Ingeniería de Software

Clase 4: Kanban

(20/03/2025)

Kanban

Metodología que optimiza el proceso de desarrollo de software por un flujo continuo de trabajo.
Kanban es un ciclo cerrado y todo lo que se compromete uno es lo que se tiene que entregar al final del sprint.

- Desarrollo incremental, dividiendo el trabajo en partes
  -Backlog | Stories | To do | In progress | To verify | Done

Básicamente, dividir el trabajo de forma visual (con notas)

- Limitante principal es limitar el trabajo en progreso en cada etapa.
- Por columna poner ciertas restricciones para tener trazabilidad

Cuando el equipo es nuevo, se avanza en un trabajo simultáneamente, es decir, me excedo de trabajo en un área.

- **Tarjeta de señal:** Unidad de trabajo, la cual se mueve a traves del flujo de la organización solo cuando existe la capacidad de tomar la tarea e el sgte paso del proceso.
- Puede perder la visibilidad del objetivo final

Principios de Kanban

1. Visualizar el trabajo
   Todos pueden visualizar el estado de cada tarea
   Promueve la transpariencia
2. Limita el trabajo en progreso (WIP)
3. Gestionar el flujo
   Observa y optimiza el trabajo a traves del progreso
   Identificacion del bloqueo
4. Visualizacion de Politicas
   Yo no puedo moverme a una columna sin que este
   _todos los commits hechos, criterios de aceptacion
5. Colaboracion, evolucion y experimentacion
   Colaboracion del equipo para mejorar el proceso
   Evaluacion de resultados
   Adaptacion del proceso

Requerimientos funcionales y no funcionales

Existen dos requerimientos:

Usuario (funcionales)

Requerimientos: Descripciones de los servicios del sistema en funcion de las necesidades del cliente

Sistema (No funcionales)

Limitaciones: Netamente asociados a un sistema o dominio

La documentación es importante en cuanto a requerimientos. Forman un contrato de trabajo de un producto

Requerimientos Funcionales (FR)

El producto debe ofrecer un servicio para satisfacer necesidades

- Principales Lectores:
  Gerentes, usuarios finales del sistema, ingenieros o analistas funcionales, contratistas (financieros).
- Requerimientos debe estar escrito a prueba de weones
- Puede incluir diagramas de servicio
- Al ser de alto nivel, pueden ser muy ambiguos
  EJ) Un usuario debe ser capaz de buscar las listas de citas de todas las clinicas
  MUY GENERAL (especificar los requerimientos)

Para escribir un FR se ocupa el sgte formato:

"El ... (tipo de usuario / sistema / componente del sistema) NECESIDADES
Verbo en futuro
Descripcion"

- Las necesidades pueden venir de un cliente o usuario

EJ) "El administrador podrá crear usuarios de tipo "Read Only" para que puedan acceder al sistema"

En desarrollo Ágil (US)

los requerimientos funcionales son denominados Historias de Usuario

- Estas historias de usuario tambo conviven en una línea temporal
  EJ) Requerimiento para etapa 1, etapa 2, etapa 3...
- Forman parte de un Product Roadmap

Son INVEST
Independent
Negotiable
Valuable
Estimable
Small
Testable

Formato de escritura

Como	**Rol**
Quiero (necesito)	Funcionalidad
Para	Beneficio
Supuesto (opcional depende del contexto)	Relacion

Criterios de Aceptación

- Criterio 1
- Criterio 2
- Criterio 3

Requerimientos NO Funcionales (NFR)

Limitaciones a los servicios o funciones que ofrece el sistema o producto

- Al ser una capa un poco más "profunda", al ser mas restrictivo puedo evitar gastos innecesarios
- Se aplica al sistema en conjunto o bien a servicios individuales
- Definen las propiedades del sistema o producto, tiempo de respuesta y almacenamiento
-

EJ) Sistema operativo, version de navegador / libreria, lenguaje de programacion, etc
la app funciona en ios y para androiod (versiones especificas)
Requerimientos mas restrictivos

Clasificaciones

Requerimientos del Producto

- Velocidad de ejecucion+
- Fiabilidad de datos
- % de error aceptado (casi 0%)

Requerimientos Organizacionales

- Politicas y procedimientos organizacionales
- Estandares
- Requisitos de implementacion

Requerimientos Externos

Externos al sistema

- Proceso de desarrollo
- Interoperabilidad
- Legislativo (si hay conflicto de contrato)

Métricas

```
Propiedad
Velocidad
Tamaño
Facilidad de uso
Reliability               Tiempo medio de fallo / Disponibilidad
Confiabilidad
Portabilidad
```

Hay apps que murieron por ser lentas

Clase 5: Requerimientos

(24/03/2025)

Requerimientos de Dominio

Derivan del dominio de la aplicacion del sistema o producto.

