híbrido, que combina elementos de procesos livianos y pesados para aprovechar las ventajas de ambos enfoques.

Roles en el desarrollo de software.

En el desarrollo de software, se desempeñan varios roles, y cada uno tiene responsabilidades específicas en el proceso. La distribución de roles puede variar según la metodología y el tamaño del proyecto, pero a continuación se presentan algunos roles comunes en el desarrollo de software:

1. **Desarrollador de Software (Programador):** Los desarrolladores de software son responsables de escribir el código del programa según los diseños y requerimientos establecidos. Pueden trabajar en diferentes lenguajes de programación y en el desarrollo de componentes de software.

2. **Analista de Requerimientos:** Los analistas de requerimientos colaboran con los clientes y los interesados para comprender y documentar los requerimientos del software. Luego, comunican estos requerimientos al equipo de desarrollo.

3. **Diseñador de Software:** Los diseñadores de software se encargan de crear la arquitectura y la estructura del sistema. Esto incluye el diseño de la interfaz de usuario, la arquitectura del sistema y la planificación de la base de datos.

4. **Tester o Analista de Pruebas:** Los testers o analistas de pruebas son responsables de la creación y ejecución de casos de prueba para evaluar el software en busca de errores y problemas. También informan sobre la calidad del software.

5. **Gerente de Proyecto:** Los gerentes de proyecto supervisan y gestionan todas las actividades relacionadas con el desarrollo de software. Esto incluye la planificación, la programación, la asignación de recursos y la comunicación con los interesados.

6. **Arquitecto de Software:** Los arquitectos de software diseñan la estructura general del sistema, definiendo cómo se comunican los componentes y asegurando que se cumplan los principios de diseño.

7. **Especialista en Base de Datos:** Los especialistas en bases de datos se centran en el diseño, la administración y la optimización de bases de datos. Aseguran que el software acceda y almacene datos de manera eficiente y segura.

8. **Especialista en Seguridad:** Los especialistas en seguridad se dedican a identificar vulnerabilidades y amenazas de seguridad en el software y a aplicar prácticas para protegerlo contra ataques.

9. **Diseñador de Interfaz de Usuario (UI/UX):** Los diseñadores de interfaz de usuario se concentran en crear una experiencia de usuario atractiva y funcional. Definen la apariencia y la usabilidad de la interfaz de usuario del software.

10. **Especialista en Redes o Infraestructura:** En proyectos que requieren una infraestructura de red, estos especialistas se encargan de diseñar, configurar y mantener la red y la infraestructura de servidores.

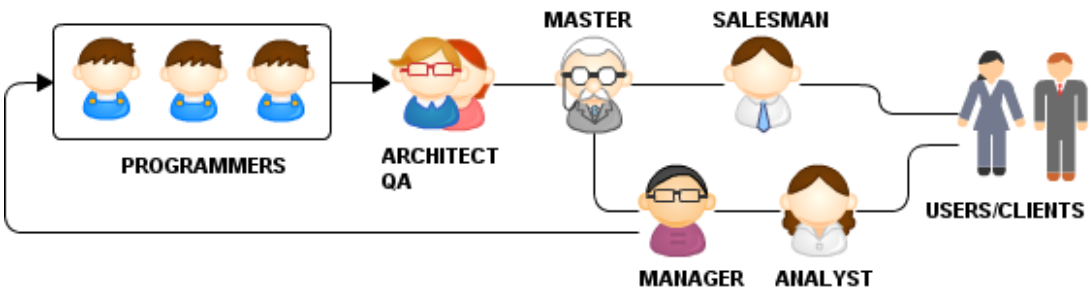
11. **Especialista en Documentación Técnica:** Los especialistas en documentación técnica se ocupan de crear y mantener la documentación técnica, que incluye manuales de usuario, documentación técnica interna y comentarios en el código.

12. **Líder de Equipo o Scrum Master:** En equipos ágiles, el líder de equipo o Scrum Master es responsable de facilitar la comunicación y la colaboración entre los miembros del equipo, garantizando que se cumplan los principios ágiles.

13. **Product Owner:** En metodologías ágiles como Scrum, el Product Owner es el responsable de definir y priorizar las funcionalidades del software desde la perspectiva del cliente.

Es importante señalar que en proyectos pequeños o equipos pequeños, una persona puede desempeñar varios de estos roles simultáneamente. En proyectos más grandes, es común contar con un equipo interdisciplinario que incluye estos y otros roles especializados.

Imagen 3: Roles en el proceso de desarrollo



Tomado de: http://unidad.wikidot.com/tema3

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: ¿ Es importante utilizar una metodología antes de desarrollar?

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Ensayo: Procesos de Desarrollo de Software.

- ✓ Escribe un ensayo de 1 a 2 páginas donde expliques qué son los procesos de desarrollo en la ingeniería de software.
- ✓ Analiza las metodologías más utilizadas (Cascada, Ágil, Espiral, DevOps), sus ventajas y en qué casos se aplican mejor.
- ✓ Entrega en formato PDF o Word, con fuente Arial o Times New Roman, tamaño 12, interlineado 1.5.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

II. DATOS GENERALES DE LA UNIDAD 2

Información General

1.1.	Nivel	Segundo					
1.2.	Nombre Unidad	Metodologías de Desarrollo					
1.3.	Tema 1 – Unidad	Metodologías					
1.4.	Tema 2 – Unidad	Estructura del Proceso					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	2					
1.7.	Total, Horas Unidad	15					
1.8.	Detalle de horas Unidad	ACD	6	AA	4	APE	5

Resultado de Aprendizaje de Unidad

Al finalizar la unidad "Metodologías de Desarrollo" de la materia de Metodología del Desarrollo del Software, los estudiantes serán capaces de comprender y aplicar diversas metodologías y enfoques para la creación de software. Esto incluye un conocimiento profundo de metodologías tradicionales como el modelo en cascada y el desarrollo incremental, así como enfoques ágiles como Scrum y Kanban. Los estudiantes podrán evaluar y seleccionar la metodología más adecuada según el contexto del proyecto, gestionar el ciclo de vida del desarrollo del software, y adaptar prácticas de gestión y desarrollo para mejorar la eficiencia y calidad del producto final.

Metodología

La clase virtual grabada comienza con una introducción clara sobre lo que los estudiantes aprenderán en esa sesión. Se establecen los objetivos de aprendizaje para que los estudiantes sepan qué esperar y en qué enfocarse.

El contenido de la asignatura se desarrolla dividiéndolo en secciones o módulos. Se utiliza una estructura organizada y lógica para abordar cada tema, combinando diferentes enfoques pedagógicos según el contenido, como explicaciones, ejemplos, demostraciones, gráficos y diagramas. Aunque la clase es grabada, es importante incorporar elementos interactivos para mantener el compromiso de los estudiantes. Esto se puede lograr a través de preguntas para reflexionar, ejercicios, cuestionarios o pausas para discusión, fomentando su participación activa.

DESARROLLO

Unidad 2: Metodologías de Desarrollo Tradicionales

Tema 1: Metodologías

Objetivo: Aplicar metodologías de diseño de software en la solución de problemas comunes y no comunes.

- Metodología pesada RUP
- Historia de RUP
- Características principales de RUP

Las metodologías de desarrollo de software son enfoques estructurados que guían a los equipos de desarrollo a lo largo del ciclo de vida de un proyecto, desde la concepción inicial hasta la entrega y mantenimiento del producto final. Estas metodologías proporcionan un marco de trabajo que facilita la planificación, organización y ejecución de las tareas necesarias para crear software de alta calidad, minimizando riesgos y optimizando recursos.

