ANÁLISIS DE SISTEMAS II

CARRERA:

ANÁLISIS DE SISTEMAS

SEMESTRE:

CUARTO

Mcs Adriana González

INDICE

Contenido

Introducción a la materia

Hola mis alumnos, en Sistemas II", aprenderemos a tomar el Rol del analista de sistemas, quien es un profesional responsable y dedicado en analizar, diseñar, implementar y mantener los S.I., como también a utilizar las tecnologías de la información y comunicación (TIC); entendiendo que, su trabajo implica comprender las necesidades y procesos de la organización, traduciéndolas en soluciones técnicas, prácticas y efectivas.

ADRIANA E. GONZALEZ R.
- Magister en Gerencia Estratégica -

Una publicación de

Instituto Universitario de Tecnología para la Informática

DEFINICIÓN DEL PROYECTO

Unidad I

¿Qué es un proyecto?

Un proyecto es un **conjunto ordenado de actividades con el fin de satisfacer ciertas necesidades o resolver problemas específicos**. Un proyecto es un plan de trabajo.

Todo proyecto tiene su ciclo de vida: inicio, planificación, ejecución y cierre del conjunto de procesos que lo componen.

El ciclo de vida del Proyecto abarca: desde que el Proyecto nace hasta que el Proyecto finaliza.

En nuestro caso para dar cumplimiento a la materia, desarrollaremos un proyecto relacionado con Sistemas de Información.

¿Cuáles son los tipos de sistemas de información?

Existen 7 tipos de sistemas de información:

- ✓ Sistemas **de procesamiento de transacciones.**
- ✓ Sistemas **de información gerencial**.
- ✓ Sistemas **de control de procesos de negocio.**
- ✓ Sistemas **de información de marketing**.
- ✓ Sistemas **de colaboración empresarial**.
- ✓ Sistema **de apoyo a la toma de decisiones**.
- ✓ Sistemas **de información ejecutiva.**

Que es el Ciclo de Vida de un Sistema:

El Ciclo de Vida de un Sistema de Información o Software (SDLC: Systems Devetopment Life Cycle), es un sistema, automatizado, que engloba a personas, máquinas y/o métodos organizados para recopilar, procesar, transmitir datos que representan información.

En el Ciclo de Vida de un Sistema de Información, en su fase de planificación, es la que nos permite conocer sobre el alcance que tendrá el proyecto, que puntos abarcará, los posibles riesgos que puede llegar a presentar y el orden en el cual se ejecutarán todas las tareas en el proceso de su creación.

El desarrollo de software puede ser difícil de administrar debido a los requisitos cambiantes, los avances de la tecnología y la colaboración interfuncional. La metodología del ciclo de vida del desarrollo de software (SDLC) ofrece un marco de administración sistémico con entregas específicas en cada etapa del proceso de desarrollo de software. Como resultado, todas las partes interesadas establecen por adelantado los objetivos y requisitos de desarrollo del software y también cuentan con una planificación para conseguirlo.

Los detalles del proceso SDLC varían para equipos diferentes. Sin embargo, a continuación, se describen algunas fases comunes del SDLC:

El ciclo de vida del desarrollo de software (SDLC) es un proceso rentable y eficiente en términos de tiempo empleado por los equipos de desarrollo para diseñar y crear software de alta calidad. El objetivo del SDLC es minimizar los riesgos del proyecto por medio de una planificación anticipada que permita que el software cumpla las expectativas del cliente durante la fase de producción y posteriormente. Esta metodología establece una serie de pasos que dividen el proceso de desarrollo de software en tareas que se pueden asignar, completar y medir.

ACTIVIDAD EVALUATIVA DE LA UNIDAD Nro.1:

1.- Que es proyecto?

2.- Cuales serían los tipos de Sistemas Información

3.- Que entiende usted por Ciclo de Vida Sistema?

4.- ¿Por qué es importante el SDLC?

5.- ¿Según lo leído, cuáles serían las fases del SDLC?

METODOLOGIAS DEL SOFTWARE

Unidad II

Metodologías del Software Introducción a las metodologías utilizadas para organizar, desarrollar y gestionar proyectos de software. Estas guían las fases del ciclo de vida del software, desde la planificación hasta el mantenimiento.

El ciclo de vida del software

Representa una estructura organizada que guía las distintas etapas por las que pasa un proyecto de software, desde su concepción inicial hasta su fase de mantenimiento. Comienza con la planificación, en donde se identifican y analizan los requisitos del sistema, incluyendo las necesidades de los usuarios y las limitaciones técnicas o presupuestarias. En esta etapa se determina el alcance del proyecto, estableciendo metas claras y documentadas que orienten al equipo a lo largo del proceso.

Luego sigue el análisis, una fase esencial en la que se profundiza en los requisitos previamente establecidos. Aquí se definen los aspectos funcionales y

no funcionales del sistema, se identifican posibles riesgos y se evalúa la viabilidad del proyecto desde diferentes ángulos.

En la fase de diseño, se divide en dos grandes partes. Por un lado, se realiza un diseño preliminar que define la arquitectura del sistema, estableciendo cómo los distintos componentes interactuarán entre sí. Por otro lado, se lleva a cabo un diseño detallado que especifica minuciosamente los componentes individuales, como las bases de datos, las interfaces de usuario y los algoritmos que garantizarán el funcionamiento eficiente del sistema. Posteriormente, se pasa a la implementación, en donde se traduce el diseño en código funcional. Los programadores crean los módulos del sistema siguiendo el diseño establecido, integrándolos cuidadosamente para asegurar que todas las partes trabajen de manera cohesiva.

Una vez que el sistema está desarrollado, se somete a pruebas rigurosas en las que se verifica que cumpla con los requisitos iniciales. Se realizan diferentes tipos de pruebas, como las pruebas unitarias, que verifican cada módulo individual, y pruebas de integración, que aseguran que los módulos funcionan correctamente en conjunto. La implantación sigue a las pruebas, momento en el que el software es implementado en el entorno de producción y puesto a disposición de los usuarios finales. Durante esta etapa también se ofrece capacitación a los usuarios, si es necesario, para garantizar que puedan usar el sistema de manera efectiva. Finalmente, el ciclo de vida culmina con la fase de mantenimiento, en donde se corrigen errores que puedan surgir después del lanzamiento y se realizan ajustes o mejoras basadas en nuevas necesidades o cambios en el entorno operativo del cliente.

Metodologías Tradicionales

Por otro lado, las metodologías tradicionales, también conocidas como enfoques "predictivos", se basan en procesos estructurados y bien definidos. Su característica principal es que las fases del ciclo de vida del desarrollo del software, como análisis, diseño, desarrollo, pruebas e implementación, se completan de manera secuencial y no se vuelven a repetir una vez terminadas. Un ejemplo destacado es el modelo **Cascada (Waterfall)**, en el que el progreso fluye de manera lineal, como una cascada, desde una fase a la siguiente. Estas metodologías son más adecuadas para proyectos con requisitos claros y bien definidos desde el inicio, donde es poco probable que se produzcan cambios significativos durante el desarrollo. Aunque pueden ser menos flexibles que las ágiles, su estructura rigurosa facilita la planificación detallada, el control de costos y el cumplimiento estricto de los plazos.

Metodologías Ágiles

Las metodologías ágiles son enfoques flexibles y adaptativos para el desarrollo de software que priorizan la colaboración constante, la entrega continua de valor y la capacidad de responder rápidamente a los cambios. Estas metodologías operan a través de iteraciones o sprints cortos, en los cuales se desarrolla, prueba y ajusta el software de manera incremental. Un principio clave es la interacción entre individuos sobre los procesos rígidos, así como la importancia de obtener retroalimentación frecuente de los clientes para asegurar que el producto final cumpla con sus expectativas.

