RESUMEN 1ER PARCIAL IS-1

- ## UNIDAD 1.

INGENIERÍA DE SOFTWARE:

Aplica teorías, métodos y herramientas para el desarrollo de software profesional. Es la disciplina de la ingeniería que se ocupa de todos los aspectos de la producción de software. Tiene que ver con el desarrollo de software rentable.

El costo del software suele ser mayor que el costo del hardware y el mantenimiento es más costoso que el desarrollo.

PRODUCTOS DE SOFTWARE:

- ***Productos genéricos:*** son sistemas que se comercializan y venden a cualquier cliente. El desarrollador decide que funcionalidades va a tener el producto. Por ejemplo: Excel.
- ***Productos personalizados:*** es software que encarga un cliente específico para satisfacer sus propias necesidades. El cliente decide que funciones va a tener el producto. Por ejemplo: Sistema ERP para un negocio.

ACTIVIDADES DEL PROCESO DE SOFTWARE:

1. ***Especificación:*** clientes e ingenieros definen el software que se producirá y las restricciones.
2. ***Diseño y desarrollo:*** etapa donde se diseña y programa el software.
3. ***Validación:*** comprobación de que el software hace lo que quiere el cliente.
4. ***Evolución del software:*** etapa donde se modifica el software en respuesta a las necesidades cambiantes del cliente o el mercado.

ATRIBUTOS DE UN BUEN SOFTWARE:

- ***Mantenimiento:*** debe escribirse de forma que pueda evolucionar para satisfacer necesidades cambiantes de los clientes.
- ***Confiabilidad y seguridad:*** no debe causar daño físico o económico en caso de fallo del sistema. Los usuarios malintencionados no deben poder acceder o dañar el sistema.
- ***Eficiencia:*** debe optimizar el uso de los recursos del sistema.
- ***Aceptabilidad:*** debe ser aceptable para el tipo de usuario para el que está diseñado. Debe ser comprensible, utilizable y compatible con otros sistemas.

TIPOS DE APLICACIONES:

- ***Aplicaciones autónomas:*** se ejecutan en un equipo local y no es necesario estar conectado a una red.
- ***Aplicaciones interactivas:*** se ejecutan en un equipo remoto y los usuarios pueden acceder desde sus propios ordenadores.
- ***Sistemas embebidos:*** controlan y gestionan dispositivos de hardware.
- ***Sistemas procesamiento por lotes:*** diseñados para procesar los datos en grandes lotes.
- ***Sistemas de entretenimiento:*** destinados a entretener al usuario.
- ***Sistemas para simulación:*** destinados a modelar procesos físicos.
- ***Sistemas de adquisición de datos:*** recopilan datos utilizando sensores y envían estos a otros sistemas para su procesamiento.

FUNDAMENTOS DE LA ING. DE SOFTWARE:

- Poseer metodología para el desarrollo del sistema.
- Fiabilidad y rendimiento.
- Comprender los requisitos y que debe hacer el software.
- Si es posible, reutilizar software en vez de crear uno nuevo.

- **UNIDAD 2**

EL PROCESO DEL SOFTWARE:

Son actividades necesarias para desarrollar un sistema de software. Existen distintos tipos de procesos:

- ***Proceso dirigido por plan:*** todas las actividades se planifican con antelación y el progreso se mide en contra de este plan.
- ***Procesos ágiles:*** la planificación es gradual y es más fácil cambiar el proceso para reflejar los requisitos cambiantes de los clientes.

MODELOS DE PROCESOS DE SOFTWARE:

- ***Modelo de cascada:***
  Es un modelo dirigido por plan, se implementa en proyectos grandes, críticos, donde trabaja mucha gente y todo tiene que estar documentado (principal ventaja). Posee distintas fases:
  1. Análisis de requerimientos y su definición.
  2. Diseño del sistema y del software.
  3. Implementación y pruebas de unidades.
  4. Integración y pruebas del sistema.
  5. Operación y mantenimiento.

  Las desventajas de un modelo de cascada son:
  - Inflexible, la división del proyecto en fases estructuradas hace difícil responder a las necesidades cambiantes de los clientes. Pocos sistemas tienen requisitos estables.
  - El cliente no tiene interacción.
  - Una fase tiene que ser completada para poder pasar a la siguiente.
- ***Desarrollo incremental:***
  Puede ser dirigido por plan o ágil, se basa en la reutilización sistemática de código, los sistemas se integran a partir de componentes o sistemas existentes.
  Ventajas:
  - Reduce el costo de atender las necesidades cambiantes de los clientes o el mercado.
  - Se requiere menor cantidad de análisis y documentación.
  - Es más fácil obtener retroalimentación durante el desarrollo.
  - Entrega más rápida y despliegue de software de utilidad para el cliente.

  Desventajas:
  - Como definir las funciones que se deben realizar primero, para ello se debe realizar un estudio de factibilidad antes de comenzar el proyecto.
  - El proceso no es visible porque no es rentable producir documentos que reflejen todas las versiones del sistema y por qué se trabaja por partes.
  - Degrades del código.
- ***Ingeniería de software orientado a la reutilización:***
  Se basa en la reutilización sistemática de código, los sistemas se integran a partir de componentes o sistemas existentes. Es el enfoque estándar para la construcción de muchos tipos de sistemas. Posee distintas etapas:
  1. Análisis de requerimientos.
  2. Análisis de componentes.
  3. Modificación de requerimientos.
  4. Configuración del sistema con reutilización.
  5. Desarrollo e integración.

ESPECIFICACIONES DE SOFTWARE:

- ***Estudio de factibilidad:*** es técnicamente y financieramente factible construir el sistema.
- ***Requerimientos, obtención y análisis:*** requerimientos de los diferentes actores del sistema.
- ***Especificación de requerimientos:*** definición de los requisitos en detalle.
- ***Validación de requerimientos:*** comprobación de la validez de los requisitos.

VALIDACIÓN DEL SOFTWARE:

- ***Validar:*** si es lo que el cliente quería.
- ***Verificar:*** si el sistema funciona de acuerdo a los requerimientos.