El entorno es un dominio (puede ser dominio de negocio)
En el dominio tambien hay requerimientos, por lo tanto restricciones

EJ) Sistema de control de trenes tiene en cuenta características de frenado en distintas condiciones climaticas.
EJ 2) Sistema de control tiene caracteristicas de la pista, vientos, condiciones climaticas extremas.

> Van asociados al contexto del negocio donde esta mi producto
> Tiende a restringir mas que dar aperturas

Muchas veces si los requerimientos de dominio no estan satisfechos, el sistema o producto queda inutil.

Documentación de Requerimientos

Es un documento oficial que los desarrolles de software implementan.

> Incluyen requerimientos funcionales, no funcionales y de dominio

- Relevante cuando el desarrollo de software se externaliza
- En frameworks agiles, los documentos de requerimiento pasan a ser las historias de usuario

Porqué ocurre esto?

- Producto y sus funcionalidades pueden cambiar en el tiempo
- Backlog vivo y versionamiento de roadmap de producto.

Debe incluir:

- Requerimientos de los clientes (audiencias)
- Detalle para desarrollo de las funcionalidades
- Información relevante del sistema

Usuarios de un documento de requerimientos:

Clientes del sistema
Especifican los requerimientos y los leen para comprobar que cubren sus necesidades

Administradores
Usan doc de req. para planear una cotizacion para el sistema y el proceso de desarrollo del sist.

Ingenieros del sistema
Ven los req. para ver que se desarrolla

Ingenieros de prueba del sistema
Ven los req. para desarrollar pruebas de validación para el sist.

Ingenieros de mantenimiento del sistema
Para comprender

Contenido

1. Prefacio
2. Introduccion
3. Glosario
   Diccionario de terminos abordados
4. Requerimientos de usuario
5. Arquitectura
6. Especificación de Requerimientos
7. Modelos del Sistema
8. Evolución
9. Apéndices (info adicional de config)
10. Índice

Levantamiento de Requerimientos

Muchas veces no se sabe lo que se requiere.
Muchas veces son necesidades y decisiones no basadas en numeros, sino de "guata"
Muchas veces los stakeholders no estan alineados respecto a la vision estrategica.

Ciclo exotico:
1. Descubrimiento de requerimientos
Interacción con las partes interesadas y/o stakeholders para descubrir las necesidades (Req funcionales, no funcionales y dominio)

2. Clasificación y organizacion de requerimientos
Se agrupan y ordenan en grupos coherentes

3. Priorización y negociacion de requerimientos
Se da prioridad y se abordan los conflictos

4. Especificación de requerimientos
Se documentan y se van generando versiones.
Muchas veces se genera una nueva iteración.

Técnicas de Levantamiento de requerimientos

**Entrevistas

- Formales o informales

- Usuarios claves o stakeholders
  **Levantamiento del proceso de negocio
- Como funciona hoy y como puede ser mas eficiente
- Procesos a alto nivel y de a poco va abordando más detalle (dependiendo del problema)
- Identificar usuarios claves para cada una de las interacciones / etapas

> Si voy a hablar con una persona que no esta al 100% con el proyecto, perdida de tiempo

Workshops
Encuestas de Satisfacción
User Experience / Se basa en la competencia
Prototipado
Para temas de agilidad.

Preguntas para criterios de aceptacion

- ¿Es pequeño?
- Falta el supuesto
- Falta el contexto
- Cada criterio debiera ser su propia historia
- Como funciona ahora -> Como quiero que funcione (Cambio de alcance)

Clase 6: Roadmap y Mapa de Historias de Usuario

(27/03/25)

RoadMap

Mapa de las principales funcionalidades de un producto.
Intención de lo que se desea desarrollar

- Siempre presente la *visión y objetivos*
  Pueden ser a nivel organizacional como también de lo que se desea lograr con el producto
- Hitos visibles / etapas
  Donde se desea obtener las ganancias y/o entrega de valor.
- NO es una gantt
- Usado para establecer metas claras con cada hito y sujeto a un presupuesto.
- Presentados/creados en una ceremonia llamada *Inception*
  !! Concepto clave de Scrum
- Agile Inception
  Es el Kick Off -> Se comparte, concreta y se da a conocer la vision del nuevo producto.
  Lore del nuevo producto en base a la necesidad

Visión del producto

[Nombre de Producto] atiende a [Cliente/Usuario Obj]
quienes buscan [Necesidades del cliente/Usuario Obj] para [Beneficio del producto].