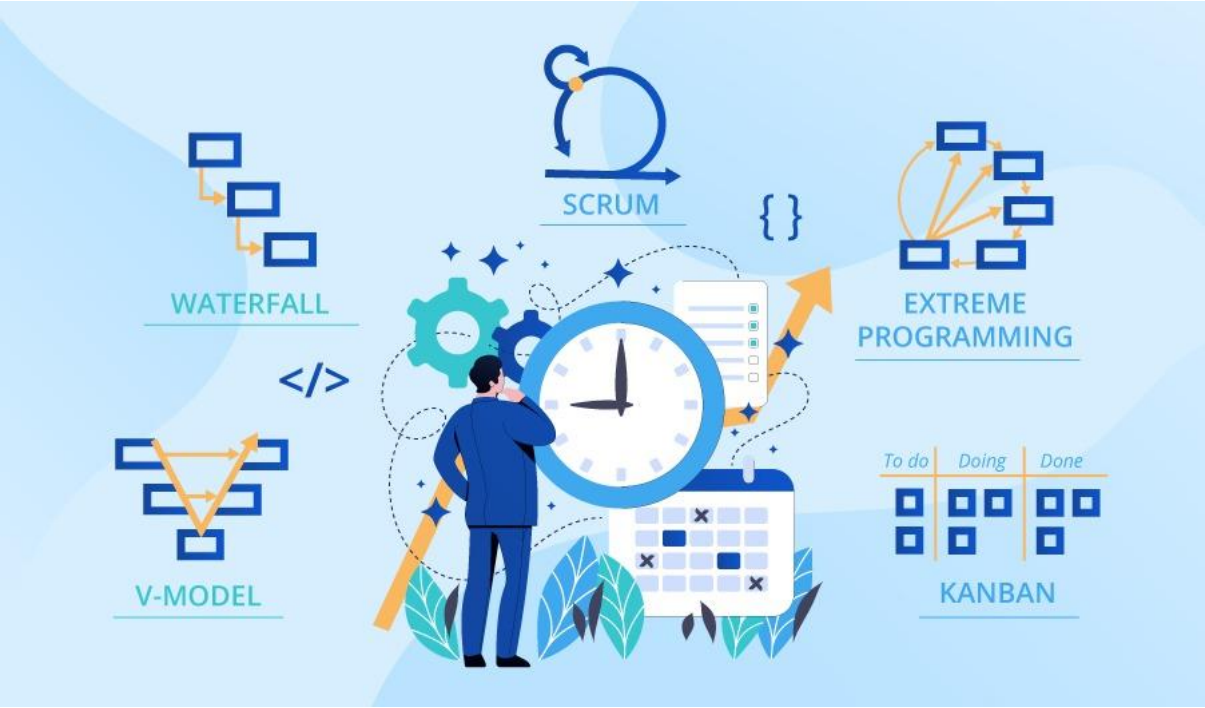
Una de las metodologías más tradicionales es el modelo en cascada, que sigue un enfoque secuencial y lineal. Cada fase del desarrollo, como el análisis de requisitos, diseño, implementación, pruebas y mantenimiento, debe completarse antes de pasar a la siguiente. Aunque este modelo es fácil de entender y gestionar, puede ser rígido y no adaptarse bien a proyectos donde los requisitos pueden cambiar con el tiempo.

En contraste, las metodologías ágiles, como Scrum y Kanban, ofrecen un enfoque más flexible y adaptable. Scrum divide el trabajo en sprints cortos y enfocados, permitiendo revisiones regulares y ajustes rápidos según los comentarios de los clientes y las necesidades del proyecto. Kanban, por su parte, se centra en la visualización del flujo de trabajo y la mejora continua, permitiendo a los equipos gestionar y optimizar sus procesos de manera incremental.

Otro enfoque ágil es el desarrollo extremo (Extreme Programming, XP), que enfatiza la importancia de la colaboración estrecha entre desarrolladores y clientes, la retroalimentación continua y la entrega frecuente de pequeñas mejoras del software. Las prácticas de XP, como la programación en parejas y las pruebas continuas, buscan mejorar la calidad del código y la capacidad de respuesta a los cambios.

La elección de la metodología adecuada depende de varios factores, como la naturaleza del proyecto, el tamaño del equipo, la estabilidad de los requisitos y las expectativas del cliente. En algunos casos, puede ser beneficioso combinar elementos de diferentes metodologías para crear un enfoque híbrido que se adapte mejor a las necesidades específicas del proyecto. En última instancia, el objetivo de cualquier metodología de desarrollo de software es asegurar que el proceso sea eficiente, predecible y capaz de entregar un producto que satisfaga las expectativas de los usuarios finales.

Imagen 1: Metodologías



Tomado de: https://emprendecontuweb.com/cuales-son-las-metodologias-de-desarrollo-de-software-mas-eficaces/

Metodología pesada RUP

La Metodología Pesada RUP (Rational Unified Process) es un enfoque de desarrollo de software que se clasifica como una metodología de desarrollo ágil. Fue desarrollada por Rational Software Corporation, que fue adquirida por IBM. RUP proporciona un marco detallado y estructurado para la planificación, ejecución y control de proyectos de desarrollo de software.

Rational Unified Process (RUP): "RUP es un proceso de desarrollo de software que se centra en la mejora continua, la calidad del software y la gestión eficiente de

proyectos. Proporciona un marco estructurado que abarca desde la concepción del proyecto hasta la entrega del producto, integrando buenas prácticas de desarrollo de software y adaptándose a las necesidades específicas del proyecto." - (Autores de RUP, Rational Software Corporation)

Ejemplos Asociados:

Desarrollo de Sistemas Empresariales: RUP ha sido aplicado con éxito en el desarrollo de sistemas empresariales complejos, donde la claridad en los requisitos y la gestión efectiva de cambios son críticas.

Proyectos de Desarrollo de Software a Gran Escala: Debido a su estructura y enfoque, RUP es adecuado para proyectos de desarrollo de software a gran escala donde se requiere una planificación y gestión detallada.

Proyectos que Requieren Modelado Riguroso: RUP es efectivo en proyectos que necesitan un modelado detallado para comunicar y documentar los aspectos técnicos del sistema.

Imagen 2: RUP



Tomado de: https://fr.educaplay.com/jeuimprimable/2055227-metodologia_rup.html

Historia de RUP

La historia del Rational Unified Process (RUP) se remonta a la década de 1990 y está vinculada a la empresa Rational Software Corporation. la historia de RUP está intrínsecamente ligada a la evolución de Rational Software Corporation y su enfoque en mejorar la ingeniería de software a través de metodologías y herramientas avanzadas. Aunque RUP ha experimentado cambios en su adopción a lo largo del tiempo, su contribución a la disciplina del desarrollo de software sigue siendo parte integral de la historia de la ingeniería de software.

Creación de Rational Software Corporation: La historia de RUP comienza con la fundación de Rational Software Corporation en 1981 por Paul Levy y Mike Devlin. La empresa se estableció con el objetivo de proporcionar herramientas y soluciones para mejorar el proceso de desarrollo de software.

Desarrollo de Herramientas Rational: Rational Software Corporation se destacó por desarrollar herramientas innovadoras para ingeniería de software, como Rational Rose, una herramienta de modelado visual que permitía a los desarrolladores crear modelos UML (Unified Modeling Language) para representar gráficamente el diseño de sistemas.

Énfasis en la Ingeniería de Software: La empresa Rational tenía un enfoque centrado en la ingeniería de software y se esforzaba por mejorar la calidad y eficiencia del desarrollo de software mediante el uso de metodologías estructuradas y herramientas visuales.

Desarrollo del Rational Unified Process (RUP): A mediados de la década de 1990, Rational Software Corporation, liderada por Philippe Kruchten, comenzó a desarrollar una metodología de proceso unificado que más tarde se conocería como el Rational Unified Process (RUP). Este proceso integraría las mejores prácticas de desarrollo de software en un marco coherente.

Lanzamiento de RUP: RUP fue presentado oficialmente en 1998. La metodología se basó en principios fundamentales, como el modularidad, la iteración y la colaboración entre equipos, y se diseñó para ser adaptable a diferentes tipos y tamaños de proyectos.

Adopción y Popularidad: RUP ganó rápidamente aceptación en la industria del desarrollo de software, especialmente en proyectos empresariales complejos. La metodología ofrecía un enfoque estructurado y modular para el desarrollo de software, y se benefició de la popularidad de las herramientas Rational, como Rational Rose.

Adquisición por IBM: En 2003, IBM adquirió Rational Software Corporation. Esta adquisición consolidó las herramientas Rational y RUP bajo la cartera de IBM, brindando a la metodología un respaldo adicional y una presencia más amplia en la comunidad de desarrollo de software.

Evolución y Adaptación: A lo largo de los años, RUP ha evolucionado y se ha adaptado a las cambiantes tendencias en el desarrollo de software. Se han introducido actualizaciones y versiones para mantener la relevancia en un entorno donde las metodologías ágiles, como Scrum, ganaron popularidad.

Transición a Enfoques Ágiles: A medida que las metodologías ágiles se volvieron más predominantes en la industria, RUP experimentó una disminución en su adopción. Muchas organizaciones buscaron enfoques más ligeros y flexibles para el desarrollo de software.

Legado y Contribución: A pesar de su disminución en popularidad, el Rational Unified Process dejó un legado significativo al introducir principios importantes en la ingeniería de software y proporcionar un enfoque estructurado que influyó en otras metodologías y enfoques ágiles.