PRUEBAS DEL SOFTWARE:

- ***Pruebas de componente:*** los componentes individuales se prueban de forma independiente. Estos componentes pueden ser funciones, objetos o agrupaciones de estas entidades.
- ***Pruebas del sistema:*** pruebas del sistema como un todo.
- ***Pruebas de aceptación:*** pruebas realizadas por el cliente para verificar que el sistema cumple con sus necesidades.

EVOLUCIÓN DEL SOFTWARE:

El software es flexible y puede cambiar, las circunstancias cambiantes del mercado y el cliente hacen que el software deba evolucionar y cambiar.

El cambio es inevitable en todos los proyectos de software, los costos del cambio incluyen la reelaboración, como los costos de implementación de nuevas funcionalidades.

Para evitar los costos de rehacer:

- ***Evitar el cambio:*** el proceso de software incluye actividades que pueden anticipar posibles cambios para evitar repetir el trabajo. Por ejemplo: un prototipo del sistema.
- ***Tolerar el cambio:*** el proceso está diseñado de modo que los cambios se pueden afrontar con un costo relativamente bajo.

PROTOTIPOS DE SOFTWARE:

Un prototipo es una versión inicial de un sistema que se utiliza para demostrar conceptos y probar opciones de diseño. Las ventajas del prototipado son:

- Mejor usabilidad del sistema.
- Mejor calidad del diseño.
- Mejor capacidad de mantenimiento.
- Menor esfuerzo de desarrollo.
- Aproximación más exacta a las necesidades reales de los usuarios.

ENTREGA INCREMENTAL:

El desarrollo y la entrega se desglosan en incrementos, con cada incremento se entrega parte de la funcionalidad requerida.

Se priorizan los requisitos de usuario, una vez que se inicia el desarrollo de un incremento, los requisitos quedan congelados. Las ventajas de la entrega incremental son:

- El sistema está disponible antes, con cada entrega se agrega funcionalidad al sistema.
- Las primeras entregas son como prototipos, útiles para esclarecer requisitos.
- Menor riesgo de fracaso del proyecto en general.
- Los servicios de mayor prioridad se entregan antes y reciben mayor cantidad de pruebas.

Desventajas:

- Puede ser difícil identificar las funcionalidades comunes que son necesarias para todos los incrementos.
- La especificación se desarrolla en conjunto con el software.

- **UNIDAD 3.**

INGENIERÍA DE REQUERIMIENTOS:

Es el proceso de establecimiento de los servicios que el cliente necesita de un sistema y las limitaciones con las que opera y se desarrolla.

Un **requerimiento** es una propiedad o restricción que un producto de software debe satisfacer.

A la hora de expresar un requerimiento, existen varias dificultades como:

- Dificultad de comunicación entre el equipo de desarrollo y los clientes.
- Cambio de los requerimientos.
- Imposibilidad de identificar la totalidad de los requerimientos de un producto.

TIPOS DE REQUERIMIENTOS:

- ***Requerimientos del usuario:*** el cliente nos cuenta que es lo que necesita, en lenguaje natural.
- ***Requerimientos del sistema:*** son descripciones detalladas de las funciones del sistema, los servicios y las limitaciones operativas.

Objetivos y requerimientos:

- ***Objetivos:*** es una declaración general de lo que el cliente quiere, como la facilidad del uso. Estos objetivos son útiles para los desarrolladores, ya que transmiten las intenciones de los usuarios del sistema.
- ***Requerimiento no funcional verificable:*** es una declaración mediante una cierta medida que se puede probar de forma objetiva.

REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES:

- ***Funcionales:*** define las funcionalidades que el producto debe cumplir.
- ***No funcionales:*** son limitaciones que puede tener el producto en la etapa de desarrollo u operación. Dependen de la arquitectura del sistema. Por ejemplo: uso de un IDE particular, lenguaje de programación, tiempo de respuesta, etc.
    - ***Requerimientos del producto:*** el producto entregado debe comportarse de una manera particular. Por ejemplo, velocidad de ejecución, fiabilidad, etc.
      Por ejemplo: el sistema estará disponible de lunes a viernes de 8:30 a 17:30.
        - Usabilidad.
        - Eficiencia.
        - Dependibilidad.
        - Seguridad.
    - ***Requerimientos organizacionales:*** son consecuencia de las políticas y procedimientos de la organización. Por ejemplo: estándares de procesos utilizados, requisitos de implementación, etc.
      Por ejemplo: los usuarios deben autenticarse usando su tarjeta de identidad.
        - Entorno.
        - Organizacionales.
        - Desarrollo.
    - ***Requerimientos externos:*** surgen de factores que son externos al sistema y su proceso de desarrollo. Por ejemplo, requisitos de interoperabilidad, requisitos legislativos, etc.
      Por ejemplo: el sistema deberá aplicar las disposiciones de privacidad del paciente según…
        - Regulatorios.
        - Éticos.
        - Legislativos.

DOCUMENTO DE REQUERIMIENTOS DE SOFTWARE:

Es la declaración oficial de lo que se requiere de los desarrolladores del sistema. Debe establecer lo que el sistema debe hacer no como es que debe hacerlo. Los usuarios de un documento de requerimientos son:

- ***Clientes del sistema:*** especifican los requerimientos y verifican que cumplen sus necesidades.

- ***Administradores:*** planifican una oferta por el sistema y el proceso de desarrollo del sistema.
- ***Ingenieros de sistemas:*** comprender que sistema debe desarrollarse.
- ***Testers:*** desarrollar las pruebas de validación para el sistema.
- ***Encargados de mantenimiento:*** comprender el sistema y las relaciones entre sus partes.

TIPOS DE ESPECIFICACIÓN DE REQUERIMIENTOS:

- ***Lenguaje natural:*** escritos en oraciones en lenguaje natural.
- ***Lenguaje natural estructurado:*** escritos en un formulario o planilla estándar.
- ***Lenguajes de descripción de diseño:*** utiliza un lenguaje de programación, pero más abstracto.
- ***Notaciones gráficas:*** modelos gráficos, como UML.
- ***Especificaciones matemáticas:*** se basa en conceptos matemáticos.