Cuando se presenta nuestra visión, que la persona que la pruebe va a querer más y más.
Lo importante es que me lo compre.

Especificar diferenciación.

Mapa de Impacto

Meta ¿Para qué?
Objetivo debe estar alineado con la estrategia del negocio
Específico y medible.

- Objetivos *SMART*

Personas ¿Quién?
Entender quiénes tienen el poder de impactar el éxito del objetivo y cómo su comportamiento puede influir en el resultado.

Impacto ¿Cómo?
Definir los cambios en el comportamiento que esperamos de los actores identificados, para que contribuyan al logro del objetivo.

Entregables (Features) ¿Qué y Cuántos?
Identificar las funcionalides a implementar para lograr los impactos deseados.

SMART

Cultura Ágil

E *S* pecíficos
M edibles
A lcanzables
R ealistas
a *T* iempo

User Story Mapping

Visión de historias de usuario - Flexible

Conformado por las historias de usuario abordadas en cada etapa del roadmap
Es decir: Cada etapa de roadmap deberia tener un User Story Mapping

Para elaborar:

- Línea de tiempo de izquierda a derecha
- Priorización (arriba hacia abajo)
- Desglose de historias de cada una de las épicas
- Permite visualización de *MVP*
- Visualización e interacción de Roles

Diferencia entre (1) Product Owner, (2) Product Manager, (3) Project Manager

1. Dentro del equipo de Scrum
   Vela por la entrega de valor dentro del sprint
2. También es un Product Owner
   Tiene vision mas estrategica del producto
   Vela por todo, visión completa
3. Vela por los hitos en fecha de gantt y presupuesto

Elaboración de User Story Mapping

1. Identifico el usuario, Rol y Objetivo Enfocado en la necesidad
2. Definición del Journey del Usuario *(Customer Journey Map)*
   Sugerencias del usuario (carita triste, carita feliz)
3. Desglose de User Stories
4. Priorización vertical.
   Las más importantes arriba
5. Añadir detalles, subtareas, dependencias
6. Identificar releases/versiones
7. Chantar la actu (no recuerdo)

Épicas: Se pueden dividir en pequeñas tareas
Funcionalidades grandes.

Clase 7

31/03/25

Priorización y Estimación Agile

Resumen Clase pasada

- RoadMap
- User Story Mapping

Priorización

Nace del RoadMap y User Story Mapping

1. Backlog de Producto
2. *Sprint Planning:* Primera ceremonia a nivel de Scrum.
   Mientras más grande sea el equipo, más demora toma Sprint Planning.
3. Backlog de Sprint

- de 1 a 4 semanas
- entre medio hay reunion diaria de scrum

EJ) Definiciones que no quedaron muy claras, responsabilidad del Product Owner
Si hay una traba a nivel de framework, Scrum Master

Product Backlog y Planificación

Lista de funcionalidades a desarrollar para el producto.

Se sacan las dudas y se hace el tasking.

Reglas y acuerdos de equipo

- **DOR** Definition of Ready
- **DOD** Definition of Done

Planificación y Capacidad del Equipo

Radica en la # de historias que puedan ser abordadas/desarrolladas durante el Sprint.

El equipo mide la capacidad de desarrollo para cada persona

EJ) Tengo 10 dias de Sprint (desarrollo) en mi primer sprint
Pero necesito 1 día para ceremonia
1 dia entero para ceremonias (9 dias de trabajo)

Como nosotros no sabemos la capacidad del equipo. Cada dia equivale a 1 punto.
Cada integrante va a tener 9 puntos como capacidad para Sprint

De esos 9 días, se descuentan x días porque estamos en otros ramos.
Teniendo claridad de cómo está el equipo puedo dar un estimado para ver un problema.

Gestión del Sprint Backlog

Lo que se tiene que gestionar es el backlog del producto y como vamos abordando las historias comprometidas en el Sprint Backlog.

- Las Historias no se asignan, sino que cada miembro toma una historia de la pila. Lo más autónoma posible.
- Las tareas se van actualizando a medida que se avanza en el sprint.
  - Se integran conceptos de bloque, riesgos y apoyo.
  - Los puntos anteriores son declarados en la ceremonia de Daily.