Ejemplos populares incluyen **Scrum**, donde los equipos trabajan en ciclos de tiempo definidos llamados sprints, y **Kanban**, que organiza las tareas visualmente para optimizar el flujo de trabajo. Las metodologías ágiles son ideales para proyectos donde los requisitos no son completamente definidos desde el principio y pueden evolucionar con el tiempo, ya que fomentan la flexibilidad y la comunicación abierta en el equipo.

Las Diferencias Entre Las Metodologías Ágiles Y Tradicionales

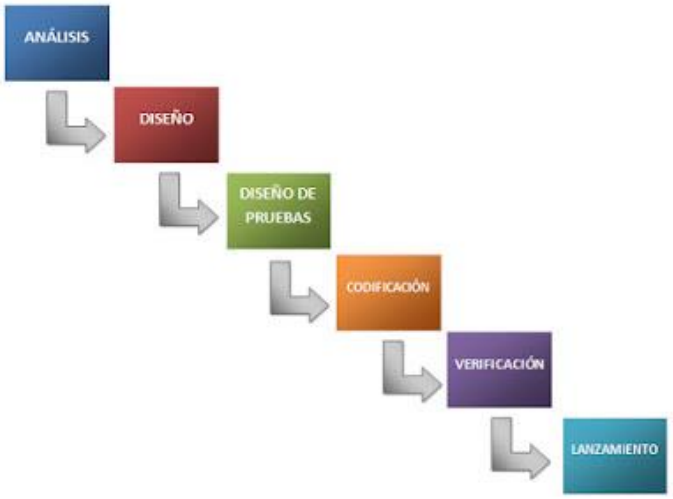
Metodologías Tradicionales y Ágiles

Para desarrollar un software de calidad primero debemos seguir ciertos pasos que nos permitan cumplir con todo lo que el software necesita satisfacer a nuestro cliente, donde la decisión más importante desde mi punto de vista sería escoger la metodología apropiada para un equipo en un determinado proyecto.

Las metodologías se pueden clasificar en dos grupos. Las metodologías tradicionales, que buscan siempre una fuerte planificación y documentación durante todo el desarrollo, y las metodologías ágiles, en las que se enfoca al desarrollo de software el cual es incremental, cooperativo, sencillo y adaptado.

Metodologías tradicionales

Método de cascada, método considerado base de los otros tradicionales Las metodologías tradicionales o denominadas a veces como metodologías pesadas.



buscan siempre llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto al pié de la letra, definido todo esto, en la fase inicial del desarrollo del proyecto.

Otra de las características importantes dentro de este las tradicionales, es el alto costos que significaría realizar un cambio a nuestro proyecto y la falta de flexibilidad en proyectos donde el entorno es cambiante.

Las metodologías tradicionales para la creación de software se caracterizan por su enfoque estructurado y secuencial. Aquí tienes las más utilizadas:

Modelo en Cascada (Waterfall):

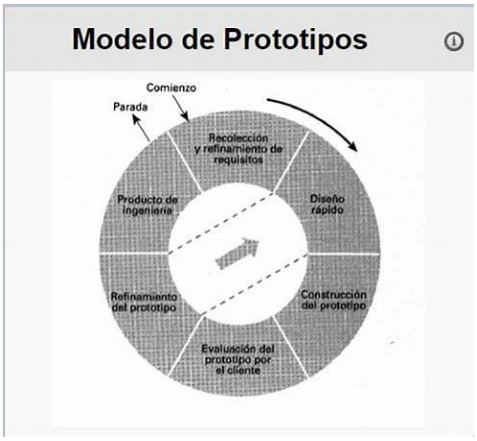
Es el enfoque más clásico y sigue un proceso lineal.

- **Las fases incluyen:** análisis de requisitos, diseño, implementación, pruebas, despliegue y mantenimiento.
- Cada fase debe completarse antes de pasar a la siguiente.
- Ideal para proyectos con requisitos bien definidos y poco propensos a cambios.

Modelo en Espiral:

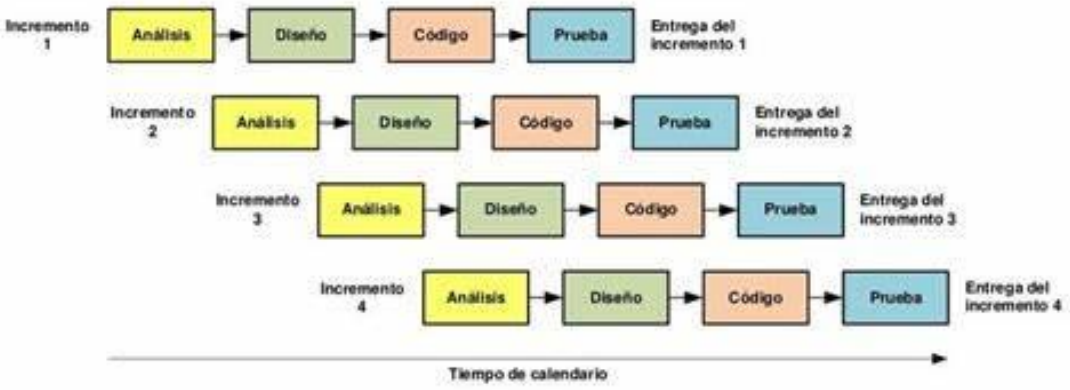
- Combina elementos del modelo en cascada con iteraciones.
- Se enfoca en la gestión de riesgos y permite revisiones constantes en cada ciclo.

- Es útil para proyectos complejos donde los requisitos pueden evolucionar.



Modelo de Prototipado:

- Consiste en crear prototipos funcionales para obtener retroalimentación temprana de los usuarios.
- Ayuda a clarificar requisitos y mejorar el diseño antes del desarrollo completo.
- Es adecuado para proyectos donde los requisitos iniciales no están completamente definidos.



Modelo Incremental:

- El software se desarrolla y entrega en incrementos o versiones funcionales.
- Cada incremento añade nuevas funcionalidades.

Desarrollo Rápido de Aplicaciones (RAD):

- Se enfoca en la rapidez y eficiencia mediante el uso de herramientas y técnicas avanzadas.
- Requiere una alta colaboración con los usuarios finales.
- Es ideal para proyectos con plazos ajustados.

Estas metodologías tradicionales son especialmente útiles en proyectos donde los requisitos son claros desde el principio y los cambios son mínimos.

Viabilidad en la actualidad del Uso de estas Metodologías

Hoy en día, las metodologías tradicionales se combinan con enfoques ágiles o híbridos para aprovechar sus puntos fuertes mientras se mitiga su rigidez. Por ejemplo:

El modelo en cascada se usa en sectores como la construcción y desarrollo de sistemas críticos.

Modelos incrementales permiten combinar planificación tradicional con entregas más rápidas.

En general, si el entorno es altamente dinámico o los requisitos cambian con frecuencia, los enfoques modernos como Agile, Scrum o DevOps tienden a ser más recomendados. No obstante, para proyectos con alta estabilidad y

claridad desde el inicio, las metodologías tradicionales pueden ser una opción sólida.

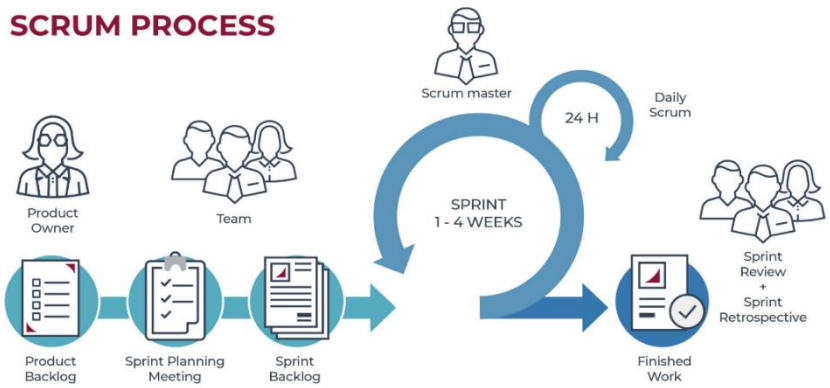
Metodologías ágiles

Ciclo de entrega general de las metodologías ágiles

las metodologías ágiles nacen como respuesta a los problemas que se presentan en las metodologías tradicionales y se basa en dos aspectos fundamentales, retrasar las decisiones y la planificación adaptativa. las metodologías están basadas en adaptabilidad de los procesos de desarrollo.