INGENIERÍA DE REQUERIMIENTOS:

Los procesos utilizados varían dependiendo de la aplicación, la organización, las personas, etc. Pero hay una serie de actividades genéricas:

- ***Obtención y análisis:*** involucra al personal técnico que trabaja con los clientes para averiguar el dominio de aplicación, los servicios que debe proporcionar y las limitaciones operativas del sistema. Existen varios problemas con la obtención y el análisis de requerimientos:
    - Los interesados expresan requisitos en sus propios términos y las diferentes partes pueden tener requisitos contradictorios.
    - Los requisitos cambian durante el proceso de análisis.
    - Factores organizativos y políticos pueden influir en los requisitos del sistema.
- ***Validación***
- ***Gestión***

DESCUBRIMIENTO DE REQUERIMIENTOS:

Es el proceso de recopilación de información sobre las necesidades del sistema mediante la interacción con los distintos actores. Para descubrir los requerimientos se suelen utilizar entrevistas de distintos tipos:

- ***Entrevistas cerradas:*** lista de preguntas predeterminadas.
- ***Entrevistas abiertas:*** varios temas se exploran con las partes interesadas.

Por lo general, se suele utilizar una mezcla de entrevista cerrada y abierta.

- ***Escenarios:*** son ejemplos reales de cómo se puede utilizar un sistema. Deberían incluir:
    - Descripción de la situación de partida.
    - Descripción del flujo normal de los acontecimientos.
    - Descripción de lo que puede salir mal.
    - Descripción de la situación cuando el escenario termina.

CASOS DE USO:

Muestran los posibles actores ya sean internos o externos que interactúan con el sistema. Un conjunto de casos de uso debe describir todas las posibles interacciones con el sistema.

ETNOGRAFÍA:

Es una **técnica de observación** que se usa para entender los procesos operacionales. Se observa cómo la gente hace las tareas, los estudios etnográficos relevan detalles críticos de procesos.

GESTIÓN DE REQUERIMIENTOS:

Es el proceso que permite realizar el seguimiento de los cambios en los requerimientos durante el proceso de ingeniería de requerimientos y de desarrollo del sistema.

Es necesario hacer un seguimiento de los requerimientos y sus vínculos para evaluar el impacto de los cambios.

- **UNIDAD 4**

DESARROLLO RÁPIDO DE SOFTWARE:

Los requisitos de las empresas cambian rápidamente y es imposible tener requerimientos de software estables. Ya que el software tiene que evolucionar rápido para reflejar los cambios en las necesidades del negocio.

MÉTODOS ÁGILES:

El objetivo de los métodos ágiles es reducir los gastos generales del proceso de software y ser capaces de responder a las necesidades cambiantes sin rehacer trabajo de manera excesiva. Hay ciertos problemas con los métodos ágiles:

- Difícil mantener el interés de los clientes involucrados.
- Los miembros del equipo pueden ser inadecuados para la intensa participación.
- Priorizar cambios puede ser difícil donde hay múltiples partes.
- Mantener simplicidad requiere trabajo.
- Los contratos pueden ser un problema.

DESARROLLO GUIADO POR PLAN Y DESARROLLO ÁGIL:

- ***Desarrollo guiado por plan:*** identifica etapas separadas en el proceso de software con salidas asociadas a esas etapas. Las salidas suelen usarse como base para la siguiente actividad. Conviene usar métodos guiados por plan cuando:
    - La especificación y el diseño son muy detallados antes de pasar a la implementación.
    - El sistema es grande y el equipo también.
    - Las organizaciones tradicionales tienen cultura basada en planes.
    - El sistema tiene que ser aprobado por un regulador externo.
- ***Desarrollo ágil:*** la especificación, diseño, implementación y pruebas están intercalados. Conviene usar métodos ágiles cuando:
    - Se pueden realizar entregas incrementales.
    - El equipo es pequeño y se puede comunicar de manera informal.
    - Los diseñadores y programadores tienen niveles más altos de capacitación.

PROGRAMACIÓN EXTREMA:

Es el más conocido y utilizado método ágil. Las nuevas versiones se pueden construir varias veces al día y los incrementos se entregan a los clientes cada 2 semanas, sólo si todas las pruebas son satisfactorias. Requiere distintos puntos:

- Participación del cliente.
- Entrega incremental.
- Personas no procesos.
- Adoptar el cambio.
- Refactorización.

Las solicitudes de los usuarios se expresan como escenarios o historias de usuario y el cliente elige las historias para su inclusión en el próximo incremento en base a sus prioridades y calendario.

Programación extrema sostiene que no vale la pena anticipar los cambios ya que estos no se pueden prever de forma fiable.

Las pruebas son fundamentales para XP. El software se comprueba después de cada cambio. En XP, se desarrollan las pruebas en primer lugar y los usuarios participan en el desarrollo de la prueba. Se utilizan las pruebas automatizadas cada que vez una nueva versión está construida.

Escribir pruebas antes del código aclara los requisitos para ser implementados. A la hora de escribir pruebas, se presentan algunas dificultades:

- Los programadores prefieren la programación a las pruebas y a veces se toman atajos al escribir pruebas.

- Algunas pruebas pueden ser muy difíciles de escribir de forma incremental.
- Es difícil juzgar la integridad de un conjunto de pruebas.

SCRUM:

Es un método ágil, su atención se centra en la gestión del desarrollo iterativo. Posee tres fases:

- ***Fase inicial:*** es una fase de anteproyecto, donde se establecen los objetivos generales y se diseña la arquitectura del software.
- ***Ciclos:*** cada ciclo desarrolla un incremento del sistema.
- ***Fase final:*** concluye el proyecto, completa documentación, manuales de usuario y evalúa las lecciones aprendidas del proyecto.

Los **sprints** son de longitud fija, normalmente 2-4 semanas. Lo que se desarrollará durante el sprint se define entre el equipo y el cliente. Al final el trabajo, se revisa y se presenta a las partes interesadas.

El ***"Scrum Master"*** organiza reuniones diarias, revisa la lista de trabajos por hacer, registra decisiones, mide el progreso y se comunica con los clientes.