Hay equipos que inflan el pesaje de la historia. Dictan que hace más, pero a la entrega de valor da un 10%. Pésima práctica.

- No se modifica el objetivo del sprint.

- En caso de haber cualquier cambio, debe ser acordado con el equipo en conjunto con el Product Owner.

Para Historias:

> Las historias deben ser independientes y pequeñas para no depender de otras historias y atrasarme.

Estimación Agile

```
Esfuerzo VS Duración
Valor VS Esfuerzo
```

Estimación realizada por el equipo de desarrollo en la ceremonia de Sprint Planning.
Existen estándares de cuánto debe durar como máximo una planificación y estimación.

Estimación por Tamaño Camiseta

XS, S, M, L, XL, etc..

- El pivote tiende a ser XS o S
- El equipo establece los criterios para la historia Pivote, NO el Scrum master/Product manager

Consta de 3 etapas:

- Estimación Individual
- Estimación grupal que representa la tendencia de las estimaciones individuales
- Ajuste y consenso

Resultado final: historia de usuario tiene un tamaño acordado por el equipo.

Estimación por Fibonacci

De las más usadas

Mejor punteria para backlog, para tareas más específicas.

Se establece secuencia Fibonacci para realizar la estimación de las historias (1,2,3,5,8,13..)

> Muchas veces se ocupa para establecer tallas de camiseta (No recomendado).

- Se establece un Pivote
  Tiende a ser historia con valor 1 o 2.
- Muy usada con herramientas como Planning Poker
  Se incorporan cartas asociadas a la secuencia de Fibonacci y otras adicionales como "?" o "Cafe" para indicar un break.
- Estimación individual, grupal y ajuste (Al igual que el anterior)

Ojo con los puntos

13 puntos: Demasiado grande
8 puntos: Hasta ahí llegue, ideal separarlo.
5 para abajo: Bien

Clase 9

07/04/25

Product Backlog

Epica: Gestion de reuniones

Epica: Administrador de correos

1. Crear correo
2. Eliminar correo

3. Editar correo
4. Enviar correo
5. Crear etiquetas para administrar correos
6. Asignar etiquetas
7. Visualizar correo
8. Crear favorito
9. Asignar favorito
10. Asignar firma a cierto correo
11. Agregar emoticones

Critico 20%
Prioritario 40%
Medianamente prioritario 20%
Deseable 20%

- Requerimientos funcionales
- Requerimientos No funcionales
- Requerimientos de dominio

Planning

Objetivo:
¿*Qué* es lo que se puede entregar como parte del incremento resultante del sprint?
¿*Cómo* se realizará el trabajo necesario para entregar el incremento?

Historia de usuario 1 --> sub-task 1, 2, 3, .., n. ---> Esfuerto (de la historia)
|
Historia de usuario N

Pivote

Tarea más básica en base a esfuerzo

planning poker online

Clase 10

10/04/25

Modelamiento y UML

Abstracción del sistema. Diagramas que sirven para representar/diseñar sistemas.
No es necesariamente una representación fiel de la realidad (depende del contexto).

Distintas perspectiva:

- Externa
  Modela contexto o ambiente del sistema
- Interactiva
  Modela interacciones, componentes, ambientes
- Estructural
  Modela datos que son procesados
- Comportamiento
  Modela el comportamiento, acciones y respuesta a eventos

Modelamiento de Sistemas con UML

Usado para:

- Fomentar la discusión de un producto/sistema
- Documentar requerimientos
- Detallar y describir cada uno de los procesos de implementación

Diagramas UML para:

- Diagrama de contexto
- Diagrama de *actividades*
- Diagrama de *casos de uso*
- Diagrama de secuencia
- Diagrama de clases
- Diagrama de *estados*

Diagrama de Actividades

Creación de flujos de trabajo más importantes.
Flujos que forman parte de un proceso

- Permite identificar de forma genérica el panorama de nuestro requerimiento y los procesos a los que pertenece
- Permite el desglose de un progreso
- Inicio
  Procesitos
  Condiciones
- Fin

> Flujos parecidos a la Epica
> No es Historia de usuario

Diagrama de Casos de Uso

Funcionalidad de usuario (como interactua el usuario)

Involucra al usuario o rol y como este va a interactuar con el sistema
Mas sencillo de usar para demostrar al usuario

- Relato de como un usuario interactúa con el sistema

Diferencias entre Caso de uso e Historias de Usuario

- Ambos representan un trozo de funcionalidad de valor para el usuario
- Las historias de usuarios son mas pequeñas y tienen tiempo acotado

Estructura

- Supuestos iniciales
  Que esperan los usuarios al iniciar el escenario
- Normal
- Que puede fallar
- Otras actividades: lo que puede ocurrir mientras sucede un escenario
- Estado final: Cómo va a quedar

Clase 11

14/04/25 ultimatum: Hasta esta clase entra la materia

Modelamiento y UML PT2

Diagrama de Secuencia

Muestran las secuencias de distintos flujos.
Modelan interacciones entre actores y/u objetos de un sistema.