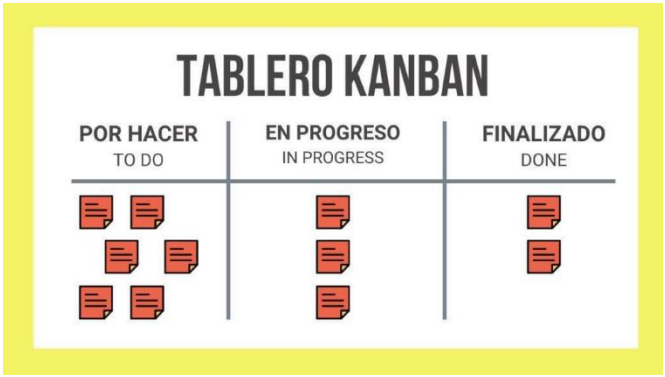
Las metodologías ágiles ven más importante la capacidad de adaptarse a los cambios que el seguir un plan estricto de desarrollo.

Las metodologías ágiles más utilizadas son aquellas que permiten adaptarse rápidamente a los cambios y optimizar la gestión de proyectos. Aquí tienes las principales:



Scrum:

- Se basa en ciclos cortos llamados "sprints" que suelen durar entre 1 y 4 semanas.
- Incluye reuniones diarias (daily stand-ups) para evaluar el progreso y resolver problemas.
- Es ideal para proyectos que requieren entregas frecuentes y retroalimentación constante.



Kanban:

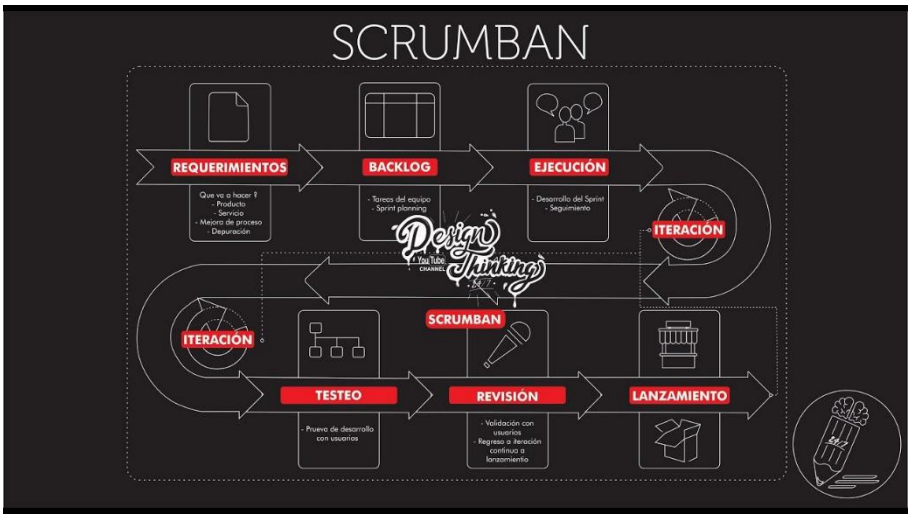
- Utiliza tableros visuales para gestionar tareas y flujos de trabajo.
- Se enfoca en la mejora continua y la reducción de tiempos de espera.
- Es útil para proyectos con procesos repetitivos y en entornos de producción.

Extreme Programming (XP):

- Promueve prácticas como la programación en pareja y las pruebas continuas.
- Se centra en la calidad del código y la satisfacción del cliente.
- Es especialmente popular en el desarrollo de software.

Lean Development:

- Busca eliminar desperdicios y maximizar el valor entregado al cliente.
- Se enfoca en la eficiencia y la mejora continua.
- Es adecuado para proyectos que requieren optimización de recursos.



Scrumban:

- Combina elementos de Scrum y Kanban.
- Ofrece flexibilidad en la planificación y ejecución de tareas.
- Es ideal para equipos que buscan un enfoque híbrido.

Estas metodologías son ampliamente utilizadas en diversos sectores, no solo en el desarrollo de software, debido a su capacidad para adaptarse a entornos dinámico

Diferencias entre las metodologías ágiles y tradicionales.

Metodologías ágiles	Metodologías tradicionales
Están preparadas para cambios durante el proyecto	Son poco flexibles a los cambios
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte activa en el proceso de desarrollo	El cliente interactúa con el equipo solo mediante reuniones de entregas
Grupos pequeños, 10 integrantes o menos y trabajando en el mismo sitio en el cual todos tienen conocimiento sobre todo el proceso de desarrollo	Grupos grandes y posiblemente distribuidos donde a cada interante se le asignan tareas especificas
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Ventajas/Desventajas de las metodologías ágiles

Ventajas

La primera y la que más resalta sobre las metodologías tradicionales es que ofrecen una rápida respuesta a cambios de requisitos a lo largo del desarrollo del proyecto gracias a su proceso iterativo, ya que es tan importante el realizar un buen análisis de requisitos, como tener la posibilidad de modificarlos de forma rápida evitando grandes pérdidas en cuanto a costes, motivación, tiempo.

El cliente puede colaborar, puede observar cómo va avanzando el proyecto y dar sus opiniones sobre su evolución gracias a las interacciones frecuentes del equipo con el cliente. Esto le da tranquilidad, basándose en las 2 ventajas anteriores, se puede descubrir una nueva ventaja, que al utilizar las metodologías ágiles, los cambios que el cliente quiera aplicar al proyecto van a causar meno

s trabajo adicional que en las tradicionales, ya que se va a entregar en un corto plazo de tiempo una pequeña versión funcional del proyecto al cliente, y si éste quiere cambiar algo, solo se habrá perdido unas semanas de trabajo.

Con las tradicionales las entregas se realizaban después de haber avanzado bastante en el proyecto, eso quiere decir que el equipo ha estado trabajando meses para que despues un cambio que quiera realizar el cliente, ocasione la pérdida de todo ese trabajo.

Buscan la simpleza a la hora de trabajar para así eliminar el trabajo

innecesario y ahorrar tiempo.

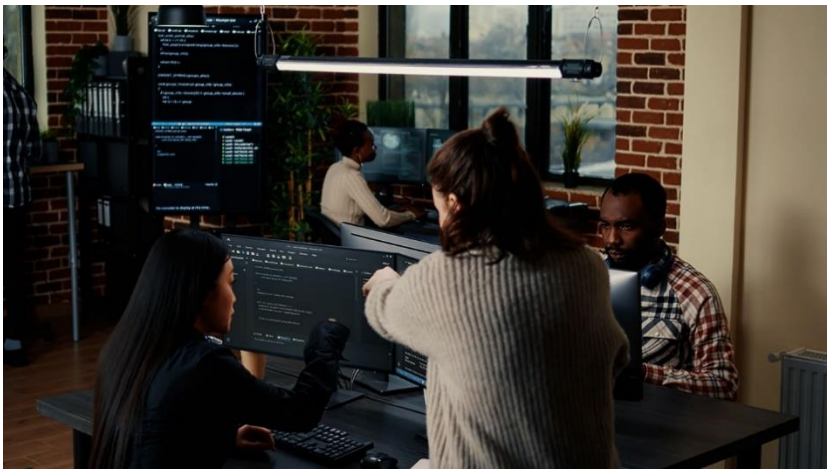
Desventajas

Está claro que en la vida todo tiene ventajas y desventajas, aunque las ventajas sean muy llamativas a la hora de inclinarse por una metodología ágil antes que una tradicional, estas metodologías ágiles también presentan inconvenientes que hay que asumir cuando se decide trabajar con ellas. Estos son:

- Se tiene poca documentación del diseño. Al no haber documentación es el código (y los comentarios que hagamos en el) lo que se toma como documentación.
- Problemas que se presentan a partir de la comunicación oral. ya que si tenemos un contrato escrito, no se puede borrar, cuando el contrato es hablado es muy fácil caer en la ambigüedad de lo que el cliente o el equipo de trabajo dijo y lo que su contra parte puede interpretar.