Ventajas de Scrum:

- El producto se divide en fragmentos manejables y comprensibles.
- Los requisitos inestables son más manejables.
- Todo el equipo conoce todo el proceso.
- Los clientes ven la entrega y se obtiene retroalimentación en seguida.
- Aumenta la confianza entre clientes y desarrolladores.

USABILIDAD:

Es la facilidad con la que una persona puede utilizar una herramienta particular u otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto con **efectividad, eficiencia y satisfacción.**

El grado de usabilidad se mide a partir de distintos tipos de pruebas:

- ***Pruebas empíricas:*** no se basa en opiniones o sensaciones, si no en pruebas de usabilidad realizadas en laboratorios u observadas mediante trabajo de campo.
- ***Pruebas relativas:*** el resultado depende de las metas planteadas.

PRINCIPIOS DE LA USABILIDAD:

- ***Facilidad de aprendizaje:*** facilidad con la que nuevos usuarios desarrollan una interacción efectiva con el sistema.
- ***Facilidad de uso:*** facilidad con la que el usuario hace uso de la herramienta.
- ***Flexibilidad:*** variedad de posibilidades con las que el usuario y el sistema pueden intercambiar información.
- ***Robustez:*** nivel de apoyo al usuario que facilita el cumplimiento de sus objetivos.

DISEÑO CENTRADO EN EL USUARIO:

Reconoce las necesidades y los intereses de los usuarios. Se basa en las capacidades y limitaciones naturales de las personas y permite lograr una mayor facilidad de uso. Los beneficios del DCU son:

- Mayor productividad.
- Menos errores.
- Menor capacitación.
- Participación activa de los usuarios.
- Iteración en el diseño.

TÉCNICAS DEL DISEÑO CENTRADO EN EL USUARIO:

- ***Identificación de los usuarios:***
    - ***Usuarios primarios:*** personas que usarán el producto.

    - ***Usuarios secundarios:*** ocasionalmente usen el producto o lo consumen a través de intermediarios.
    - ***Usuarios terciarios:*** se ven afectados por el uso del producto.
- ***Prototipado:*** en esta fase los diseñadores desarrollan soluciones con distintos diseños para que sean evaluados por los usuarios.
- ***Evaluación:*** a medida que avanza el ciclo de diseño, los prototipos pueden ser producidos y luego probados por los usuarios. Estas evaluaciones revelan la satisfacción de los usuarios con el producto.
- ***Pruebas de usabilidad:*** requieren la participación de usuarios reales personas que concuerdan con el perfil de futuros usuarios. Emplea técnicas para recoger datos empíricos.

LAS 8 REGLAS DE SHNEIDERMAN

1. ***Consistencia:*** uso de iconos que sean familiares.
2. ***Atajos:*** con el constante uso de un producto o servicio, se demandan formas más rápidas para realizar tareas.
3. ***Retroalimentación:*** cada acción debe tener una retroalimentación legible y razonable.
4. ***Cierre de procesos:*** los usuarios deben saber cual fue el resultado de sus acciones.
5. ***Manejo de errores:*** ofrecer forma sencilla de corregir errores.
6. ***Deshacer operaciones:*** ofrecer formas sencillas de retroceder o revertir acciones.
7. ***Sensación de control:*** permitir que el usuario sea el que inicia las cosas.
8. ***Reducir la carga de memoria:*** la interfaz debe ser lo mas sencilla posible y con una jerarquía de información evidente.

UNIDAD 5.

MODELADO DE SISTEMAS:

Un **modelo** es una **abstracción** de un sistema o entidad del mundo real. Y una **abstracción** es una simplificación que solo incluye detalles relevantes. El modelado de sistemas se representa mediante un tipo de notación gráfica, por lo general, en **UML**.

Perspectivas del sistema:

- ***Perspectiva externa:*** el contexto del sistema. (**Diagrama de casos de uso**).
- ***Perspectiva interactiva:*** las interacciones entre el sistema y su entorno. (**Diagrama de secuencia**).
- ***Perspectiva estructural:*** la estructura de los datos del sistema. (**Diagrama de clases**).
- ***Perspectiva conductual:*** comportamiento dinámico del sistema y como responde a los eventos. (**Diagrama de estados**).

MODELOS DE CONTEXTO:

Sirven para ilustrar el contexto operativo del sistema, muestra lo que se encuentra fuera de los límites del sistema. Muestran el sistema y su relación con otros sistemas.

MODELOS DE CASOS DE USO:

Muestran los posibles actores ya sean internos o externos que interactúan con el sistema. Un conjunto de casos de uso debe describir todas las posibles interacciones con el sistema. Describe que hace el sistema, no como lo hace. Las relaciones entre casos de uso pueden ser:

- ***Generalización:*** un caso de uso se puede especializar en uno o mas casos de uso hijos.
- ***Inclusión:*** un caso de uso utiliza siempre a otro caso de uso.
- ***Extensión:*** un caso de uso tiene un comportamiento opcional, reflejado en otro caso de uso.

MODELOS DE SECUENCIA:

Se utilizan para modelar las interacciones entre los actores y los objetos dentro de un sistema. Un diagrama de secuencia muestra la secuencia de interacciones que tienen lugar durante un caso de uso particular.

MODELOS DE EVENTOS:

Muestran como el sistema responde a acontecimientos externos e internos. Cuando ocurre un evento, el sistema puede pasar de un estado a otro.

MODELOS DE CLASES:

Se utilizan para mostrar las clases de un sistema y las asociaciones entre estas clases. Posee distintos tipos de relaciones:

- ***Asociación:*** es el tipo de relación más común y representa una dependencia semántica. Es una simple línea continua que une las clases.
- ***Agregación:*** indica a un objeto y las partes que componen a ese objeto. Se representa con una línea que tiene un rombo en blanco.
- ***Composición:*** representa una relación jerárquica de una forma mas fuerte. Cuando el elemento contenedor desaparece, desaparecen los contenidos. Se representa con una línea que tiene un rombo en negro.
- ***Herencia:*** esta relación permite que una clase reciba los atributos y métodos de otra clase. Estos atributos y métodos recibidos se suman a los que tiene la clase por si misma.