- Actor
- Objeto
- Linea de vida
- Barras de activación. Longitud del rectangulo indica la duracion de la activación

- <<message>>
- Fragmentos
  Interaccion entre dos objetos se produce cuando un objeto envia un wsp(mensaje) a otro. Síncrono o Asíncrono (Asíncrono preferible)

Tutorial:

- Definir objetos y actores
- Definir metodos, parametros y retornos
- Ubicar de izquierda a derecha, arriba hacia abajo

Tarea pa la casita arreglar Diagrama de Secuencia

Diagrama de Estado

> Dibujo ovarios

Modelo basado en eventos, muestra como un sistema responde a inputs y outputs.

- Basado en el supuesto que un sistema tiene un num finito de estados y que los eventos pueden causar una transición de un estado a otro.
- Permite ver los input y output en los distintos flujos y trazar las fallas que vayamos detectando en el proceso de ejecución.

Usado para representar una prueba funcional automatizada

- Puedo ver en que paso se me cae el sistema

EJ) ¿Qué pasa si el pago falla?

Orden para diseñar un sistema:

1. *Diagrama Casos de uso*
   Funcionalidad que quiere el usuario
2. *Diagrama de Secuencia*
   Secuenciación de cada uno de los casos y como interactúan los objetos
3. *Diagrama de Actividades*
   Flujos
4. *Diagrama de Estados* (Una actividad puede tener muchos estados)
   Cada uno de los estados de las respectivas interacciones

Clase 12

17/04/25

Simulacro de Solemne

Preguntitas

1. cual de las sgtes afirmaciones describe la principal diferencia entre un enfoque agil y uno tradicional?
   R: Enfoque agil: se permite entregar valor de forma incremental y ajustar funcionalidad segun feedback del mercado.
   Enfoque tradicional es mas rigido.
2. que riesgo puede enfrentar un equipo agil en el desarrollo de un producto?
   R:
   mala estimacion que no permite realizar entregas de valor
   incrementos de backlog sin priorizacion

   > Habla a nivel de desarrollo

Puede tener un sprint largo y perfectamente puede entregar valor

3. Cual de los sgtes marcos o metodologias corresponde a los enfoques agiles?
   R: Scrum, kanban, (XP) extreme programming, pair programming
4. Cual de las sgtes opciones representa una metrica adecuada para evaluar un requerimiento no funcional en un sistema de software?
   R: ninguna de las anteriores

- uptime (99,9% operativo)
- cantidad de request por minuto del sistema

> Requerimiento no funcional: Asociado a restricciones

5. Cual de las sigueintes afirmaciones describe adecuadamente un requerimiento funcional segun las mejores practcas en analisis de sistemas?
   R: Enunciado que detalla servicios que el sistema debe ofrecer, redactado en lenguaje natural
6. Cual es la diferencia entre caracteristica de meta y caracteristica de hostoria de un producto de usuario
   R: Meta describe objetivo medible y alineado al negocio
   Historia de usuario Representa funcionalidad concreta desde perspectiva del usuario
7. como seria equipo agil de rendimiento?
   R: multifuncional, autoorganizado y bonito
8. Riesgo comun al implementar scrum sin comprension de principios?
   R: Ausencia de product owner empoderado puede provocar perdida de foco

> Scrum se privilegia interacciones cara a cara, evita documentación detallada
> Es el roadmap y weas similares
> product owner prioriza product map: ¿Qué quiero?

Scrum master
Lider del equipo de desarrollo

Product owner
entrega de valor y backlog sprint a sprint. Representa al negocio. Vela por valor entregado
Tiende a perder foco

Product manager
Product Owner pero con vision mas elevada y estrategica. Incluyendo lucas (presupuesto) velando por los hitos. No vela por sprint. Vela por todo.

9. Roles principales que conforman un equipo scrum
   R: Scrum master, product owner, developers
10. Cuales representa ventaja del uso de UML en el desarrollo de software?
    R: Lenguaje grafico estandarizado que mejora la comprension, diseño y documentacin de sistemas desde multiples perspectivas.