- Restricciones en cuanto a tamaño de los proyectos, ya que si es un proyecto q conlleva mucho tiempo las ágiles están enfocadas a proyectos q se puedan hacer de forma rápida.

- Problemas derivados del fracaso de los proyectos ágiles. Si un proyecto fracasa se tiene poca documentación, la cual sería de vital importancia si se busca saber la causa del fracaso, La comprensión del proyecto y todo su funcionamiento queda en las mentes de los desarrolladores.



Los equipos de trabajo bajo metodologías del software

Desempeñan un papel crucial en la organización y desarrollo de proyectos exitosos. Su estructura, dinámica y funcionamiento varían según el tipo de metodología aplicada, ya sea ágil, tradicional o híbrida.

En las metodologías ágiles, los equipos se caracterizan por ser multifuncionales y autoorganizados. Cada miembro tiene diversas habilidades y contribuye activamente al cumplimiento de los objetivos. Los roles clave incluyen, por ejemplo, el Product Owner, quien gestiona las prioridades y los requisitos del cliente; el Scrum Master, que actúa como facilitador asegurando que el equipo siga las prácticas ágiles; y los desarrolladores, encargados de construir el producto. La comunicación es constante y colaborativa, basada en reuniones regulares como las Daily Stand-ups, lo cual fomenta la transparencia y la rápida identificación de bloqueos o problemas.

En las metodologías tradicionales, la estructura de los equipos suele ser jerárquica y los roles están bien definidos. Cada miembro se enfoca en una fase

específica del ciclo de vida del software, como análisis, diseño, desarrollo o pruebas. La comunicación sigue una cadena de mando clara y suele ser formal, frecuentemente liderada por un gerente de proyecto que supervisa las tareas y asegura el cumplimiento del cronograma y presupuesto. Este enfoque es ideal para proyectos donde los requisitos son claros y no se espera que cambien significativamente.

El Estudio Preliminar del Proyecto

Es una etapa clave en el proceso de desarrollo de software que permite evaluar la viabilidad y definir los fundamentos del proyecto antes de avanzar a las fases de diseño y construcción. Durante esta etapa, se recopilan y analizan todos los aspectos necesarios para garantizar que el proyecto se ajuste a los objetivos propuestos, identificando posibles riesgos, restricciones y oportunidades.

Se comienza con la definición de los objetivos del proyecto, donde se establece qué se pretende lograr con el software, considerando las necesidades de los usuarios finales o del cliente. Luego, se realiza una recopilación inicial de requisitos, tanto funcionales (lo que debe hacer el sistema) como no funcionales (rendimiento, seguridad, escalabilidad, entre otros). También es fundamental evaluar el alcance del proyecto, delimitando lo que será incluido o excluido, para evitar desviaciones posteriores.

El estudio preliminar incluye además un análisis de los recursos necesarios, tanto humanos como técnicos, así como una evaluación de los costos estimados y los tiempos requeridos para cada etapa. Esto puede acompañarse de un análisis de factibilidad en sus tres dimensiones principales:

técnica (determinando si se cuenta con la tecnología y conocimientos para llevarlo a cabo), económica (evaluando si el proyecto es rentable) y operativa (confirmando si se puede implementar en el entorno previsto).

Finalmente, esta etapa concluye con la documentación de toda la información recolectada, usualmente en un informe preliminar, que servirá como base para tomar decisiones y planificar el resto del desarrollo. Es un proceso crucial para minimizar riesgos y asegurar que el proyecto esté bien fundamentado desde el principio

Los Tiempos de Entrega

En un proyecto de desarrollo de software representan la duración estimada para completar cada fase del ciclo de vida del proyecto, desde la planificación hasta la implementación y el mantenimiento. Gestionar adecuadamente estos tiempos es crucial para garantizar que se cumplan las expectativas del cliente y se mantengan los costos bajo control.

En metodologías tradicionales, como el modelo Cascada, los tiempos de entrega se planifican de manera detallada al inicio del proyecto mediante cronogramas que especifican cuánto tiempo tomará cada fase, como el análisis, diseño, desarrollo y pruebas. Una vez establecidos, estos tiempos son rígidos y no se modifican fácilmente, lo que permite tener un control más predecible del proyecto. Herramientas como diagramas de Gantt se utilizan para visualizar las tareas y sus dependencias, asegurando que cada etapa se complete dentro del plazo estipulado.

En las metodologías ágiles, los tiempos de entrega se gestionan de forma iterativa y más flexible. El desarrollo se divide en ciclos cortos llamados sprints (generalmente de 1 a 4 semanas), y al final de cada sprint se entrega un

incremento funcional del software. Esto permite ajustes rápidos basados en la retroalimentación del cliente, asegurando entregas continuas de valor. En este enfoque, los tiempos no se calculan al detalle para todo el proyecto desde el inicio, sino que se van ajustando en función de las prioridades y la velocidad del equipo (velocidad medida a través de historias de usuario completadas).

.

El estudio de factibilidad en la realización de un sistema

Es un análisis integral que se realiza al inicio de un proyecto para evaluar si es viable llevarlo a cabo desde diferentes perspectivas, antes de invertir recursos significativos. Este proceso asegura que las decisiones tomadas estén bien fundamentadas y ayudan a reducir los riesgos asociados al proyecto.

Durante el estudio, se consideran principalmente tres tipos de factibilidad: técnica, económica y operativa. En la factibilidad técnica, se analiza si las herramientas, tecnologías, infraestructura y habilidades disponibles son suficientes para implementar el proyecto. Esto incluye evaluar si el equipo cuenta con el conocimiento técnico necesario y si es factible usar las tecnologías propuestas.

Factibilidad Técnica, Económica y Operativa

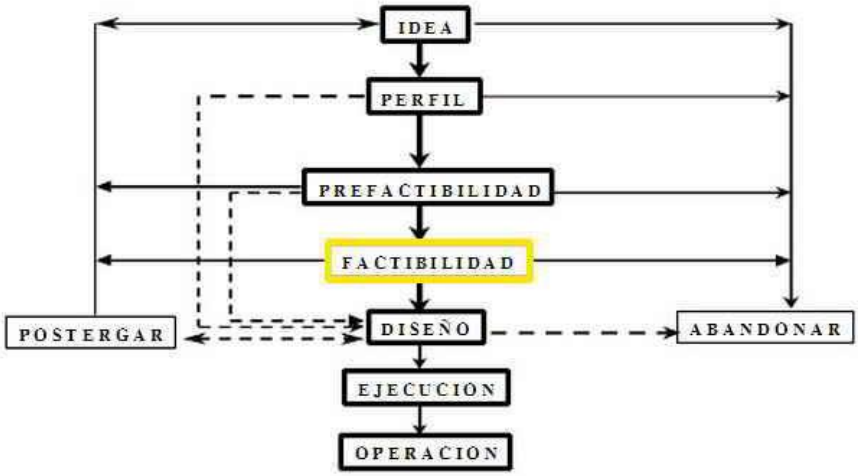
La factibilidad técnica, económica y operativa son pilares fundamentales en el análisis de viabilidad de cualquier proyecto. Cada una aborda un aspecto crítico que determina si el proyecto puede ejecutarse de manera exitosa.