Características principales de RUP

Enfoque Iterativo e Incremental: RUP se basa en el principio de desarrollar el software de manera iterativa e incremental. Los proyectos se dividen en iteraciones, cada una de las cuales produce una versión funcional del software. Esto permite la entrega temprana de funcionalidades y facilita la adaptación a cambios en los requisitos.

Ciclo de Vida del Proyecto Dividido en Fases: RUP organiza el ciclo de vida del proyecto en cuatro fases principales: Inicio, Elaboración, Construcción y Transición. Cada fase tiene objetivos específicos y actividades asociadas. Estas fases proporcionan una estructura para la planificación y el control del proyecto.

Arquitectura Centrada en Casos de Uso: RUP pone un fuerte énfasis en la identificación y diseño de casos de uso. La arquitectura del sistema se desarrolla alrededor de estos casos de uso, lo que ayuda a garantizar que el sistema cumpla con las necesidades y expectativas del usuario final.

Gestión de Cambios Controlada: RUP aborda la gestión de cambios de manera estructurada. Los cambios en los requisitos se evalúan y se incorporan de manera controlada durante las iteraciones. Esto ayuda a mantener la integridad del sistema y a gestionar eficientemente las modificaciones.

Modelado Visual: Utiliza modelos visuales, como diagramas de clases y diagramas de casos de uso, para representar gráficamente los requisitos, la arquitectura y el diseño del sistema. El modelado visual facilita la comunicación y la comprensión entre los miembros del equipo.

Roles y Responsabilidades Definidos: RUP define roles y responsabilidades claramente dentro del equipo de desarrollo. Los roles incluyen analistas, diseñadores, programadores y testers, entre otros. Cada rol tiene tareas y actividades específicas.

Gestión de Riesgos Integrada: Incorpora la gestión de riesgos de manera integral. Se identifican y gestionan proactivamente los riesgos a lo largo del proyecto. Esto contribuye a la toma de decisiones informada y a la mitigación de posibles problemas.

Calidad Incorporada en el Proceso: RUP se centra en la calidad del software desde el principio. La realización de pruebas se integra en cada iteración, lo que contribuye a la detección temprana de problemas y a la entrega de un producto de mayor calidad.

Adaptabilidad a Diferentes Tipos de Proyectos: RUP es adaptable a una variedad de proyectos, desde pequeños hasta grandes y complejos. Sus principios se pueden ajustar para adaptarse a las necesidades específicas de diferentes tipos de desarrollo de software.

Soporte de Herramientas Rational: RUP es complementado por herramientas de desarrollo de software Rational, como Rational Rose y Rational ClearCase. Estas herramientas proporcionan soporte para el modelado visual, la gestión de cambios y otras actividades clave en el proceso.

Documentación Clara y Detallada: RUP hace énfasis en la documentación clara y detallada a lo largo del proyecto. Los artefactos, como documentos de requisitos y manuales, son parte integral de cada fase del ciclo de vida del proyecto.

Alineación con Mejores Prácticas de Ingeniería de Software: RUP se basa en mejores prácticas de ingeniería de software y se ha desarrollado en línea con principios de ingeniería de software eficiente y efectiva.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: ¿Qué es XP?

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Investigación: Metodologías de Desarrollo Tradicionales.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: Estructura del Proceso

Objetivo: Aplicar metodologías de diseño de software en la solución de problemas comunes y no comunes.

- Proceso centrado en la arquitectura
- Proceso iterativo e incremental
- Otras prácticas RUP
- Estructura del proceso

Proceso Centrado en la Arquitectura

El proceso centrado en la arquitectura es una metodología que pone un fuerte énfasis en la arquitectura del software como base para el desarrollo del sistema. Esta aproximación asegura que las decisiones arquitectónicas críticas se tomen tempranamente en el ciclo de vida del desarrollo. La arquitectura del software define la estructura global del sistema y las interacciones entre sus componentes, proporcionando una base sólida y estable sobre la cual se puede construir el resto del sistema.

Este enfoque ayuda a identificar y mitigar riesgos técnicos desde el principio. Al establecer una arquitectura sólida, los equipos pueden prever problemas potenciales y tomar decisiones informadas que afectan a la escalabilidad, seguridad y mantenimiento del sistema. Además, la documentación arquitectónica actúa como una guía para los desarrolladores, asegurando que todos sigan una visión común y bien definida del sistema.

Un ejemplo de proceso centrado en la arquitectura es la metodología RUP (Rational Unified Process), que promueve la creación de una arquitectura robusta como una de sus prácticas fundamentales. En RUP, se lleva a cabo una fase inicial de concepción y elaboración en la que se define y valida la arquitectura antes de proceder con el desarrollo detallado.

Proceso Iterativo e Incremental

El proceso iterativo e incremental es una metodología de desarrollo que divide el proyecto en pequeñas iteraciones, cada una de las cuales produce un incremento del producto final. En cada iteración, se realizan actividades de análisis, diseño, implementación y pruebas, lo que permite obtener retroalimentación constante y realizar ajustes continuos en el proyecto.

Este enfoque es especialmente útil en entornos donde los requisitos pueden cambiar o no están completamente definidos desde el inicio. Al desarrollar el sistema de manera incremental, los equipos pueden adaptarse rápidamente a los cambios y mejoras solicitados por los stakeholders. Además, este proceso reduce los riesgos asociados con el desarrollo de software al permitir la identificación temprana de problemas y su resolución en iteraciones posteriores.

El modelo iterativo e incremental se complementa con técnicas ágiles como Scrum, que estructuran las iteraciones en sprints cortos y enfocados, permitiendo la entrega continua de valor al cliente. Esto facilita la colaboración entre el equipo de desarrollo y los usuarios finales, asegurando que el producto evolucione de acuerdo con las necesidades reales del negocio.

Otras Prácticas RUP

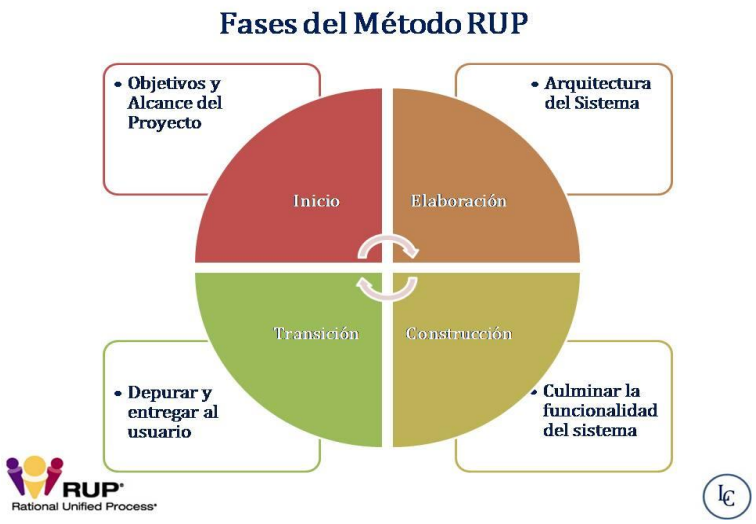
RUP (Rational Unified Process) es un marco de desarrollo de software extensible que se adapta a las necesidades específicas del proyecto. Además de centrarse en la arquitectura y el desarrollo iterativo e incremental, RUP incorpora varias prácticas clave diseñadas para mejorar la calidad y eficiencia del proceso de desarrollo.

Una de estas prácticas es la gestión de requisitos, que implica la identificación, documentación y seguimiento de los requisitos del sistema a lo largo del ciclo de vida del proyecto. Esta práctica asegura que el producto final cumpla con las expectativas y necesidades del cliente.

Otra práctica importante es la gestión de cambios, que proporciona un marco estructurado para manejar las solicitudes de cambio de manera controlada. Esto incluye la evaluación del impacto de los cambios, la aprobación de los mismos y la actualización de la documentación y el código según sea necesario.

Además, RUP enfatiza la importancia de las pruebas continuas y la integración temprana. Las pruebas continuas permiten la detección y corrección de errores en etapas tempranas, lo que reduce el costo de corrección de defectos. La integración temprana asegura que los diferentes componentes del sistema funcionen bien juntos desde el principio, evitando problemas de integración en fases avanzadas del proyecto.

Imagen 1: RUP



Tomado de: https://lean-management.site/rup/

Estructura del Proceso

La estructura del proceso en metodologías tradicionales de desarrollo de software suele ser bien definida y secuencial. En modelos como el Waterfall (cascada), el proceso se divide en fases distintas: requisitos, diseño, implementación, verificación y mantenimiento. Cada fase debe completarse antes de pasar a la siguiente, y los resultados de cada fase actúan como entrada para la siguiente.