Parte 2

Clase 13: Diseño e Implementación (PT2)

08/05/25

Tutorial como implementar lo visto previamente

Objetivos

Comprender las actividades mas importantes de un proceso de diseño general orientado a objetos

- Identificar algunos de los diferentes modelos que pueden usarse para documentar un diseño
- Conocer la idea de patrones de diseño y como estos son una forma de reutilizar el conocimiento

Programación Orientada a Objetos (POO)

Sirve para modelar y representar problemas complejos

- Pasamos de tener un codigo de arriba hacia abajo en que las funcionalidades estan mezcladas y son dificiles de separar

Pilares fundamentales

- *Abstracción*
  Proceso de enfocarse en características esenciales

- *Encapsulamiento*
  Clases
- *Herencia*
  Una clase hija hereda atributos y métodos de otra clase
- *Polimorfismo*
  Capacidad que un mismo objeto pueda responder a diferentes mensajes o métodos de manera distinta.
  Herencia pero adaptada para el caso específico

Utilidad

- Facilita reutilización de código
- Extensión de funcionalidades
- Mantenibilidad: sistema modular

Diagrama de Clases

Se usan cuando se desarrolla un sistema mediante el paradigma de POO (como plantilla)

- Objetivo: Modelar objetos del mundo real y sus relaciones como parte de los requerimientos o definiciones de diseño
- Representa también modelos de datos
  Complemento útil para aplicaciones webs

1. Public (+)
2. Private (-)
3. Protected (#)

Interfaz

Definición de comportamiento que especifican conjunto de metodos que las clases deben implementar

- Simplemente definen una firma comun para las clases que los implementan
- Permite a las clases compartir funcionalidad en comun sin relacionarse jerárquicamente
- Puede participar en subsistemas

Tipos de Relaciones

Tipos de relaciones entre clases

Clase 14

12/05/25

Problemas de Implementación

- **Reutilización**
  Problemas de capacidad del equipo
  Documentación
  Costos Implícitos:
  - Busqueda de componentes
  - Adaptacion a necesidades/requerimientos
  - Costo de integrar con componentes vigentes
- **Gestión de configuraciones**
  Versiones del proyecto

Principal objetivo
- Documentacion controlada (funcionalidad)
- Trazabilidad de cambios

- **Host-Target Development**
  Ejecución en otro equipo
  Ambiente de desarrollo

Clase 16

15/05/25

Demo: desarrolladores
Roadmap: Scrum master
Proximos pasos: PO

Clase 17: Testing

22/05/25

Testing

Objetivo
Entender las etapas del testing desde su desarrollo hasta la aceptación por parte del usuario o cliente.

Introducción
Objetivo: Demostrar que un sistema "hace" lo que debe hacer y en el proceso encontrar defectos antes de liberarse a los usuarios.

- No demuestran ausencia de errores sino su presencia por 2 enfoques:
  - *Validation Testing*
    Verifica que el sistema cumpla con los requerimientos
  - *Defect Testing*
    Busca inputs que generan errores o comportamientos inesperados

¿Estamos construyendo el producto correcto?
- Validar que satisfaga al usuario
- Validar que sea confiable para su uso
¿Estamos construyendo el producto de forma correcta?
- Verificar que se cumplan requisitos funcionales RF y NO funcionales
- Nivel de confianza depende del objetivo con el que fue creado el sistema

- Propósitos de software
  A mayor criticidad, mayor su confiabilidad
- Expectativas del usuario
  A medida que el software evoluciona, las expectativas también.

Proceso de Prueba y Actividades

- *Development testing* (prueba de desarrollo)
- *Release testing* (versiones de prueba)
- *User testing* (pruebas de usuario)

Mezcla entre testing manual y automatizado.

Tipos de Testing

1. *Testing Unitario / Unit Testing*
   - Asociado a metodo/clase especifico
   - Objetivo: cubrir todas las funcionalidades de un objeto
   - Para esto: Secuencia de pruebas
   - A veces tienen dependencias con otros objetos que pueden o no estar desarrollados
2. *Prueba de Caja Negra / Black Box Testing* SAS

   - No se necesita conocer la estructura interna del software o código fuente.
   - Verifica la funcionalidad del sistema desde el exterior
   - Basada en especificaciones
   - No requiere conocer el código
   - Se centra en inputs y outputs
   - Valida la funcionalidad visible