Factibilidad técnica se centra en evaluar si el proyecto puede llevarse a cabo con los recursos tecnológicos y habilidades disponibles. Esto incluye analizar si la infraestructura tecnológica existente, como hardware, software, redes y equipos, es adecuada para las necesidades del proyecto. También considera las capacidades del equipo humano encargado de desarrollar el sistema, revisando si cuentan con la experiencia técnica y las herramientas necesarias para implementar el proyecto sin enfrentar limitaciones insuperables.

En la factibilidad económica, se determinan los costos asociados al desarrollo del proyecto y los posibles beneficios que generará. Este análisis busca garantizar que el proyecto sea rentable, considerando presupuestos, retornos de inversión y recursos financieros disponibles. Por último, la factibilidad

operativa evalúa si el proyecto es práctico y funcional dentro del entorno en el que será implementado. Esto implica analizar si el software podrá integrarse con los procesos actuales de la organización y si cumplirá con las expectativas de los usuarios finales.

Examina si el proyecto puede integrarse y funcionar eficientemente dentro del entorno organizacional para el cual está diseñado. Este análisis evalúa si los usuarios finales podrán adoptar el sistema fácilmente, si la solución es compatible con los procesos existentes y si satisface las necesidades y expectativas del cliente. También incluye la evaluación de riesgos operativos y el impacto que podría tener la implementación del proyecto en las operaciones diarias



Actividad Evaluativa De La Unidad

1. ¿Cuáles son las principales características de las metodologías ágiles en el desarrollo de software?
2. Describe la diferencia entre las metodologías ágiles y las metodologías tradicionales.
3. Explica cuál es el rol del Product Owner dentro de un equipo de trabajo que utiliza metodologías ágiles.
4. Menciona las etapas principales del modelo Cascada como ejemplo de metodología tradicional.
5. ¿Qué aspectos se deben evaluar al formar equipos de trabajo bajo metodologías de software?
6. ¿Qué se evalúa en un estudio de factibilidad?
7. ¿Qué factor es crucial para garantizar tiempos de entrega en un proyecto ágil?
8. Qué característica define a un equipo de trabajo efectivo en metodologías ágiles?

ANÁLISIS Y DIAGNOSTICO

Unidad III

¿Qué es un análisis?

El análisis permite la comprensión de un determinado fenómeno o evento de una manera integral, sin aislarlo de otros que ocurren en el escenario social.

El análisis tiene en cuenta el contexto específico en el que se recopilan los datos, en lugar de simplemente analizar los datos de forma aislada. Este contexto podría incluir información sobre la hora, la ubicación y otros factores que pueden influir en los datos.

¿Qué es el análisis de un sistema de información?

También se denomina análisis de sistemas a una de las etapas de construcción de un sistema informático, que consiste en relevar la información actual y proponer los rasgos generales de la solución futura.

El análisis y diseño de sistemas es el proceso que se lleva a cabo para recolectar información necesaria que nos permitirá conocer el problema que se tiene y a su vez plantear una posible solución a través de la utilización de un sistema de información.

Este proceso cuenta con 5 fases: a) Análisis y Requerimientos;
b) Diseño del sistemas; c) Codificación del sistema; d) Pruebas del sistemas;

ENTREVISTA CON EL CLIENTE

Para llevar a cabo una entrevista efectiva con el cliente orientada al análisis y levantamiento de requerimientos, puedes seguir un enfoque estructurado que combine buenas prácticas y metodologías de desarrollo de software. Aquí tienes algunos pasos clave que podrían ayudarte:

1. Preparación previa a la entrevista

Definir objetivos: Identifica qué información necesitas obtener, como requisitos funcionales, no funcionales y expectativas del cliente.

Recopilar información preliminar: Investiga sobre el negocio del cliente y los problemas que el software debe resolver.

Preparar preguntas clave: Divide tus preguntas en categorías (procesos actuales, problemas, metas, prioridades, etc.).

2. Conducción de la entrevista

Establecer empatía: Inicia la conversación explicando el propósito de la reunión y creando un ambiente cómodo para el cliente.

Técnicas de entrevista:

Utiliza preguntas abiertas para explorar las necesidades del cliente (Ej.: "¿Qué proceso actual le gustaría mejorar?").

Complementa con preguntas cerradas para confirmar detalles específicos (Ej.: "¿Este proceso se realiza a diario?").

Escucha activa: Toma notas, valida lo que has entendido y pide ejemplos concretos si es necesario.

Uso de diagramas: Herramientas como diagramas de flujo o mapas mentales pueden ayudar a visualizar procesos o ideas.

3. **Levantamiento y validación de requerimientos**

Clasifica los requerimientos: Diferencia entre:

- **Requerimientos funcionales**: ¿Qué debe hacer el software? (Funciones específicas).

- **Requerimientos no funcionales:** Rendimiento, seguridad, escalabilidad, etc.

Validación con el cliente: Resume lo discutido y confirma que los requisitos capturados sean correctos y completos

DEFINICIÓN DE REQUERIMIENTOS:

Un requerimiento es una característica que debe incluirse en un nuevo sistema y puede *consistir* en una forma de captar o procesar datos, producir información, controlar una actividad o dar apoyo a una tarea.

¿Qué son los requerimientos de un sistema de información?

Requisitos de información, describen la información que debe almacenar y gestionar el sistema ara dar soporte a los procesos de negocio. Reglas de negocio, describen las reglas o restricciones por cuyo cumplimiento debe velar el sistema.

Flujo de actividades:

Requerimientos Funcionales y no Funcionales

En el desarrollo de software, es fundamental contar con requisitos claros y bien definidos para el éxito de cualquier proyecto. Estos requisitos suelen dividirse en dos categorías: funcionales y no funcionales. Si bien ambos son esenciales, cumplen distintas funciones a la hora de dar forma al producto final. Requerimientos funcionales delinear lo que debe hacer un sistema, centrándose en acciones, comportamientos e interacciones específicas.

Por otro lado, requerimientos no funcionales Describir el rendimiento del sistema, abordando atributos de calidad como el rendimiento, la seguridad y la escalabilidad. Comprender la diferencia entre estos dos tipos de requisitos es fundamental para garantizar que el sistema no solo cumpla con su propósito previsto, sino que también proporcione una experiencia de usuario fluida y confiable.

Los requerimientos funcionales y no funcionales son componentes esenciales en el desarrollo de sistemas y software, ya que definen lo que el sistema debe hacer y cómo debe hacerlo. Aquí te explico cada uno en detalle:

Requerimientos Funcionales

Los requerimientos funcionales describen las funciones específicas que el sistema debe realizar. Responden a la pregunta: ¿Qué debe hacer el sistema?

Ejemplos:

- Registrar nuevos usuarios.
- Procesar transacciones de pago.
- Generar reportes de ventas.
- Enviar notificaciones por correo electrónico.
- Validar contraseñas al iniciar sesión.
- Características principales:
- Relacionados directamente con el objetivo del sistema.
- Se derivan de las necesidades del cliente y los casos de uso.
- Son verificables a través de pruebas funcionales.

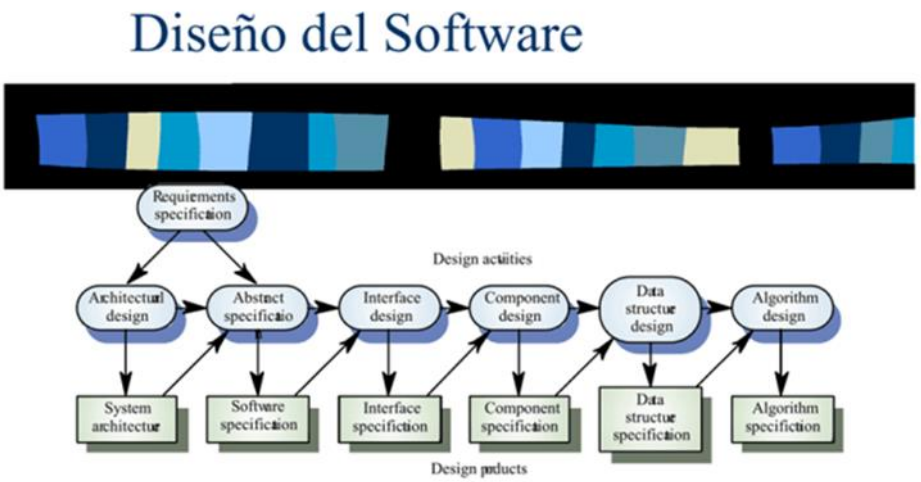
Requerimientos No Funcionales

Los requerimientos no funcionales especifican las cualidades y restricciones del sistema. Responden a la pregunta: ¿Cómo debe operar el sistema?