Esta estructura facilita la planificación y gestión del proyecto, ya que cada fase tiene objetivos claros y entregables específicos. Sin embargo, también puede ser rígida y no tan adaptable a cambios en los requisitos del cliente una vez que se ha avanzado en el proceso.

En contraste, enfoques como RUP, aunque considerados tradicionales, incorporan cierta flexibilidad al combinar la estructura secuencial con ciclos iterativos. Esto permite una mayor adaptabilidad y mejora continua del producto, manteniendo al mismo tiempo una organización y documentación claras a lo largo del proyecto.

La elección de la estructura del proceso depende de varios factores, incluyendo la naturaleza del proyecto, la estabilidad de los requisitos y la cultura organizacional. Los equipos deben evaluar cuidadosamente estos factores para seleccionar la metodología que mejor se adapte a sus necesidades y objetivos.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: ¿Qué es RUP?

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Investigación: Estructura del Proceso centrado en la arquitectura.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

III. DATOS GENERALES DE LA UNIDAD 3

Información General

1.1.	Nivel	Segundo					
1.2.	Nombre Unidad	Metodologías de Desarrollo Ágiles					
1.3.	Tema 1 – Unidad	Metodología XP					
1.4.	Tema 2 – Unidad	Metodología SCRUM					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	3					
1.7.	Total, Horas Unidad	15					
1.8.	Detalle de horas Unidad	ACD	6	AA	4	APE	5

Resultado de Aprendizaje de Unidad

Al finalizar la unidad, el estudiante comprenderá los principios fundamentales de las metodologías ágiles de desarrollo de software, con un enfoque en Extreme Programming (XP) y SCRUM. Identificará las características, ventajas y aplicaciones de cada metodología, diferenciando sus enfoques y estructuras. Además, desarrollará habilidades para aplicar prácticas ágiles en la gestión y ejecución de proyectos de software, fomentando la colaboración, la adaptabilidad y la mejora continua en equipos de trabajo.

Metodología

El aprendizaje se llevará a cabo mediante la exploración de material interactivo, videos explicativos y estudios de caso sobre la implementación de metodologías ágiles en proyectos reales. Se fomentará la participación en foros de discusión para comparar XP y SCRUM, analizando sus beneficios y desafíos en diferentes contextos. Asimismo, los estudiantes realizarán actividades prácticas, como la simulación de sprints en SCRUM y la aplicación de prácticas de desarrollo XP, utilizando herramientas digitales de gestión ágil como Trello, Jira o Azure DevOps. La retroalimentación docente y la evaluación colaborativa permitirán mejorar la comprensión y aplicación de estos marcos de trabajo en entornos de desarrollo de software.

DESARROLLO

Unidad N.3: Metodologías de Desarrollo Ágiles

Tema 1: Metodología XP

Objetivo: Analizar la metodología Extreme Programming (XP) como un enfoque ágil para el desarrollo de software, comprendiendo sus principios, prácticas y beneficios, con el fin de optimizar la eficiencia del trabajo en equipo, la calidad del código y la adaptabilidad a los cambios en los proyectos de software.

- Concepto, Características
- Proceso, Requisito, Roles
- Gestión y seguimiento

Introducción

La Unidad 3 de la asignatura "Metodología de Diseño de Software" explicaremos sobre las metodologías ágiles de desarrollo, centrándonos específicamente en Extreme Programming (XP). En esta clase, exploraremos los conceptos fundamentales, las características distintivas, el proceso de desarrollo, los requisitos, los roles, así como la gestión y el seguimiento en la metodología XP.

Metodología XP: Concepto y Características: La metodología Extreme Programming (XP) es una metodología ágil que se enfoca en mejorar la calidad del software y la satisfacción del cliente mediante la adaptabilidad y la flexibilidad en el desarrollo. Algunas de las características clave de XP incluyen:

1. **Retroalimentación Continua:**
    - XP prioriza la retroalimentación constante entre los desarrolladores y los clientes a lo largo de todo el ciclo de vida del desarrollo.
2. **Desarrollo Iterativo e Incremental:**
    - El desarrollo en XP se realiza en iteraciones cortas e incrementales, permitiendo la entrega rápida y frecuente de versiones del software.

3. **Pruebas Continuas:**

    - La metodología XP aboga por la escritura de pruebas antes del código y la ejecución de pruebas de manera continua para garantizar la calidad y la detección temprana de errores.

4. **Programación en Parejas (Pair Programming):**

    - Los desarrolladores trabajan en parejas, compartiendo una estación de trabajo. Esto promueve la colaboración, la revisión de código en tiempo real y la transferencia de conocimientos.

Extreme Programming (XP) es una metodología de desarrollo de software ágil que se centra en mejorar la calidad del software y la satisfacción del cliente mediante prácticas de desarrollo flexibles y adaptativas. XP se originó a fines de la década de 1990 y se ha ganado popularidad debido a su enfoque pragmático y su capacidad para adaptarse rápidamente a los cambios en los requisitos del cliente. La metodología promueve la colaboración estrecha entre los miembros del equipo, la retroalimentación continua y la entrega frecuente de software funcional.

Características Principales de Extreme Programming (XP):

1. **Desarrollo Iterativo e Incremental:**

    - XP aboga por el desarrollo en ciclos cortos e incrementales. El software se desarrolla y entrega en iteraciones, lo que permite obtener rápidamente funcionalidades útiles y adaptarse a cambios en los requisitos.

2. **Retroalimentación Continua:**

    - La retroalimentación constante es una característica clave de XP. Los clientes, programadores y otros miembros del equipo colaboran estrechamente para recibir y proporcionar retroalimentación a lo largo de todo el proceso de desarrollo.

3. **Programación en Parejas (Pair Programming):**

    - Los desarrolladores trabajan en parejas en una misma estación de trabajo. Esta práctica no solo mejora la calidad del código, sino que

también fomenta la colaboración, el aprendizaje compartido y la revisión continua del código.

4. **Pruebas Continuas:**
    - XP promueve la escritura de pruebas antes de escribir el código de producción. Las pruebas son ejecutadas de manera continua durante el desarrollo para asegurar la calidad y facilitar la detección temprana de errores.
5. **Cliente en el Sitio (On-Site Customer):**
    - Un representante del cliente está disponible en el sitio para colaborar estrechamente con el equipo de desarrollo. Esto facilita la comunicación directa, la toma de decisiones rápidas y la alineación constante con las necesidades del cliente.
6. **Refactorización Constante:**
    - XP fomenta la mejora continua del código a través de la refactorización constante. Esto implica ajustar y mejorar el diseño del código sin cambiar su comportamiento externo, lo que contribuye a mantener un código más limpio y fácil de mantener.
7. **Planificación del Juego (Planning Game):**
    - La planificación del juego en XP es un proceso colaborativo en el que el cliente y el equipo de desarrollo participan en la definición de historias de usuario y la asignación de tiempos de desarrollo.
8. **Reuniones Diarias (Stand-up Meetings):**
    - Se llevan a cabo reuniones diarias cortas para mantener a todos los miembros del equipo informados sobre el progreso, los obstáculos y las próximas tareas. Estas reuniones fomentan la transparencia y la resolución rápida de problemas.

Proceso en XP:

El proceso en Extreme Programming (XP) se caracteriza por su enfoque ágil y flexible, centrado en la entrega incremental de software funcional y la adaptación constante a

los cambios en los requisitos del cliente. A continuación, se describen las etapas y prácticas clave del proceso en XP:

1. **Exploración (Exploration):**
    - En la etapa de exploración, se trabaja estrechamente con el cliente para identificar y comprender los requisitos del sistema. Se definen las historias de usuario, que son breves descripciones de funcionalidades desde la perspectiva del usuario final.
2. **Planificación del Juego (Planning Game):**
    - La planificación del juego implica la colaboración entre el cliente y el equipo de desarrollo para priorizar las historias de usuario y asignar tiempos de desarrollo. Durante esta fase, se establecen los objetivos para la iteración.
3. **Iteraciones (Iterations):**
    - El desarrollo en XP se realiza en ciclos cortos llamados iteraciones. Cada iteración tiene una duración fija y se centra en la entrega de un conjunto específico de características. Al final de cada iteración, se obtiene una versión funcional del software.
4. **Diseño Simple (Simple Design):**
    - Se fomenta la práctica de mantener el diseño del software lo más simple posible. Esto implica tomar decisiones de diseño que satisfagan los requisitos actuales sin agregar complejidad innecesaria.
5. **Programación en Parejas (Pair Programming):**
    - Los desarrolladores trabajan en parejas, compartiendo una estación de trabajo. Uno es el "conductor" (quien escribe el código) y el otro es el "observador" (quien revisa y proporciona comentarios). Esta práctica mejora la calidad del código y facilita el aprendizaje compartido.
6. **Pruebas Continuas (Continuous Testing):**
    - Las pruebas son una parte integral del proceso en XP. Se escriben pruebas antes de desarrollar el código de producción, y estas pruebas

se ejecutan de manera continua durante el desarrollo para garantizar la calidad y detectar errores tempranos.