- **UNIDAD 6.**

DISEÑO ARQUITECTÓNICO:

Representa el vinculo entre los procesos de especificación y diseño. Suelen llevarse a cabo en paralelo con las actividades de algunas especificaciones.

Modelos arquitectónicos:

- ***Modelo-Vista-Controlador:***
  - ***Controlador:*** actúa como intermediario entre el modelo y la vista. Gestiona el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno. Es responsable de recibir los eventos de entrada.
  - ***Modelo:*** contiene una representación de los datos que maneja el sistema y las funcionalidades del sistema. Es responsable de acceder a la capa de almacenamiento de datos, definir la funcionalidad del sistema y llevar un registro de las vistas y controladores del sistema.
  - ***Vista:*** compone la información que se envía al cliente y los mecanismos de interacción con este. Es responsable de recibir los datos del modelo y mostrarlos al usuario.
- ***Modelo repositorio:*** son datos compartidos en un repositorio o base de datos central y puede ser visitada por todos los subsistemas. Cada subsistema mantiene su propia base de datos y pasa datos explícitamente a otros subsistemas. Es más utilizado cuando se comparten grandes cantidades de datos.
- ***Modelo cliente-servidor:*** es un conjunto de servidores independientes que ofrecen servicios específicos y un conjunto de clientes que piden estos servicios.
- ***Modelo de capas:*** modela la interacción de subsistemas, organiza el sistema en un conjunto de capas y cada uno de los cuales provee un conjunto de servicios.
- ***Modelo de tubería:*** cada componente es discreto y realiza un tipo de transformación de datos. Los datos fluyen de un componente a otro para su procesamiento.

RESUMEN 2DO PARCIAL IS-1

- **UNIDAD 7.**

DISEÑO E IMPLEMENTACIÓN:

Es la etapa en el que se desarrolla un sistema de software ejecutable. El diseño de software es la fase en la que se identifica los componentes de software y sus relaciones sobre la base de requisitos del cliente y la implementación es el proceso de realización del diseño como un programa.

Es posible comprar sistemas que pueden ser adaptados y orientados a los deseos de los usuarios.

ETAPAS DEL PROCESO DE DOO:

Las etapas de procesos de diseño orientados a objetos dependen de la organización, las más comunes son:

1. ***Definir el contexto y los modos de uso del sistema:*** es esencial para decidir la manera de proporcionar la funcionalidad requerida del sistema y cómo estructurar el sistema. También permite establecer los límites del sistema.
2. ***Diseñar la arquitectura del sistema:*** se identifican los componentes principales que conforman el sistema y sus interacciones para luego poder organizar los componentes utilizando un patrón arquitectónico.
3. ***Identificar los principales objetos del sistema:*** se basa en la habilidad, experiencia y conocimiento del dominio de los diseñadores del sistema. Es un proceso iterativo, por lo que es poco probable hacerlo bien la primera vez.
4. ***Desarrollar modelos de diseño***.
5. ***Especificar las interfaces de objetos.***

PATRONES DE DISEÑO:

Es una descripción del problema y la esencia de su solución, debe ser lo suficientemente abstracto para ser reutilizado en diferentes entornos. Suelen hacer uso de la herencia y el polimorfismo.

- ***El patrón observador:*** separa la vista del estado del objeto, del objeto en sí. Se utiliza cuando se necesitan varias vistas de estado. Cuando el estado del objeto cambia, todas las vistas son automáticamente notificadas y actualizadas para reflejar el cambio.

IMPLEMENTACIÓN DEL SOFTWARE:

Es la etapa de la ingeniería de software en la que se crea la versión ejecutable. Los aspectos importantes en la implementación son:

- ***Reutilización:*** los costos y la presión de las entregas causaron que la no reutilización fuese cada vez más inviable, especialmente para los sistemas comerciales y basados en Internet. Existen distintos niveles de reutilización:
  - ***Nivel de abstracción:*** se utiliza el conocimiento de las abstracciones exitosas en el diseño del software.
  - ***Nivel de objeto:*** se vuelve a utilizar directamente los objetos de una biblioteca en lugar de escribir el código manualmente.
  - ***Nivel de los componentes:*** son colecciones de objetos y clases de objetos que se reutilizan en los sistemas de aplicación.
  - ***Nivel de sistema:*** se vuelve a utilizar los sistemas de aplicación enteros.
- ***Administración de la configuración:*** el objetivo es apoyar el proceso de integración de sistemas para que todos los desarrolladores puedan acceder al código y a la documentación del proyecto de una manera controlada, averiguar que cambios se han hecho y compilar y enlazar componentes para crear un sistema. Existen distintas actividades de gestión de la configuración:
  - ***Gestión de versiones:*** sirve para realizar un seguimiento de las diferentes versiones de los componentes de software.
  - ***Integración de sistemas:*** proporciona apoyo a los desarrolladores a definir qué versiones de los componentes se utilizan para crear cada versión de un sistema.
  - ***Seguimiento de problemas:*** brinda apoyo para que los usuarios reporten errores y problemas.
- ***Desarrollo huésped – objetivo:*** la mayoría del software está desarrollado en un equipo, pero se ejecuta en una máquina diferente. Se podría decir que tenemos una plataforma de desarrollo y una plataforma de ejecución.

HERRAMIENTAS DE DESARROLLO:

Por lo general se utiliza un compilador integrado y un sistema de edición dirigido a la sintaxis que permite crear, editar y compilar código.

IDE:

Las herramientas de desarrollo de software se agrupan para crear un entorno de desarrollo integrado. Son creados para apoyar el desarrollo de un lenguaje especifico de programación.

DESARROLLO DE CÓDIGO ABIERTO:

Es un enfoque para el desarrollo de software en el que se publica el código fuente de un sistema de software y se invita a los voluntarios a participar en el proceso de desarrollo.

- ## UNIDAD 8.

METAS DE LA PRUEBA DE SOFTWARE:

1. Demostrar al desarrollador y al cliente que el software cumpla con sus requisitos.
2. Descubrir situaciones en las que el comportamiento del software es incorrecto, indeseable o no se ajusta a su especificación.