Ejemplos:

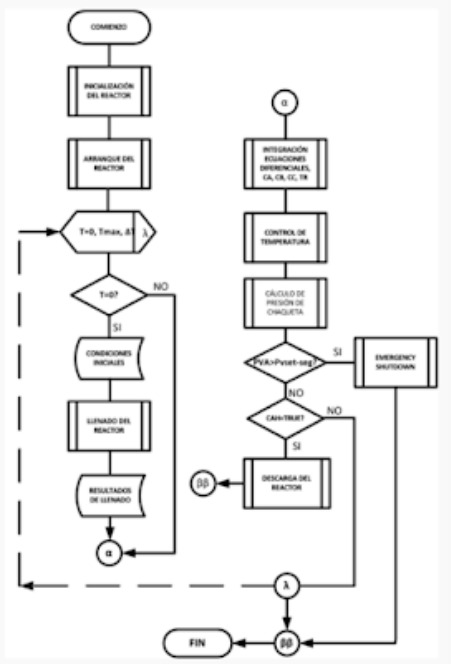
- El sistema debe responder en menos de 2 segundos.
- Garantizar la seguridad de los datos mediante encriptación.
- Ser compatible con navegadores móviles y de escritorio.
- Tener una disponibilidad del 99.9% al año.
- Características principales:
- Enfocados en la experiencia del usuario y el rendimiento del sistema.
- Pueden incluir aspectos de seguridad, escalabilidad, usabilidad, eficiencia

DISEÑO DEL SOFTWARE:



¿Qué es el diseño en el desarrollo de software?

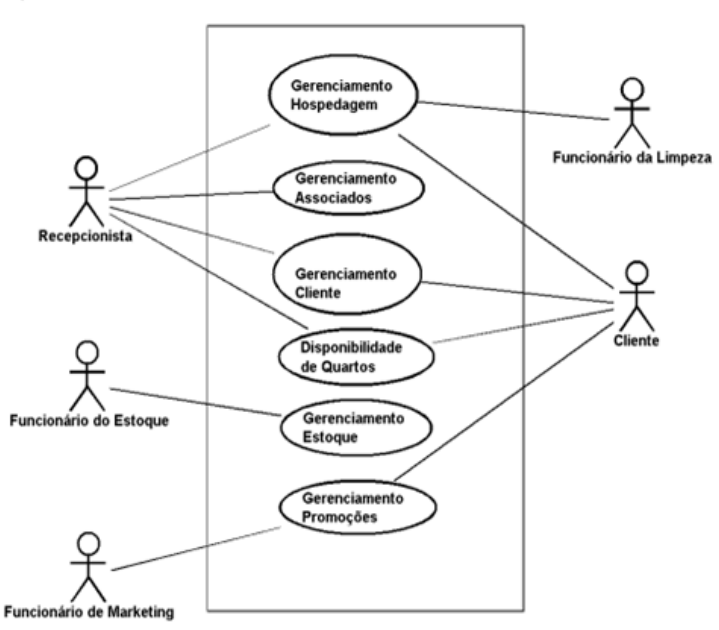
El diseño de software es el proceso de visionado y definición de soluciones software a uno o más conjuntos de problemas. Uno de los componentes principales del diseño de software es la especificación de requisitos del software (ERS).



Uso de Diagramas de Flujo

Los diagramas de flujo son herramientas visuales que representan de manera secuencial y lógica los pasos de un proceso o sistema. En el desarrollo de software, son útiles para detallar la lógica detrás de algoritmos, procedimientos o flujos de trabajo, permitiendo identificar posibles problemas y optimizar procesos antes de implementarlos.

Estos diagramas emplean símbolos estándar como óvalos para representar inicio y fin, rectángulos para acciones o tareas, rombos para decisiones y flechas para el flujo entre los elementos. Su uso facilita la comunicación dentro del equipo de trabajo y con los interesados en el proyecto, ya que ofrece una representación

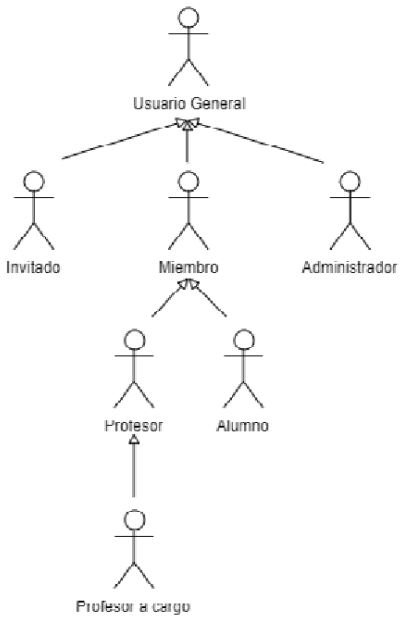


clara y comprensible del funcionamiento del sistema o de procesos específicos.

Casos de Uso UML

Los diagramas de casos de uso UML (Unified Modeling Language) se utilizan para modelar las interacciones entre los actores y el sistema, identificando las funcionalidades clave que debe ofrecer. Estos diagramas ayudan a comprender y documentar los requerimientos funcionales del sistema desde la perspectiva del usuario. Cada caso de uso representa un escenario en el que un actor interactúa con el sistema para cumplir un objetivo.

Los componentes principales de un diagrama de casos de uso incluyen los actores (usuarios o sistemas externos que interactúan con el sistema), los casos de uso (acciones o funcionalidades), y las relaciones entre ellos, como asociaciones, inclusiones o extensiones. Este enfoque asegura que el diseño del sistema esté alineado con las necesidades del usuario final.



Actores del Sistema

Los actores del sistema son las entidades externas que interactúan con el software, desempeñando un rol esencial en los casos de uso y en la definición de los requerimientos del sistema. Estos pueden ser personas, como los usuarios finales, u otros sistemas que se conectan o comunican con el software desarrollado.

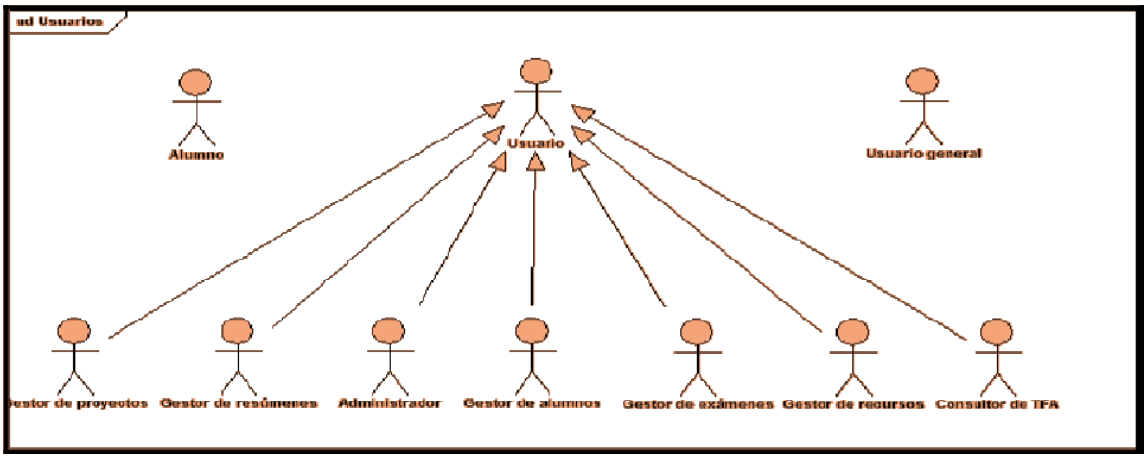
Los actores se clasifican en actores primarios, que inician interacciones para cumplir sus objetivos (por ejemplo, un cliente que realiza una compra en línea), y actores secundarios, que apoyan las operaciones del sistema, pero no lo inician directamente (por ejemplo, un sistema de pago externo). Identificar correctamente a los actores del sistema garantiza que todas las funcionalidades necesarias sean consideradas y que el software cumpla con los requisitos planteados.}

Objetivo del diseño de software

El diseño del software ayuda a modificar elementos que se utilizan para armar el formato que tendrá el programa, la importancia del diseño del software se puede definir en una sola palabra: calidad; ya que dentro del diseño es donde se establece la calidad del proyecto; E

diseño es la única manera de materializar con precisión los requerimientos del cliente.