7. **Reuniones Diarias (Stand-up Meetings):**
    - Se llevan a cabo reuniones diarias cortas donde el equipo se pone al día con el progreso, comparte información sobre posibles obstáculos y coordina las próximas tareas. Estas reuniones promueven la transparencia y la colaboración.
8. **Cliente en el Sitio (On-Site Customer):**
    - Un representante del cliente está disponible en el sitio para proporcionar clarificaciones rápidas, tomar decisiones y garantizar que el desarrollo se alinee con las necesidades del cliente.
9. **Refactorización Constante (Continuous Refactoring):**
    - Se practica la refactorización constante para mejorar el diseño del código sin cambiar su comportamiento externo. Esto contribuye a mantener un código limpio y fácil de mantener.
10. **Entregas Frecuentes:**
    - Después de cada iteración, se realiza una entrega de software funcional al cliente. Esto permite obtener comentarios rápidos y realizar ajustes según sea necesario

Requisitos y Roles:

1. **Requisitos:**
    - Los requisitos en XP se expresan en términos de historias de usuario, que son breves descripciones de funcionalidades desde la perspectiva del usuario final. Estas historias son evaluadas y priorizadas en colaboración con el cliente.
2. **Roles en XP:**
        - **Cliente (Customer):** Colabora estrechamente con el equipo de desarrollo, establece prioridades y proporciona retroalimentación continua.

- **Programadores (Programmers):** Responsables de la implementación del código y la realización de pruebas.
- **Entrenador (Coach):** Facilita la adopción y mejora continua de las prácticas de XP.
- **Tester:** Trabaja en estrecha colaboración con los programadores para realizar pruebas continuas.
- **Tracker:** Gestiona la planificación y el progreso a través de la herramienta de seguimiento (tracker).

Gestión y Seguimiento en XP:

1. **Planificación del Juego (Planning Game):**
    - XP utiliza la planificación del juego para establecer las prioridades de desarrollo. El cliente y el equipo de desarrollo participan en la definición de las historias de usuario y la asignación de tiempos de desarrollo.
2. **Seguimiento mediante el Tracker:**
    - El Tracker es una herramienta utilizada para realizar el seguimiento del progreso del proyecto, incluyendo la implementación de historias de usuario y la gestión del tiempo.
3. **Reuniones Diarias (Stand-up Meetings):**
    - Se llevan a cabo reuniones diarias cortas para mantener a todos los miembros del equipo informados sobre los avances y los posibles obstáculos.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: La Metodología XP (Extreme Programming) es un enfoque ágil que busca mejorar la calidad del software mediante prácticas colaborativas y flexibles.

- ✓ ¿Cuáles crees que son las ventajas de usar XP en el desarrollo de software?

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Evaluación de Herramientas para la Gestión y Seguimiento en XP.

- ✓ Investiga herramientas utilizadas en XP para la gestión y seguimiento de proyectos, como JIRA, Trello, Pivotal Tracker o GitHub Projects.
- ✓ Explica cómo estas herramientas ayudan a gestionar el desarrollo ágil en equipos que aplican XP.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: Metodología SCRUM

Objetivo: Examinar la metodología SCRUM como un enfoque ágil para la gestión de proyectos, comprendiendo sus principios, roles, eventos y artefactos, con el propósito de optimizar la planificación, ejecución y entrega de productos de software de manera eficiente y adaptable a los cambios.

Concepto y Características

SCRUM es un marco de trabajo ágil utilizado para gestionar proyectos de desarrollo de software y otros proyectos complejos. Surgió como una respuesta a las limitaciones de los enfoques tradicionales de gestión de proyectos, centrándose en la flexibilidad, adaptabilidad y entrega continua de valor al cliente. SCRUM se basa en los principios del Manifiesto Ágil y utiliza un enfoque iterativo e incremental para maximizar la eficiencia y la calidad del producto final. incluyen:

1. Roles:

- Product Owner: Representa a los stakeholders y es responsable de maximizar el valor del producto. Define los elementos del backlog y prioriza su entrega.
- Scrum Master: Facilita el proceso SCRUM, elimina obstáculos y asegura que el equipo siga las prácticas y reglas de SCRUM.
- Equipo de Desarrollo: Profesionales que entregan el incremento de producto. Son autoorganizados y multifuncionales.

2. Eventos de SCRUM:

- Sprint Planning: Reunión para planificar las actividades del próximo sprint, donde se seleccionan elementos del backlog para ser trabajados.
- Daily Standup (Reunión Diaria): Breve reunión diaria donde cada miembro del equipo informa sobre su progreso, desafíos y planificación para el día.
- Sprint Review: Revisión al final del sprint donde se presenta el trabajo completado y se recopila feedback para ajustes futuros.
- Sprint Retrospective: Reunión para reflexionar sobre el sprint recién concluido, identificar mejoras y planificar ajustes para el próximo sprint.

3. Artefactos de SCRUM:

- Product Backlog: Lista dinámica de características, mejoras y correcciones pendientes, priorizadas por el Product Owner.
- Sprint Backlog: Lista de tareas seleccionadas del Product Backlog para el sprint actual, creada por el equipo de desarrollo durante la Sprint Planning.
- Incremento: Producto potencialmente entregable al final de cada sprint, que debe ser funcional y cumple con los criterios de aceptación.

SCRUM ha demostrado ser eficaz para equipos de desarrollo en entornos dinámicos, fomentando la adaptabilidad y la entrega iterativa de productos de alta calidad. Sin embargo, su éxito depende de la dedicación y colaboración de todos los miembros del equipo.

Características Principales de Metodología SCRUM:

1. **Iterativo e Incremental:**
   - SCRUM organiza el trabajo en iteraciones llamadas "Sprints". Cada Sprint tiene una duración fija (generalmente de 2 a 4 semanas) y culmina con la entrega de un incremento de producto funcional. Este enfoque permite una mejora continua y la adaptación a los cambios durante el desarrollo.
2. **Roles Definidos:**
   - SCRUM define roles claramente definidos para los participantes del equipo:

Scrum Master: Facilita el proceso SCRUM, elimina obstáculos y asegura que el equipo siga las prácticas adecuadas.

Product Owner: Representa a los interesados y prioriza el backlog del producto.

Equipo de Desarrollo: Grupo multifuncional responsable de entregar el incremento de producto.

3. **Backlog del Producto:**
   - El Backlog del Producto es una lista priorizada de todas las funcionalidades, mejoras y tareas pendientes. El Product Owner es responsable de gestionar

y priorizar este backlog, asegurándose de que el equipo trabaje en las características más valiosas primero.

4. **Reuniones Ceremoniales:**

- SCRUM introduce varias reuniones regulares para facilitar la colaboración y la toma de decisiones:

Reunión de Planificación del Sprint: Define el trabajo que se realizará durante el Sprint.

Reuniones Diarias o de Estándares: Actualizaciones diarias para revisar el progreso y abordar posibles problemas.

Revisión del Sprint: Presenta el incremento de producto al final del Sprint.

Retrospectiva del Sprint: Evalúa el rendimiento del equipo y busca mejoras continuas.

5. **Transparencia y Visibilidad:**

- SCRUM promueve la transparencia en todos los niveles del proyecto. La información sobre el progreso, los obstáculos y los cambios está disponible para todos los miembros del equipo y las partes interesadas. Esto fomenta una comprensión común y una toma de decisiones informada.

6. **Adaptabilidad:**

- SCRUM está diseñado para ser adaptable a cambios en los requisitos o prioridades. La flexibilidad inherente permite que el equipo responda rápidamente a nuevas condiciones, proporcionando un mayor valor al cliente.

Proceso en Scrum:

1. **Equipos y Roles:**

- **Product Owner**: Es responsable de definir y priorizar los elementos del product backlog, representando los intereses del cliente.

- **Scrum Master:** Facilita el proceso Scrum, elimina obstáculos y asegura que el equipo siga las prácticas ágiles.

- Equipo de Desarrollo: Un grupo multifuncional y autoorganizado que entrega incrementos de producto al final de cada sprint.