VALIDACIÓN Y PRUEBAS DE DEFECTOS:

- ***Pruebas de validación:*** se espera que el sistema realice correctamente el uso de un determinado conjunto de casos de prueba que reflejen el uso esperado del sistema.
- ***Pruebas de defectos:*** están diseñados para exponer los defectos. No tienen por qué reflejar cómo se utiliza normalmente el sistema.

VERIFICACIÓN VS VALIDACIÓN:

- ***Verificación:*** si el software cumple con los requerimientos funcionales y no funcionales establecidos.
- ***Validación:*** el software debe hacer lo que el usuario realmente necesita.

INSPECCIONES DE SOFTWARE:

Se enfocan en el código fuente de un sistema con el objetivo de descubrir anomalías y defectos. Puede ser utilizado antes de la implementación y se pueden aplicar a cualquier representación del sistema. Ventajas de las inspecciones son:

- Es un proceso estático, por lo que no hay interacción entre los errores.
- Las versiones incompletas de un sistema pueden ser inspeccionadas fácilmente.
- Permite refactorizar mejorando la calidad del programa.

ETAPAS DE PRUEBA:

1. ***Pruebas de desarrollo:*** el sistema se prueba durante el desarrollo para descubrir errores y defectos.
    a. ***Pruebas de unidad:*** se ponen a prueba los componentes individuales de forma aislada.
    b. ***Pruebas de componentes:*** se integran varias unidades individuales para crear componentes compuestos.
    c. ***Pruebas del sistema:*** el sistema se pone a prueba en su conjunto.
2. ***Pruebas de versión:*** un equipo testea una versión completa del sistema antes de que sea puesto en operación.
3. ***Pruebas de usuario:*** los usuarios potenciales del sistema, prueban el sistema en su propio entorno.
    a. ***Pruebas alfa:*** los usuarios del software trabajan con el equipo de desarrollo para poner a prueba el software en el sitio del desarrollador.
    b. ***Pruebas beta:*** una versión del software está disponible para los usuarios y les permite experimentar y plantear a los desarrolladores los problemas que descubren.
    c. ***Pruebas de aceptación:*** los clientes prueban el sistema para decidir si está listo para ser aceptado y utilizado en el entorno del cliente. Etapas de la prueba de aceptación:

PRUEBAS AUTOMATIZADAS:

Siempre que sea posible las pruebas de unidad deben ser automatizadas de forma que se ejecuten y comprueben sin necesidad de intervención manual. Los componentes de las pruebas automatizadas son:

- ***Configuración:*** se inicializa el sistema con el caso de prueba.
- ***Llamada:*** se llama al objeto o método para ensayar.
- ***Declaración:*** se compara el resultado de la llamada con el resultado esperado.

EFECTIVIDAD DE PRUEBA DE UNIDAD:

Los casos de prueba deben demostrar que cuando se usa como esperaba, el componente que se está probando hace lo que se supone que debe hacer.

Estrategias de pruebas:

- ***Prueba de partición:*** se identifica grupos de entrada que tienen características comunes y deben ser procesadas de la misma manera.
- ***Prueba basada en lineamientos:*** se utiliza la experiencia previa de los tipos de errores que los programadores suelen hacer cuando se desarrolla los componentes.

PRUEBA DE INTERFACE:

El objetivo es detectar fallas debido a errores de interfaz o suposiciones invalidas sobre interfaces. Existen distintos tipos de interfaces:

- ***Interfaces de parámetro:*** los datos pasan de un componente a otro.
- ***Interfaces de memoria compartida:*** un bloque de memoria se comparte entre componentes.
- ***Interfaces de procedimiento:*** un componente encapsula un conjunto de procedimientos.
- ***Interfaces que pasan mensajes:*** un componente solicita un servicio de otro componente.

PRUEBAS DE CASOS DE USO:

Los casos de uso desarrollados para identificar las interacciones del sistema se pueden utilizar como base para las pruebas del sistema.

POLÍTICAS DE PRUEBAS:

Las pruebas exhaustivas del sistema son imposibles, por lo que se establece una política para definir la cobertura. En general se debe probar:

- Funciones del sistema accesibles a través de menús.
- Combinaciones de funciones accesibles a través del mismo menú.
- Funciones donde haya entrada del usuario.

DESARROLLO BASADO EN PRUEBAS:

Es un enfoque en el que se entrelazan el desarrollo del código y el de las pruebas. Las pruebas se escriben antes que el código y no se pasa al siguiente incremento hasta que el código pase la prueba. Es usado en XP.

BENEFICIOS DEL DESARROLLO BASADO EN PRUEBA:

- Cada segmento de código que se escribe tiene por lo menos una prueba asociada.
- Las pruebas de regresión se desarrollan progresivamente a medida que se desarrolla un programa.
- Cuando falla una prueba, es obvio dónde está el problema.
- Las pruebas son una forma de documentación.

UNIDAD 9.

EVOLUCIÓN Y/O MANTENIMIENTO:

El desarrollo del software no se detiene cuando un sistema se entrega, continúa a lo largo de toda su vida. Luego de distribuir un sistema, inevitablemente debe modificarse con el fin de mantenerlo útil.

PROCESO DE EVOLUCIÓN DEL SOFTWARE:

Los procesos de identificación de cambios y evolución del sistema son cíclicos y continúan a lo largo de la vida de un sistema.

LEYES DE LEHMAN:

1. ***Cambio continuo.***
2. ***Complejidad incremental.***
3. ***Evolución prolongada del programa o autorregulación.***
4. ***Conservación de la estabilidad organizacional.***
5. ***Conservación de la familiaridad.***
6. ***Crecimiento continuo.***
7. ***Reducción de la calidad.***
8. ***Realimentación del sistema.***

MANTENIMIENTO DEL SOFTWARE:

Es el proceso general de cambiar un sistema después de que este se entregó. Usualmente se aplica a software personalizado, en el que grupos de desarrollo separados intervienen antes y después de la entrega. Existen tres motivos:

- Corregir un mal funcionamiento.
- Mejorar atributos de calidad.
- Adaptarlo a los cambios del entorno.

El mantenimiento del software es más costoso que el desarrollo. Resulta mas costoso agregar funcionalidad después de que un sistema está en operación, que implementar la misma funcionalidad durante el desarrollo.