- Detecta errores de comportamiento

Ventajas

- Simula condiciones reales de usuario
- Permite encontrar errores funcionales importantes
- Útil cuando el código no está disponible/confidencial
- Reutilizable
  Desventajas (-)
- No puede detectar errores de implementación interna o código muerto
- Cobertura limitada si no comprueba todos los caminos
- Depende en gran medida de la calidad de requerimientos o su documentación

3. *Prueba de Caja Blanca / White Box Testing*

- Tengo conocimiento interno del código fuente
- Verifica el funcionamiento interno de una app incluyendo:
  - Estructuras de control
  - Flujos de datos
  - Rutas lógicas
  - Condiciones de decisión
- Basada en Estructura interna
- Requiere conocimientos técnicos
- Control de flujo de ejecución
- Mide cobertura de código
- Detecta errores lógicos

Ventajas

- Permite alta cobertura de código
- Detecta errores lógicos internos difícil de encontrar desde UI

Desventajas

- NO valida que el sistema hace lo que tiene que hacer
- Requiere acceso y comprensión del código fuente
- No considera errores por requisitos malentendidos o casos de uso omitidos
- Puede ser costoso manener si el codigo cambia constantemente

Pruebas End to End
Valida desde la entrada funcional hasta la salida puntual
Mezcla Caja blanca, negra

EJ) Caja negra
Selenium, postman, Jmeter, Appium, LoadRunner/ Jmeter
EJ) Caja blanca
Java, pytest, Clover, Mockito

Clase 18

26/05/25

Ej) Caja negra, puede testearse tanto como caja Blanca como Unit test

Automatización de pruebas

Como pueden haber muchos test, se recomienda automatizar pruebas

- CI/CD, Travis, Jenkins, Gitlab
- *Configuración*
  Donde se inicializa el sistema con los test cases (inputs y outputs)
- *Llamada*
  Donde se llaman los objetos/metodos para ser testeados

- *Declaración*
  Donde se compara elresultado de la llamada con el esperado
  Like, Dislike

Para crear un caso de prueba

- Subclase de Unit test --> *TestUnit*
- QA

Test Driven Development

Enfoque donde se mezcla el desarrollo y testing

- Desarrollo incremental. No se comienza a desarrollar hasta que se pasan todos los test

CI/CD

Continuous Improvement and Deployment (Revisar)

Automatizar software y desarrollar mejor calidad de código

Continua integración, Entrega, Desarrollo

La idea es que al hacer pruebas automatizadas, automaticamente paso mi código a producción.
Llego, construyo, hago el merge y pasa una versión completa entregando las fallas y

Quality Assurance

Calidad de software

- Velando que sea todo y más
- No podemos fallar a nuestros usuarios. Si no cumplimos, los usuarios no usarán el software y se perderá confianza.
- Mejorar procesos de trabajo y eficiencia

¿Cómo nos aseguramos?

- Documentacion (versiones)
- Gestion de riesgo
- Colaboración, evaluación de habilidades de proyecto
- Definiendo estrategias para:
  - Arquitectura aplicaciones/servicios
  - Pruebas
  - Entornos de pruebas (QA, Staging, UAT/E2E)
  - CI/CD
  - Control de versiones y estrategia de ramificación

Quality Control

Buscar activamente defectos y/o erroes

QA Mido lo que tengo que "controlar"
QC Métricas para medir calidad

1. Quality Assurance
2. Quality Control
3. Testing

- Exitoso
- Con observaciones
- Con problemas (error, fallidos)

Clase 20

02/06/25

Objetivo
Entender que existen patrones de diseño

- Proposito, Estructura, Aplicaciones, Consecuencias
- Va antes que Testing
- Garantizar proyectos escalables

Principios S.O.L.I.D.

1. Singel Responsibility
2. Open/Closed
3. Liskov Sustitution
4. Interface Segregation
5. Dependency Inversion

Ventajas

- Facilita mantenimiento del código
- Reduce la complejidad de añadir nuevas funcionalidades
- Aumenta la reusabilidad de piezas y componentes
- Mejora la calidad del código y su comprensión

1. SRP - Single Responsability Principle

Cada clase debe ser responsable de una única responsabilidad

2. OCP - Open/Closed Principle

Las entidades (clases, modulos, funciones, etc) deben estar abiertas para extension, *cerradas* a su modificación.
No puedo literalmente cambiar sus atributos (pisar la info que ya esta dentro)

- Debe poder agregar nuevas funcionalidades sin modificar el código existente
  ¿Cómo? Polimorfismo
  Misma función, distintos datos