2. Artefactos:

- **Product Backlog:** Una lista priorizada de todas las funcionalidades, mejoras y correcciones que se desean para el producto.
- **Sprint Backlog:** Una selección de elementos del product backlog que el equipo se compromete a completar durante un sprint.
- Incremento: El resultado del trabajo del equipo al final de cada sprint, que debe ser potencialmente entregable y añadir valor al producto.

3. Eventos:

- **Sprint:** Un período de tiempo fijo (generalmente de 2 a 4 semanas) durante el cual se realiza el trabajo y se entrega un incremento del producto.
- **Reunión de Planificación del Sprint:** Al inicio de cada sprint, el equipo selecciona elementos del product backlog y define la meta del sprint.
- **Daily Scrum:** Reunión diaria corta donde el equipo comparte actualizaciones sobre el progreso y discute cualquier impedimento.
- **Revisión del Sprint**: Al final de cada sprint, el equipo presenta el incremento y recibe retroalimentación del cliente o stakeholders.
- **Retrospectiva del Sprint:** Una reflexión al final de cada sprint para mejorar el proceso, identificar problemas y planificar ajustes.

4. Ciclo de Desarrollo:

- El proceso Scrum sigue un ciclo iterativo e incremental, donde los sprints sucesivos construyen sobre el trabajo previo, permitiendo adaptaciones continuas en función de los cambios en los requisitos o prioridades.

Requisitos y Roles:

3. **Requisitos:**

    - Los requisitos de Scrum se centran en proporcionar una estructura flexible que permita a los equipos adaptarse a cambios rápidos y entregar productos de alta calidad de manera regular

4. **Roles en Scrum:**

1. **Product Owner (Dueño del Producto):**

Responsabilidades:

- Definir y priorizar el backlog del producto.
- Tomar decisiones sobre las funcionalidades y la dirección del proyecto.
- Colaborar con los interesados para comprender y satisfacer sus necesidades.

2. **Scrum Master (Facilitador Scrum):**

Responsabilidades:

- Facilitar y garantizar el cumplimiento de los eventos de Scrum (reuniones, revisiones, etc.).
- Eliminar obstáculos y promover un entorno de trabajo colaborativo.
- Ayudar al equipo a comprender y adoptar los principios y prácticas ágiles.

3. **Equipo de Desarrollo:**

Responsabilidades:

- Diseñar, desarrollar y probar incrementos de producto.
- Colaborar estrechamente con el Product Owner para comprender los requisitos.
- Ser autoorganizado y multidisciplinario para entregar valor al final de cada iteración (Sprint).

Gestión y Seguimiento en Scrum:

1. **Reuniones Diarias (Daily Scrum):**

- Definición: Una reunión corta y enfocada que se lleva a cabo diariamente para que el equipo comparta su progreso, planifique el día y resuelva posibles obstáculos.

- Gestión y Seguimiento: Durante la reunión diaria, los miembros del equipo informan sobre lo que hicieron desde la última reunión, lo que planean hacer hasta la próxima y cualquier impedimento que puedan tener. Facilita la gestión del progreso diario y la identificación temprana de problemas.

2. **Revisión del Sprint (Sprint Review):**

- Definición: Una reunión al final de cada Sprint en la que el equipo presenta el trabajo completado y se recopila retroalimentación del cliente o stakeholders.

- Gestión y Seguimiento: La revisión del Sprint es un evento para evaluar el producto resultante y realizar ajustes según la retroalimentación recibida. Facilita la gestión del producto y permite a los stakeholders seguir de cerca el progreso.

3. **Retrospectiva del Sprint:**

- Definición: Una reunión que tiene lugar al final de cada Sprint en la que el equipo reflexiona sobre su rendimiento y busca mejoras continuas.

- Gestión y Seguimiento: La retrospectiva del Sprint es esencial para la gestión y seguimiento del proceso de desarrollo. Permite identificar áreas de mejora, resolver problemas internos y ajustar el proceso para incrementar la eficiencia.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: La Metodología Scrum es una de las más utilizadas en el desarrollo ágil de software debido a su enfoque iterativo e incremental.

- ✓ ¿Cuáles crees que son las ventajas de usar Scrum en un equipo de desarrollo?
- ✓ Explica cómo su estructura basada en Sprints, roles y reuniones diarias ayuda a mejorar la productividad y la calidad del producto final.

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Ensayo sobre la Metodología Scrum: Concepto y Características.

Escribe un ensayo de 1 a 2 páginas donde expliques:

- ✓ ¿Qué es Scrum y en qué se diferencia de otras metodologías ágiles?
- ✓ ¿Cuáles son sus principales características?
- ✓ ¿Cuáles son sus ventajas y desafíos en la gestión de proyectos?

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

IV. DATOS GENERALES DE LA UNIDAD 4

Información General

1.1.	Nivel	Segundo					
1.2.	Nombre Unidad	Aplicación de una metodología de desarrollo de software según tendencia					
1.3.	Tema 1 – Unidad	La Aplicabilidad					
1.4.	Tema 2 – Unidad	El Desarrollo					
1.5.	Unidad organizacional	Unidad Profesional					
1.6.	Unidad	4					
1.7.	Total, Horas Unidad	15					
1.8.	Detalle de horas Unidad	ACD	6	AA	4	APE	5

Resultado de Aprendizaje de Unidad

El resultado de aprendizaje de la Unidad 4 es que los estudiantes sean capaces de diseñar y ejecutar proyectos de desarrollo de software utilizando una metodología actualizada y basada en tendencias emergentes. Al finalizar la unidad, los estudiantes habrán adquirido las habilidades necesarias para preparar adecuadamente un proyecto de desarrollo de software, así como para planificar y llevar a cabo sprints de manera eficiente, lo que les permitirá mejorar la calidad y la eficiencia en sus proyectos de desarrollo de software.

Metodología de Unidad

Introducción y Objetivos: Comienza la clase virtual grabada con una introducción clara de lo que los estudiantes aprenderán en esa sesión. Establece los objetivos de aprendizaje para que los estudiantes sepan qué esperar.

Presentación de Contenido: Desarrolla el contenido de la asignatura, dividiéndolo en secciones o módulos. Utiliza una estructura organizada y lógica para abordar cada tema. Puedes combinar diferentes enfoques pedagógicos según el contenido, como explicaciones, ejemplos, demostraciones, gráficos y diagramas.

DESARROLLO

Unidad 4: Aplicación de una metodología de desarrollo de software según tendencia

Tema 1: La Aplicabilidad

Objetivo: Evaluar la aplicabilidad de metodologías de desarrollo de software según las tendencias actuales, considerando su impacto en la eficiencia, escalabilidad y adaptabilidad de los proyectos, con el propósito de seleccionar e implementar el enfoque más adecuado según los requerimientos y el entorno tecnológico.

En el ámbito del desarrollo de software, la aplicación de metodologías basadas en tendencias emergentes se ha convertido en un factor determinante para alcanzar el éxito en los proyectos. La rápida evolución de la tecnología y las cambiantes demandas del mercado requieren enfoques flexibles y adaptativos que permitan a los equipos de desarrollo mantenerse alineados con las necesidades del cliente y los avances tecnológicos. En esta unidad, exploraremos la aplicación práctica de una metodología de desarrollo de software según tendencia, centrándonos en dos aspectos cruciales: la preparación del proyecto y la planificación de sprints. Abordaremos cómo estos elementos se integran en el proceso de desarrollo para garantizar la eficiencia y la calidad en la entrega de productos de software. A través de la comprensión y la aplicación de estos conceptos, los profesionales del desarrollo de software estarán mejor preparados para enfrentar los desafíos y aprovechar las oportunidades que presenta el entorno actual de desarrollo de software.

La aplicación de metodologías de desarrollo de software según tendencias emergentes es fundamental para asegurar la eficiencia y la calidad en los proyectos de desarrollo de software. En esta unidad, nos enfocaremos en dos aspectos clave: la preparación del proyecto y la planificación de sprints.

Preparación del Proyecto:

La preparación adecuada del proyecto es un paso crucial en el proceso de desarrollo de software. Según Pressman (2014), la preparación del proyecto implica la identificación de los objetivos del proyecto, la definición clara de los requisitos del sistema y la asignación adecuada de recursos. Además, autores como Sommerville (2016) señalan que es esencial establecer un plan de trabajo detallado que incluya cronogramas, roles y responsabilidades, y un presupuesto adecuado.

La preparación del proyecto también implica la realización de un análisis de riesgos, como sugiere Boehm (1989). Esto implica identificar posibles obstáculos y problemas que podrían surgir durante el desarrollo del proyecto y desarrollar estrategias para mitigarlos.