PREDICCIÓN DEL MANTENIMIENTO DE SOFTWARE:

Predecir qué cambios pueden proponerse al sistema y qué partes serán más difíciles de mantener. Se debe valorar:

- ***El número y la complejidad de las interfaces del sistema:*** cuánto mas grande es el número de interfaces es más probable que se requieran cambios al agregar nuevos requerimientos.
- ***El número de requerimientos de sistemas inherentemente inestables:*** los requerimientos que reflejan políticas y procedimientos de la organización son más inestables que los que se basan en características de un dominio estable.
- ***Procesos empresariales donde se usa el sistema:*** cuantos más procesos use un sistema, habrá más demandas de cambio.

TIPOS DE MANTENIMIENTO:

- ***Preventivo:*** consiste en mejorar las propiedades del software sin alterar sus especificaciones funcionales. Ejemplos:
  - ***Refactorización del código:*** para mejorar su legibilidad.
  - ***Modificar el software:*** identificando componentes reusables.
  - ***Comprobación:*** de la validez de los datos de entrada.
- ***Correctivo:*** consiste en localizar, corregir y eliminar los defectos del software. Ejemplos:
  - ***De procesamiento:*** salidas incorrectas.
  - ***De rendimiento:*** tiempo de respuesta alto en una búsqueda.
  - ***De programación:*** inconsistencias en el diseño de un programa.
  - ***De documentación:*** inconsistencias entre la funcionalidad de un programa y el manual de usuario.
- ***Perfectivo:*** satisfacer cambios en requisitos funcionales o no funcionales. Ejemplos:
  - ***Agregar:*** una funcionalidad no contemplada.
  - ***Mejorar:*** usabilidad del sistema o la velocidad de respuesta optimizando algoritmos.
- ***Adaptativo:*** modificar el software para adaptarlo a un nuevo entorno operativo. Ejemplos:
  - ***Cambios:*** del sistema operativo, del gestor de bases de datos, de la configuración de hardware, cambios en la organización o legislativos.

ACTIVIDADES DEL MANTENIMIENTO DE SOFTWARE:

- ***Comprensión del software y de los cambios a realizar:*** es necesario el conocimiento a fondo de la funcionalidad.
- ***Modificación del software:*** crear y modificar las estructuras de datos, la lógica de procesos, las interfaces y la documentación. Se deben evitar los efectos provocados por sus cambios.
- ***Realización de pruebas:*** realizar pruebas selectivas que nos aseguren la corrección del software.

• UNIDAD 10:

PROYECTO:

Es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único.

- Tiene un principio y un final definidos.
- El final se alcanza cuando se logran los objetivos del proyecto, cuando se termina el proyecto por que sus objetivos no se cumplirán o no pueden ser cumplidos o cuando ya no existe la necesidad que dio origen al proyecto.
- Implica la ejecución de un conjunto de actividades coordinadas y controladas para lograr un objetivo conforme a requisitos específicos.

GESTIÓN DE PROYECTOS DE SOFTWARE:

Son aquellas actividades que aseguran que el software será entregado en el tiempo justo y estará de acuerdo con los requerimientos del cliente. Es necesario por que el desarrollo estará siempre sujeto a las limitaciones de costos y tiempos de la empresa. Incluye los siguientes puntos:

- Identificar los requisitos.
- Establecer objetivos claros y posibles de realizar.
- Equilibrar demandas concurrentes de calidad, alcance, tiempo y costos.
- Adaptar las especificaciones, los planes y el enfoque a las diversas inquietudes y expectativas de los interesados.

ACTIVIDADES DE LOS ADMINISTRADORES:

- ***Planeación del proyecto:*** planeación, estimación y calendarización, así como la asignación de tareas.
- ***Informes:*** responsables de informar el avance del proyecto a los clientes y administradores de la compañía que desarrolla el software.
- ***Gestión del riesgo:*** valorar los riesgos que pueden afectar el proyecto y monitorizar dichos riesgos.
- ***Gestión del personal:*** responsables de administrar el equipo de personas.
- ***Redactar propuestas:*** describe los objetivos del proyecto y cómo se realizará.

RIESGOS:

Un riesgo es la posibilidad de que una circunstancia adversa se pueda presentar. Existen varios tipos:

- ***Riesgos del proyecto:*** alteran el calendario o los recursos del proyecto.
- ***Riesgos del producto:*** afectan la calidad o el rendimiento del software a desarrollar.
- ***Riesgos empresariales:*** afectan a la organización que desarrolla o adquiere el software.

GESTIÓN DEL RIESGO:

- ***Identificación del riesgo:*** identificar posibles riesgos para el proyecto, el producto y la empresa.
    - Riesgos tecnológicos: derivan de las tecnologías de software o hardware usadas para desarrollar el sistema.
    - Riesgos personales: se asocian con las personas en el equipo de desarrollo.
    - Riesgos organizacionales: se derivan del entorno organizacional donde se desarrolla el software.

    - Riesgos de herramientas: resulta de las herramientas y otro software de soporte que se usa para desarrollar el sistema.
    - Riesgos de requerimientos: procede de cambios a los requerimientos del cliente.
    - Riesgos de estimación: surgen de estimaciones administrativas de los recursos requeridos para construir el sistema.
- ***Análisis de riesgos:*** valorar la probabilidad y las consecuencias de dichos riesgos. Para cada riesgo identificado estimar la probabilidad de ocurrencia y gravedad en caso de ocurrir.
- ***Planeación del riesgo:*** elaborar planes para enfrentar el riesgo.
    - Estrategias de evitación: reducir la probabilidad de que surja el riesgo.
    - Estrategias de minimización: reducir el efecto del riesgo.
    - Planes de contingencia: hacer frente y tener una estrategia para ello.
- ***Monitorización del riesgo:*** valorar regularmente el riesgo y los planes para atenuarlo.

GESTIÓN DE PERSONAL:

Las personas son el principal capital de la empresa. Una incorrecta gestión del personal garantiza la falla del proyecto. Se debe asignar responsabilidades acordes a las habilidades de cada persona.

Un rol importante del administrador es motivar al personal, organizando el trabajo y el entorno laboral para alentar a las personas a trabajar mejor.