El proceso de Diseño es un conjunto de pasos repetitivos que permiten al diseñador describir todos los aspectos del Sistema a construir.



¿Cuál es la importancia del desarrollo de software y cuáles son sus ventajas?

El desarrollo de software permite crear sistemas con altos estándares de seguridad que protegen la información empresarial y la de los clientes; Además, facilita la implementación de medidas de cumplimiento normativo específicas de la industria, asegurando que la empresa cumpla con regulaciones y estándares.

ACTIVIDAD EVALUATIVA DE LA UNIDAD Nro.4:

1.- Que es un prototipo de sistemas?

2.- ¿Qué es refinar prototipos?

3.- ¿Qué es el diseño preliminar:?

4.- Cual es objetivo del diseño de software?

5.-¿Cuál sería la importancia del desarrollo de software?

DISEÑO DETALLADO:

Unidad V

¿Qué es el diseño en el desarrollo de software?

El diseño de software es el proceso de visionado y definición de soluciones software a uno o más conjuntos de problemas. Uno de los componentes principales del diseño de software es la especificación de requisitos del software (ERS).

El diseño preliminar se centra en la transformación de los requisitos en los datos y la arquitectura del software. El diseño detallado se ocupa del refinamiento y de la representación arquitectónica que lleva a una estructura de datos refinada y a las representaciones algorítmicas del software.

Etapa de diseño de software?

En la fase de diseño, los ingenieros de software analizan los requisitos e identifican las mejores soluciones para crear el software. Por ejemplo, pueden plantearse la integración de módulos ya existentes, elegir la tecnología e identificar herramientas de desarrollo.

El diseño del software ayuda a modificar elementos que se utilizan para armar el formato que tendrá el programa, la importancia del diseño del software se puede definir en una sola palabra: **calidad**; ya que dentro del diseño es donde se establece la calidad del proyecto.

El diseño es la única manera de materializar con precisión los requerimientos del cliente. El proceso de Diseño es un conjunto de pasos

repetitivos que permiten al diseñador describir todos los aspectos del Sistema a construir.

Los cuatro elementos del diseño son:

1. El diseño de los datos: Define la relación entre cada uno de los elementos estructurales del programa.

2. El diseño arquitectónico: Describe cómo se comunica el software consigo mismo, con los sistemas que operan junto con él y con los operadores y usuarios que lo emplean.

3. El diseño de la interfaz: Describe la forma como el sistema interactúa con el usuario más que la apariencia del sistema.

4. El diseño a nivel de componente o de procedimientos: Es una descripción procedimental de cada una de las partes que fueron especificadas en el diseño arquitectónico.

¿Qué son las entradas y salidas de un sistema de información?

Entrada: Son los datos que se introducirán al procesador. Existe un video de la universidad Galileo donde ejemplifican dicho modelo: https://youtu.be/GFYNy9WGtdE?t=13.;**Proceso:** Es la acción que acepta datos de entrada para ser procesados. **Salida:** Es la información, es decir, es el resultado de los datos ya procesados.

El diseño o modelo "entrada, proceso y salida" describe el flujo de información en un sistema o proceso. Las entradas son los datos suministrados, el proceso involucra transformaciones y operaciones, y las salidas son los resultados obtenidos. Este modelo es útil para analizar, diseñar y optimizar sistemas en diversos campos.

Objetivo de un sistema de información:

El objetivo principal de un sistema de información es producir datos para respaldar las operaciones y la toma de decisiones de la empresa. Estos serán los resultados del sistema. Muchos de estos resultados se basarán en la información ingresada al sistema. Información proporcionada por empleados, clientes y otras fuentes de entrada .

¿Cuál es la diferencia entre el software de diseño y el software de análisis?

La diferencia principal es que la producción de un análisis de software se compone de problemas más pequeños para solucionar, además, el análisis no tendría que ser diseñado de manera muy distinta por miembros de equipo diferente. En contraste, el diseño de software se enfoca a capacidades y, por

lo tanto, existen y existirán múltiples diseños para el mismo problema. Podemos encontrar ejemplos de diseño en: sistemas operativos, sitios web, dispositivos móviles o incluso el nuevo paradigma de computación.

El diseño del software ayuda a modificar elementos que se utilizan para armar el formato que tendrá el programa, la importancia del diseño del software se puede definir en una sola palabra: calidad; ya que dentro del diseño es donde se establece la calidad del proyecto; El diseño es la única manera de materializar con precisión los requerimientos del cliente.

ACTIVIDAD EVALUATIVA DE LA UNIDAD Nro.5:

1.- ¿Cuál es el objetivo principal de realiza entrevistas con el cliente durante la creación de sistemas?¿Qué es el diseño en el desarrollo de software?

2.- Qué se hace en la etapa de diseño de software?

3.- ¿Qué son las entradas y salidas de un sistema de información?

4.- ¿Cuál es el objetivo de un sistema de información:

5.- ¿Cuál es la diferencia entre el software de diseño y el software de análisis?

6-¿Qué representan los casos de uso en el desarrollo de sistemas?

7-¿Qué información se representa en un diagrama de flujo?

8-¿Qué son los requerimientos funcionales de un sistema?

CODIFICACIÓN DEL SISTEMA:

La codificación es una etapa fundamental en el proceso de creación de un software o sistema, ya que aquí se plasma en código todas las especificaciones, requerimientos y diseños definidos previamente. A continuación, un desglose de su relevancia y características:

¿Qué es la codificación?

Es el proceso donde los desarrolladores traducen los requisitos funcionales y no funcionales, así como los diagramas de diseño, a un lenguaje de programación. Esto permite que el sistema pueda ejecutarse y cumplir con las necesidades del usuario.

La etapa de codificación es una parte integral de las metodologías de desarrollo de software. Dependiendo del enfoque adoptado, esta fase puede variar en su ejecución:

Codificación en Metodologías Ágiles:

- Enfoque iterativo y colaborativo.
- Los desarrolladores trabajan en ciclos cortos (sprints), entregando incrementos funcionales del software.
- Pruebas e integración continua durante la codificación.

Metodologías Tradicionales (como el modelo en cascada):

- La codificación ocurre después de finalizar completamente las etapas de análisis y diseño.

- Existe una mayor estructura y formalidad en comparación con las metodologías ágiles.

Metodologías Híbridas (como RUP):

- Combina lo mejor de ambos mundos, estructurando la codificación en iteraciones mientras se mantiene un marco general definido.

- Independientemente de la metodología, esta fase siempre busca convertir los diseños y los requerimientos en un producto funcional.

Componentes Clave de la Codificación

Selección de herramientas y tecnologías:

- Lenguajes de programación: Java, Python, C#, entre otros.

- Entornos de desarrollo integrados (IDEs) como Visual Studio o Eclipse.

- ***Estándares de codificación*:**
- Uso de buenas prácticas para garantizar claridad, calidad y mantenibilidad del código.