Planificar un Sprint

Una vez que el proyecto está preparado, es crucial planificar los sprints de desarrollo de manera efectiva. Los sprints son períodos cortos y definidos de trabajo, durante los cuales se realizan actividades específicas de desarrollo de software. Según Schwaber y Sutherland (2017), la planificación de un sprint implica la identificación y priorización de las características a desarrollar, así como la estimación del esfuerzo necesario para completarlas.

Para planificar un sprint de manera efectiva, es fundamental contar con una comunicación clara y abierta entre todos los miembros del equipo. Autores como Cockburn (2006) destacan la importancia de la colaboración y la comunicación constante en equipos ágiles de desarrollo de software.

En Síntesis, la preparación del proyecto y la planificación de sprints son aspectos fundamentales en la aplicación de metodologías de desarrollo de software según tendencias emergentes. Estos procesos proporcionan la estructura necesaria para garantizar el éxito en el desarrollo de proyectos de software modernos y adaptados a las demandas del mercado actual.

Ejemplo con Caso de Estudio:

Imaginemos un startup tecnológico llamada "TechSol", dedicada al desarrollo de una aplicación de gestión de tareas para equipos de trabajo remotos. La empresa ha experimentado un crecimiento significativo en los últimos meses y está preparando el lanzamiento de una versión mejorada de su aplicación para satisfacer las crecientes demandas del mercado.

Preparación del Proyecto:

En primer lugar, TechSol debe preparar adecuadamente el proyecto para asegurar su éxito. Esto implica identificar claramente los objetivos del proyecto, que incluyen mejorar la interfaz de usuario, agregar nuevas funcionalidades y optimizar el rendimiento de la aplicación. Además, se deben definir los requisitos del sistema, como la compatibilidad con múltiples plataformas y la seguridad de los datos de los

usuarios. Para llevar a cabo esta preparación, el equipo de desarrollo de TechSol se reúne para realizar un análisis exhaustivo de los requisitos y establecer un plan de trabajo detallado que incluya los recursos necesarios y los plazos de entrega.

Planificar un Sprint:

Una vez que el proyecto está preparado, TechSol procede a planificar los sprints de desarrollo. En este caso, el equipo decide implementar una metodología ágil, como Scrum, para organizar el trabajo en sprints cortos y enfocados. Para el primer sprint, el equipo prioriza las tareas relacionadas con la mejora de la interfaz de usuario, que incluyen el diseño de nuevas pantallas y la optimización de la navegación. Se asignan las tareas a los miembros del equipo, se establecen los plazos de entrega y se define un objetivo claro para el sprint.

Mediante una adecuada preparación del proyecto y una planificación eficiente de los sprints, TechSol se encuentra en una posición sólida para desarrollar y lanzar con éxito la nueva versión de su aplicación de gestión de tareas. La aplicación de una metodología de desarrollo de software según tendencia, como la metodología ágil, ha permitido a TechSol adaptarse rápidamente a los cambios del mercado y entregar productos de alta calidad de manera constante.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: En la Metodología Scrum, el Sprint es un ciclo corto de trabajo en el que el equipo desarrolla una parte del producto.

- ✓ ¿Por qué crees que es importante dividir el desarrollo en Sprints?

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Elaboración de un Diagrama Detallado de las Fases de Scrum.

- ✓ Realiza un diagrama detallado que muestre todas las fases de Scrum, enfocándose en el desarrollo del Sprint.
- ✓ Incluye elementos como:
    - Product Backlog y Sprint Backlog
    - Desarrollo y pruebas en cada iteración
    - Revisión y retrospectiva del Sprint
    - Puedes hacerlo en herramientas como Draw.io, Lucidchart o PowerPoint y subirlo en PDF o imagen (JPG/PNG) a la plataforma.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.

2. Observa el video de la clase magistral.

3. Revisa en el calendario el día y hora en que se debe de realizar el Test.

4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

Tema 2: El Desarrollo

Objetivo: Analizar el proceso de desarrollo de software dentro de una metodología aplicada según tendencias actuales, comprendiendo sus fases, herramientas y buenas prácticas, con el propósito de optimizar la creación de soluciones tecnológicas eficientes y adaptadas a las necesidades del entorno.

El desarrollo del sprint

En el contexto del desarrollo de software, una práctica cada vez más extendida es la implementación del desarrollo ágil, que incluye el concepto de "sprints". Un sprint es un período de tiempo definido durante el cual se desarrolla y entrega un conjunto de funcionalidades específicas del software. Según Schwaber y Sutherland (2017), en su libro "The Scrum Guide", el desarrollo del sprint es una parte integral de la metodología Scrum, que es una de las metodologías ágiles más utilizadas en la industria del desarrollo de software. Durante un sprint, el equipo de desarrollo trabaja en estrecha colaboración para completar las tareas definidas en el backlog del sprint, con el objetivo de entregar un incremento potencialmente entregable al final del período. Este enfoque permite una rápida adaptación a los cambios y una entrega continua de valor al cliente.

Para garantizar una implementación efectiva del desarrollo del sprint, es crucial contar con un diagrama detallado que represente las fases de la metodología elegida. Según Leffingwell (2010), autor de "Scaling Software Agility", un diagrama detallado de fases proporciona una visión clara de cómo se estructuran y secuencian las actividades durante el desarrollo del sprint. Este diagrama suele incluir etapas como la planificación del sprint, donde se define el alcance y se asignan las tareas; la ejecución del sprint, donde se lleva a cabo el trabajo planificado; y la revisión del sprint, donde se realiza una demostración del trabajo completado y se recopila retroalimentación para futuras iteraciones.

Diagrama detallado de las fases de la metodología elegida

Es una representación visual que muestra las distintas etapas o fases que componen el proceso de desarrollo de software, junto con las actividades y tareas específicas que se llevan a cabo en cada una de estas fases. Este diagrama proporciona una guía clara y estructurada para el equipo de desarrollo, ayudando a entender el flujo de trabajo y a seguir el proceso de manera eficiente.

Por ejemplo, en el contexto de la metodología Scrum, un diagrama detallado de fases podría incluir las siguientes etapas:

1. **Planificación del Sprint**: Durante esta fase, el equipo se reúne para definir los objetivos del sprint y seleccionar las tareas que se abordarán en la iteración. Se establecen los criterios de aceptación para cada tarea y se asignan las responsabilidades.
2. **Desarrollo y Codificación**: En esta etapa, el equipo trabaja en la implementación de las funcionalidades definidas en el backlog del sprint. Se escriben y prueban el código, se integran las distintas partes del sistema y se realizan ajustes según sea necesario.
3. **Pruebas y Control de Calidad**: Durante esta fase, se llevan a cabo pruebas exhaustivas para asegurar la calidad del software desarrollado. Esto puede incluir pruebas unitarias, de integración, de regresión y de usuario, entre otras.
4. **Revisión y Demostración**: Al finalizar el desarrollo, el equipo realiza una revisión del trabajo completado y presenta una demostración del producto al cliente o al product owner. Se recopila feedback y se identifican posibles mejoras para futuras iteraciones.
5. **Retrospectiva del Sprint**: En esta última etapa, el equipo reflexiona sobre el proceso y las prácticas utilizadas durante el sprint. Se identifican aspectos positivos y áreas de mejora, y se definen acciones concretas para implementar en el próximo sprint.

ACTIVIDADES PRÁCTICAS

Tabla

Detalle de la Práctica

Practicas	Instrucciones	Duración de la Practica	Mecanismo de Evaluación
Foro de debate	Revisar Actividades Prácticas	0:30 mm	Por rúbricas
Práctica investiga: Taller/Ensayo/Investigación	Revisar Actividades Prácticas	3:30 h	Por rúbricas
Tutorías Prácticas	Revisar Actividades Prácticas	2:00 mm	Por participación en clase
Test evaluativo	Revisar Actividades Prácticas	0:30 mm	Formativa

Nota. Elaboración propia

Consideraciones Generales

1. Lee y revisa la Guía de Estudio.
2. Observa el video de la clase magistral.
3. Participa activamente en el FORO DEBATE.
4. Revisa en el calendario el día y hora en que se realizará las actividades prácticas y la tutoría.
5. Realizar la práctica propuesta por el docente.
6. Revisa las Rúbricas acorde a la actividad prácticas.
7. Toma los principales apuntes que vayas rescatando del texto.
8. Reflexiona sobre los temas tratados en la clase.
9. Sube el documento en formato PDF en la plataforma Moodle.
10. Revisa la calificación de tus actividades prácticas.

FORO DE DEBATE

Tema: En el desarrollo ágil de software, el Sprint es una fase clave que permite la entrega incremental del producto.