GRUPO COHESIVO:

El grupo puede establecer sus propios estándares de calidad. Los individuos aprenden de los demás y se apoyan mutuamente. El conocimiento se comparte para que pueda mantenerse la continuidad si sale un miembro del grupo.

Se alientan la refactorización y el mejoramiento continuo. Los miembros del grupo trabajan de manera colectiva para entregar resultados de alta calidad y corregir problemas.

CALENDARIZACIÓN DE PROYECTOS:

Es el proceso de decidir cómo se organizará el trabajo en un proyecto. Se definen las tareas, cuándo y cómo se ejecutan, el tiempo calendario para cada tarea y los recursos necesarios.

- **UNIDAD 11:**

CALIDAD:

Significa que un producto debe cumplir con sus especificaciones. Algunos requerimientos de calidad son difíciles de especificar (eficiencia, mantenibilidad, reusabilidad, etc.).

MANEJO DE LA CALIDAD DEL SOFTWARE:

Se refiere a lograr un nivel de calidad requerido en el producto de software. Involucra la definición de estándares de calidad apropiados y procedimientos que permiten asegurar que estos se cumplan.

ACTIVIDADES DE MANEJO DE CALIDAD:

- ***Asegurar la calidad:*** establecer procedimientos organizacionales y estándares para la calidad.
- ***Planear la calidad:*** seleccionar procedimientos aplicables y estándares para un proyecto en particular y modificar estos como sean requeridos.
- ***Control de calidad:*** garantizar que procedimientos y estándares son seguidos por el equipo de desarrollo.

CALIDAD BASADA EN PROCESOS:

Es la relación directa entre procesos y productos. Definir procesos de estándares que indiquen como llevar a cabo las revisiones, la administración de la configuración, etc., se debe monitorear el proceso de desarrollo para asegurar que se están siguiendo los estándares.

ESTÁNDARES DE SOFTWARE:

Definen características que todos los componentes deberán tener. Son clave para un efectivo manejo de calidad. Son importantes porque:

- Reúne las mejores prácticas.
- Evita la repetición de errores.
- Proporciona un marco para el análisis de la calidad.
- Proporciona continuidad.

Los problemas con los estándares son:

- No son vistos como relevantes por los ingenieros de software.
- Involucra actividades burocráticas.
- Se requieren actividades manuales para mantener los estándares.

REVISIÓN:

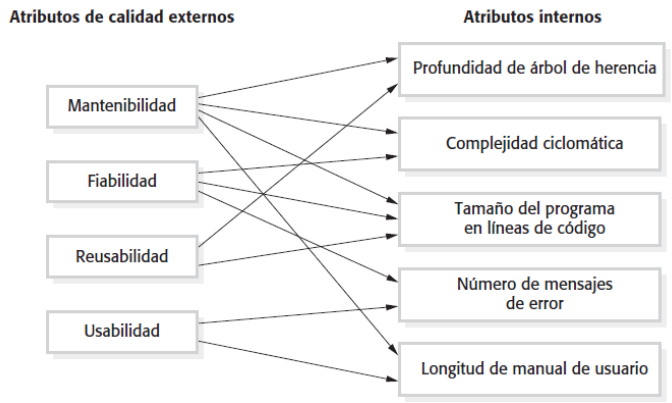
Es el principal método de validación de la calidad de un proceso o de un producto. Un grupo debe examinar parte o toda si documentación para buscar problemas potenciales. Hay diferentes tipos de revisiones con diferentes objetivos.

- Inspecciones para remover defectos (producto).
- Revisiones para estimación de progresos (procesos y productos).
- Revisiones de calidad (estándares y producto).
    - Un grupo de personas examina cada parte o todo un sistema de software y su documentación asociada.
    - El objetivo es descubrir defectos en el sistema e inconsistencias.
    - El equipo debe ser pequeño y las revisiones deben ser cortas.

MÉTRICAS DE LA CALIDAD DEL PRODUCTO:

Es una forma de predicción de la calidad del producto. La mayoría de las métricas de calidad se relacionan con la medición del acoplamiento o la complejidad del diseño.

ATRIBUTOS INTERNOS Y EXTERNOS:



MÉTRICAS ESTÁTICAS DE PRODUCTOS DE SOFTWARE:

- ***Fan-in:*** es una medida del número de funciones o métodos que llaman a otra función o método.
- ***Fan-out:*** es el número de funciones a las que llama la función.
- ***Longitud de código:*** cuánto mas grande es el tamaño del código de un componente, más probable será que el componente sea complejo y proclive a errores.
- ***Complejidad ciclomática:*** es una medida de la complejidad del control de un programa.
- ***Longitud de identificadores:*** cuanto mas largos sean los identificadores, es más probable que sean significativos y, por ende, más comprensible será el programa.
- ***Profundidad de anidado condicional:*** los enunciados if profundamente anidados son difíciles de entender y proclives a errores.

- ***Índice Fog:*** es la medida de la longitud promedio de las palabras y oraciones en los documentos. Cuánto mas alto sea el valor, más difícil será entender el documento.

MÉTRICAS ORIENTADAS:

- ***Métricas ponderadas por clase:*** es el número de métodos en cada clase, ponderados por la complejidad de cada método. Cuánto mas grande sea el valor, más compleja será la clase.
- ***Profundidad de árbol de herencia:*** representa el número de niveles discretos en el árbol de herencia en que las subclases heredan atributos y operaciones. Cuánto mas profundo sea el árbol, mas complejo será el diseño.
- ***Número de hijos:*** es el número de subclases inmediatas de una clase.
- ***Acoplamiento entre clases:*** las clases están acopladas cuando los métodos en una clase usan los métodos o variables de instancia definidos en una clase diferente. Un valor más alto, significa que las clases son estrechamente dependientes.
- ***Respuesta por clase:*** es el número de métodos que potencialmente podrían ejecutarse en respuesta a un mensaje recibido por un objeto de dicha clase.
- ***Falta de cohesión en métodos:*** se calcula al considerar pares de métodos en una clase. Es la diferencia entre el numero de pares de método sin compartir atributos y el numero de pares de método con atributos compartidos.