- **Modularidad:**
- División del sistema en módulos independientes para facilitar pruebas e integración.

Pruebas de calidad:

- Implementación de pruebas unitarias para verificar el correcto funcionamiento de cada componente.

Documentación:

- Incluir comentarios y guías en el código para facilitar futuras modificaciones o depuración.

Lugar de la Codificación en el Ciclo de Desarrollo de Software

Previo a la codificación:

- Se realiza el análisis de requerimientos, los casos de uso y el diseño del sistema.

Posterior a la codificación:

- Inicia la etapa de pruebas integradas y de sistema, donde se validan y verifican las funcionalidades implementadas.

ACTIVIDAD EVALUATIVA DE LA UNIDAD:

1. ¿Qué aspectos debes considerar al seleccionar un lenguaje de programación para la etapa de codificación?

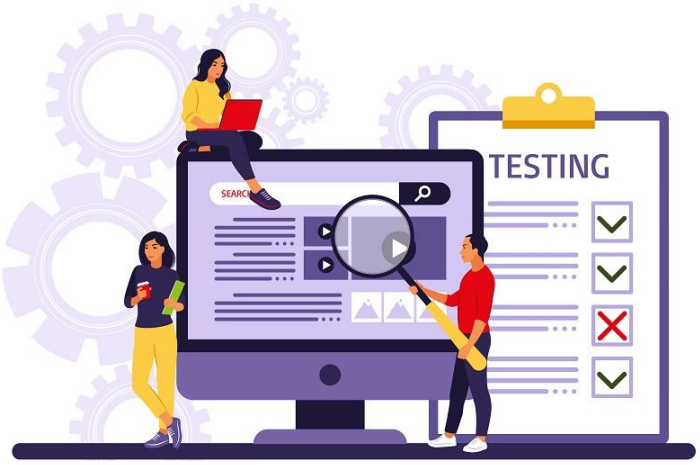
2. ¿Cómo aseguras la calidad del código durante su desarrollo?

3. ¿Qué importancia tiene la documentación en la etapa de codificación y cómo debe realizarse?

4. ¿Qué herramientas o tecnologías se utilizan comúnmente en la etapa de codificación?

5. ¿Cómo influye la modularidad en la eficiencia del proceso de codificación?

PRUEBAS DEL SISTEMA:



La realización de pruebas unitarias es una etapa esencial en el ciclo de vida del desarrollo de software. Estas pruebas aseguran que cada módulo, componente o unidad del software funcione correctamente de manera aislada, cumpliendo con los requerimientos previamente definidos.

¿Qué son las pruebas unitarias?

Son un tipo de prueba de software que verifica la funcionalidad de las partes más pequeñas y manejables de un sistema (unidades o módulos). Estas unidades pueden incluir funciones, métodos, clases o componentes independientes.

Características clave de las pruebas unitarias

Aislamiento:

- Se prueban unidades individuales, sin interacción con otros módulos o sistemas.

Automatización:

- Frecuentemente se utilizan herramientas o frameworks para automatizar las pruebas, como JUnit, NUnit o pytest.
-

Rapidez y precisión:

- Estas pruebas son rápidas y ayudan a identificar errores específicos en el código.

Ejecutadas por desarrolladores:

- Generalmente, son realizadas por el mismo desarrollador que escribió el código, para garantizar que cada pieza funcione correctamente.

Objetivos de las pruebas unitarias

- Verificar que cada unidad cumpla con los requerimientos funcionales.
- Detectar errores o defectos desde las primeras etapas del desarrollo.
- Garantizar la calidad del código antes de la integración con otros módulos.
- Facilitar el mantenimiento y refactorización del código, ya que las pruebas aseguran que los cambios no rompan la funcionalidad existente.

Etapas dentro de las metodologías de desarrollo

Metodologías Tradicionales:

- Las pruebas unitarias se realizan después de completar la etapa de codificación de todo el sistema.

Metodologías Ágiles:

- Las pruebas unitarias son continuas e iterativas, ejecutándose en cada sprint o incremento del software.

Modelos Incrementales o Iterativos:

- Se realizan en cada iteración para verificar la funcionalidad de los nuevos módulos antes de integrarlos con los previos.

ACTIVIDAD EVALUATIVA DE LA UNIDAD:

1. ¿Cuál es el objetivo principal de las pruebas unitarias en el desarrollo de software?

2. ¿Qué herramientas o frameworks se pueden utilizar para realizar pruebas unitarias?

3. ¿Cómo influye el aislamiento de módulos en la efectividad de las pruebas unitarias?

4. ¿Qué diferencias existen entre las pruebas unitarias manuales y automatizadas?

5. ¿Qué beneficios aporta la ejecución de pruebas unitarias en las etapas iniciales del desarrollo?

Recursos Interactivos

Evaluacion Y Elaboracion De Proyectos Ii:
Https://Blogereducativo.Wordpress.Com/Conal/

Sistemas De Información En Empresas: Definición, Tipos Y Ejemplos
Https://Blog.Hubspot.Es/Marketing/Sistema-Informacion

¿Qué Es El Ciclo De Vida Del Desarrollo De Software (Sdlc)?
Https://Aws.Amazon.Com/Es/What-Is/Sdlc/

Las 7 Etapas Del Ciclo De Vida Del Desarrollo De Software
Https://Www.Avenuglobal.Com/Blog/Cuales-Son-Las-Etapas-Del-Ciclo-De-Vida-Del-Desarrollo- De-Software-Etapas-Del-Sdlc

El Análisis De Un Sistema Informático
Https://Www.Euroinnova.Com/Blog/Analisis-De-Un-Sistema

Análisis Del Sistema De Información (Asi)
Https://Manuel.Cillero.Es/Doc/Metodologia/Metrica-3/Introduccion/Procesos- Principales/Desarrollo/Asi/

¿Qué Es Un Diagrama De Contexto (Y Cómo Crear Uno)?
Https://Es.Venngage.Com/Blog/Diagrama-De-Contexto/

¿Qué Es Un Diagrama Entidad Relación?
Https://Miro.Com/Es/Plantillas/Diagrama-Entidad-Relacion/

Regístrate Y Crea Tu Propio Diagrama Entidad Relación
Regístrate Y Crea Tu Propio Diagrama Entidad Relación

Calidad De Datos
Https://Www.Google.Co.Ve/Books/Edition/Calidad_De_Datos/Yi6fdwaaqbaj?Hl=Es- 419&Gbpv=1

Metodologías De Desarrollo De Software: ¿Qué Son?
Https://Www.Santanderopenacademy.Com/Es/Blog/Metodologias-Desarrollo-Software.Html

Bentancur, J. (s.f.). Requerimientos Funcionales y No Funcionales.

ABooch, G. (1996). Análisis y Diseño Orientado a Objetos con Aplicaciones. Segunda Edición. Editorial Addison-Wesley/Díaz de Santos.

Aranda Córdoba, J. R. (2015). Desarrollo y reutilización de componentes software y multimedia mediante lenguajes de guión (UF1842). IC Editorial.

Aranda Vera, Á. (2016). Instalación y parametrización del software (UF1893). IC Editorial.cademia.edu.. Recuperado de Academia.edu.

Modelos Y Metodoogias De Desarrollo
Https://Youtu.Be/Uhzgysui2a8?T=16

Metodologías De Desarrollo De Software
Https://Www.Youtube.Com/Watch?V=Uwfhwu7j49e&T=1014s

Diseño Y Desarrollo De Software
Https://Blogereducativo.Wordpress.Com/Diseno-Y-Desarrollo-Del-Software/

¿Qué Es El Diseño De Software?
Https://Youtu.Be/56kafd1dqx8?List=Pl4nuziygwyoregw9wxxhbfjh95krqqrpn&T=3

Metodologías Del Desarrollo De Software - Análisis Y Diseño
Https://Youtu.