- ✓ ¿Por qué crees que dividir el desarrollo en Sprints mejora la eficiencia del equipo y la calidad del software?

Instrucciones para Participar:

- Exponga su punto de vista acerca del tema.
- Comenta las respuestas de otros participantes.
- Promueve un ambiente respetuoso y constructivo.

PRÁCTICA INVESTIGATIVA

Taller: Análisis de un Caso Práctico sobre la Implementación de Sprints en un Proyecto.

- ✓ Investiga un caso real donde una empresa haya implementado Sprints dentro de una metodología ágil.
- ✓ Explica cómo se organizó el ciclo de desarrollo, qué resultados obtuvo y qué mejoras se lograron en la optimización del software.

Consigna:

- El trabajo de investigación debe presentarse de manera estructurada y coherente para comunicar efectivamente los hallazgos y conclusiones.
- Asegúrese de que el estilo y formato del trabajo sigan las pautas de citación y presentación establecidas por la institución.
- Utilice un lenguaje claro y evite jergas o términos técnicos innecesarios.
- Revise y corrija la redacción y la gramática antes de la presentación final (ver anexo).

PRÁCTICA TEST

Antes de realizar el test, es importante considerar varios aspectos para aprovechar al máximo la experiencia.

Consideraciones para realizar el test

1. Lee y revisa el documento.
2. Observa el video de la clase magistral.
3. Revisa en el calendario el día y hora en que se debe de realizar el Test.
4. Realiza el Test de la unidad considerando lo Siguiente:

- No cambiarte entre paginas durante la realización del Test.
- Revisa que tu conexión a internet sea estable.
- Recuerda que todo el contenido está inmerso dentro del documento base.

5. Revisa tu puntuación, de inmediato al finalizar de la actividad.

V. Bibliografía

Bibliografía básica

Roger, P. (2021). Ingeniería de Software: Un Enfoque Práctico, 9 edición. MCGRAW-HILL.

Winters, T., Manshreck, T., & Wright, H. (2022). Ingeniería de Software en Google: Lecciones sobre Programación Aprendidas a lo largo del tiempo. MARCOMBO.

Bibliografía de consulta

Sommerville, I. (2007). Ingeniería Del Software, Pearson

Maida, E. G. (2015). Metodología de desarrollo de software. Metodología de desarrollo de software. Obtenido de https://repositorio.uca.edu.ar/handle/123456789/522

Manuel Fernando Caro Piñeres, R. E. (2009). Diseño de Software educativo basado en competencias. Dialnet, 98. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=3223340

Gamboa, J. Z. (2018). Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software. Dialnet, 20-33. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=6777227

Anexo 1: Modelo de Foro

Título del Foro:

[Nombre del Tema o Pregunta]

Desarrollo:

[Breve descripción del tema o pregunta]

Reglas del Foro:

Regla 1: Antes de participar en el debate, realiza una investigación exhaustiva sobre el tema en cuestión. Comprender los hechos, datos y argumentos relacionados con el tema fortalecerá tu posición y mejorarán la calidad de la discusión.

Regla 2: Muestra respeto hacia tus compañeros de debate, incluso si no estás de acuerdo con sus opiniones. Escucha activamente sus argumentos y evita interrupciones. Expresa tus puntos de vista de manera respetuosa y constructiva.

Regla 3: Contribuye de manera equitativa al debate, evitando la monopolización de la conversación. Respeta el tiempo asignado y asegúrate de que otros también tengan la oportunidad de expresar sus opiniones.

Anexo 2: Esquema de Ensayo

Caratula: (Escribe aquí los datos referentes a la materia, nombres completos, tema, periodo, Institución Educativa y logos)

Título:

(Escribe aquí el título de tu ensayo)

Introducción:

Desarrollo:

Conclusión:

Bibliografía

Anexo 3: Esquema de Taller/Investigación

Caratula:

(Escribe aquí los datos referentes a la materia, tema, periodo, Institución Educativa y logos)

Título:

(Escribe aquí el título principal de tu investigación)

Temas a investigar:

- **Tema 1:**
- **Tema 2:**
- **Tema 3:**

Conclusión:

Bibliografía

Anexos

Anexo 4: Rúbrica de Foro

INDICADORES	MS 5 PUNTOS	S 3 PUNTOS	PS 2 PUNTOS	PUNTAJE
Comprensión del tema	La respuesta argumentada demuestra conocimiento concreto del tema tratado.	La respuesta argumentada demuestra conocimiento general de la mayoría del tema.	La respuesta argumentada demuestra un conocimiento limitado del tema.	20 %
Cuestionamientos y criterios	Presentó las dudas e inquietudes que se le presentaron durante el desarrollo del tema, o al menos sacó diferentes conclusiones, que las expuso en el debate.	Presenta al menos 2 preguntas o argumentos, aunque demuestra que no ha comprende ni revisado en profundidad el material.	No presenta dudas ni argumentos, aunque conozca del tema, no aporta al debate.	20 %
Redacción y ortografía	No presenta errores ortográficos ni de redacción.	Presenta algunos errores ortográficos y/o de redacción.	Presenta muchos errores de redacción y de ortografía.	20 %
Interacción: relación/ confrontación con intervenciones de otros/as	Comparte activamente con de sus compañeros los temas planteados y resuelve dudas de sus compañeros con argumentos valederos en más de 2 intervenciones.	Comparte con de sus compañeros los temas planteados, argumentando en al menos de 2 intervenciones.	Plantea dudas a sus compañeros con el tema, sin embargo, no participa ni responde con argumentos lógicos. Solo interviene una vez.	20 %
Participación	Participa con más de 2 intervenciones, además de su contestación.	Participa al menos con 5 Intervenciones, además de su contestación.	No interviene en otra participación a más de la suya.	20 %
			TOTAL=	100 %

Puntaje: 5 puntos

Anexo 5: Rubrica de Ensayo

INDICACIONES	MS 10 PUNTOS	S 6 PUNTOS	PS 2 PUNTOS	PUNTAJE
Enfoque o Idea principal	La idea principal nombra el tema del ensayo y esquematiza los puntos principales a discutir.	La idea principal nombra el tema del ensayo.	La idea principal esquematiza algunos o todos los puntos a discutir, pero no menciona el tema.	20 %
Transiciones o Conectores	Una variedad bien pensada de transiciones fue usada. Estas muestran claramente como están las ideas conectadas.	Las transiciones muestran cÃ³mo están las ideas conectadas, pero hay muy poca variedad.	Algunas transiciones funcionan bien, pero la conexiÃ³n entre las ideas no es clara.	20 %
Conclusión	La conclusión es fuerte y dejó al lector con una idea absolutamente clara de la posición del autor. Un parafraseo efectivo de la idea principal empieza la conclusión.	La conclusión es evidente. La posición del autor es parafraseada en las primeras dos oraciones de la conclusión.	La posición del autor es parafraseada en la conclusión, pero no al principio de la misma.	20 %
Gramática y ortografía	El autor no comete errores de gramática ni de ortografía que distraen al lector del contenido del ensayo.	El autor comete 1 a 2 errores gramaticales u ortográficos que distraen al lector del contenido del ensayo.	El autor comete 3 a 4 errores gramaticales u ortográficos que distraen al lector del contenido del ensayo.	20 %
Fuentes	Todas las fuentes usadas para las citas, las estadísticas y los hechos son creíbles y están citadas correctamente.	Todas las fuentes usadas para las citas, las estadísticas y los hechos son creíbles y la mayoría está citada correctamente.	La mayoría de las fuentes usadas para las citas, las estadísticas y los hechos es creíble y está citada correctamente.	20 %
			TOTAL=	100 %

Puntaje: 10 puntos

Anexo 6: Rúbrica de Taller/Investigación

INDICADORES	MS 10 PUNTOS	S 6 PUNTOS	PS 2 PUNTOS	PUNTAJE
Organización de las ideas	El texto es coherente y tiene sentido toda la lectura.	Presenta ideas aisladas, no enlazadas entre sí.	El texto no tiene coherencia y las ideas no están claras.	30 %
Ortografía	Tiene 4 o menos faltas ortográficas	Tiene entre 5 y 9 faltas ortográficas.	Tiene más de 10 faltas ortográficas.	30 %
Calidad de la información presentada	La información está claramente relacionada con el tema y proporciona varios niveles de relación con sus diferentes entornos.	La información da respuesta a la consigna principal del artículo, mas no aporta niveles de relaciones de la comunicación con el entorno próximo.	La información proporcionada guarda relación con las categorías, aunque su calidad argumentativa es deficiente.	40 %
			TOTAL=	100 %

Puntaje: 10 puntos