3. LSP - Liskov Sustitution Principle

La subclase debe seguir la lógica del padre, genera contradicción
Por ej: Clase padre pájaro "vuela"-> Sub clase perro "vuela" (NO tiene sentido)

4. ISP - Interface Segregation Principle

Una clase no debe estar obligada a implementar interfaces que no usa
Humano puede trabajar y comer, Robot come (*Para qué*)

5. DIP - Dependency Inversion Principle

Los métodos de alto nivel no deben depender de módulos de bajo nivel.
clase App depende directamente de MySQLDatabase()
Ñao ñao, acoplamiento fuerte

Separo la conexión y especifico la base de datos
Conecto a la base de datos INDEPENDIENTE de que tipo sea *Desacople*

Patrones de Diseño

Para resolver problematicas a nivel de desarrollo comunes de la industria.

No hay varita mágica donde una combinacion de variables/clases me haga funcionar todo.
Son buenas prácticas.

- Descripcion de un problema y solución que recibe un nombre y que puede aplicarse en otros contextos.
- *Patrón*: Problema bien conocido, características similares a otro el cual ya ha sido solucionado
  - Descripcion

- Escenario de uso
- Solución concreta
- Consecuencias de usar patrón (ventajas, desventajas)
- Ejemplos de implementación (documentación)
- Lista de patrones relacionados

Patrones Creacionales

Abstraen el proceso de creación de instancias de objetos
Ayudan a hacer un sistema independiente de cómo se crean, componen y representan sus objetos

Cuando inicio - **Creación**

Abstract Factory

Interfaz para crear familias de objetos relacionados sin tener que especificar sus clases concretas.

Factory Method

Necesito agregar nuevos tipos de x en el futuro sin modificar el código existente

- Definir método de creaión en una interfaz (clase base) y luego tener subclases que implementen este método para crear objetos
  EJ: Clase Logistics con metodo create_X() y luego RoadLogistic, SeaLogistic, AirLogistic para separar "por tierra, mar y aire"

Clase 22

16/06/25

Patrones de Comportamiento

Diagramas de SECUENCIAS

Caracterizan el modo en el que las clases y objetos interactuan y se reparten la responsabilidad

- Definen algoritmos, flujos de control o formas de interracción entre objetos para evitar acoplamiento de clases

Principales Diferencias

- *Creacionales*
  - Enfoque en el proceso de creación de objetos
- Estructurales
  - Organización y relaciones entre distintos objetos
- *Comportamiento*
  - Interacción y comportamiento

Mediator

Patron que funciona como mediador entre el Usuario y el Receptor

- Mayor escalabilidad
- Proporciona un objeto central para que los componentes se comuniquen entre sí

Chat implementado en Google Chat

Memento

Para problemas de encapsulamiento (por ej public text (tiene que ser priv))

- Permite capturar y externalizar el estado interno de un objeto sin violar el encapsulamiento

Observer

Problema de Suscripciones (Notificaciones a todos)
Permite definir una dependencia de 1 a N

- Cuando un objeto cambia su estado, todas sus dependencias son notificadas y actualizadas automáticamente

State

Problema de reproductor de música con diferentes estados (reproduciendo, pausado, detenido)
Si yo tengo más estados, difícil escalabilidad

- Permite cambiar el comportamiento de un objeto cuando su estado interno cambia
- Se puede implementar definiendo:
  - Interfaz de estado
  - Subclases para cada estado específico
  - Invoco métodos en el objeto de estado actual

Strategy

App que calcula el camino más corto. ¿Cómo puedo organizar mi codigo para que sea extensible?

Problema: No cumple con SRB Responsabilidad única, a mas rutas + modificaciones en la clase

- Define una familia de algoritmos, encapsula cada uno y los hace intercambiables
- Se puede cambiar el algoritmo en tiempo de ejecución

Template Method

Monopoly que incluye varias etapas comunes (inicializacion del juego, turno de jugadores y finalizacion del juego).
Problema: Escalamiento a múltiples juegos con la misma estructura duplicando código

- Creo un template (esqueleto) de algoritmo en un método, dejando algunos pasos para subclases

Visitor

Editor de gráficos que puede trabajar con distintos tipos de formas (circulos, rectangulos, lineas)
Problema: Escalamiento a múltiples figuras donde cada una tiene operaciones distintas

- Cada persona que manipule el objeto lo hace de forma independiente
- Puedo manipular paralelamente distintas figuras
  EJ) Prueba

Prototype NO VA

HASTA FLYWEIGHT