Modelo de ciclo de vida

Es quien me dice el orden de las etapas de ciclo de vida

Tipos de ciclos de vida

Ciclo de vida en cascada

- Captura de requerimientos (mediante entrevistas, cuestionarios, brainstorming, casos de uso, user stories, etc).
- Análisis (se organiza, se transforma del idioma del cliente al del desarrollador)
- Diseño (respondo cómo solucionar los requerimientos no funcionales y funcionales)
- Implementación
- Testing (funcional, unidad, escala total, negativo, performance)
- Mantenimiento (70-80% costo total)

Ventajas:

Es un modelo fácil de implementar y entender.

Desventajas:

- En la vida real, un proyecto rara vez sigue una secuencia lineal
- El proceso de creación del *software* tarda mucho tiempo
- Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.
- Una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior.

Ciclo de vida iterativo e incremental

En casa una de sus partes hago un ciclo de vida en cascada, a medida que voy incorporando una iteración voy incrementando el sistema. Es una porción del sistema final.

Ciclo de vida en espiral

Para cada ciclo habrá cuatro actividades:

1. Determinar objetivos
2. Análisis del riesgo
3. Desarrollar y probar
4. Planificación

Ventajas:

- Reduce riesgos del proyecto
- Incorpora objetivos de calidad
- Integra el desarrollo con el mantenimiento, etc.

Desventajas:

- Genera mucho tiempo en el desarrollo del sistema

- Modelo costoso
- Requiere experiencia en la identificación de riesgos

Metodología de desarrollo de Software

- Paradigma estructurado (datos, funciones)
- Paradigma orientado a Objetos (UML, UP)
- Métodos formales (Teoremas)
- Métodos ágiles (rescatan la esencia de una metodología sin ser tan rigurosa)

UP

- Organiza el UML
- Basado en casos de uso: Define funcionalidad con los mismos.
- Centrado en la arquitectura
- Iterativo e Incremental:

(Pasar de un subproblema a otro uniéndose con el anterior realizado
Dividir problema en subproblemas y a cada uno se le aplica el ciclo de vida en cascada)

- Solo útil cuando se usa en un sistema chico.
- Define procesos de desarrollo
- Workflow: Orden de las tareas, como hacerlas y quien las realiza.

UML

- Nos dice el orden por el cual tengo que desarrollar los diagramas
- No dice la etapa del ciclo de vida para cada uno
- Define un catálogo de diagramas que pueden ser usados
- No importa la metodología usada

UML separa en 2 sus diagramas:
Diagramas estáticos/estructurales
1. Diagrama de Clases
2. Diagrama de Objetos
3. Diagrama de Componentes
4. Diagrama de Deployment
5. Diagrama de Paquetes

Diagramas dinámicos/comportamiento
6.Diagrama de Casos de Uso
7. Diagrama de Actividades
8. Diagrama de Transición de Estados
9. Diagrama de Interacción:
   a. Diagrama de Colaboración
   b. Diagrama de Secuencia

SCRUM

Metodología ágil (Planifica, ejecuta y reflexiona). Te ayuda a fallar en 30 días o menos. Las iteraciones se llaman Sprints, duran de 1 a 4 semanas. Es autodirigido y autoorganizado, no hay jefe dentro del equipo. Cada equipo se organiza como quiere.

Valores:

- Coraje (Tomó la postura de hacer las cosas de la mejor manera sin atajos)
- Foco (Hago foco en las tareas del Sprint y estoy atento al objetivo)
- Compromiso (Estoy comprometido en el proceso, distinto a estar relacionado)
- Respeto (Respeto entre todos, no hay jerarquías, nadie manda a nadie)
- Apertura (Estoy abierto a problemas)

Pilares:

- Transparencia (Hacia el cliente, debería saber todo el cliente sobre el equipo de trabajo)
- Inspección (Veo que me fue bien o mal al final de cada Sprint, para el próximo Sprint mejorar)
- Adaptación

Equipo de Scrum:

- Compuesto por 3 roles
- Equipo de desarrollo 7 +- 2
- Product owner (cliente que sabe que hay que hacer, que asume estar disponible ante una duda y responsable. Expresa los requerimientos del sistema, los va a priorizar y asegurar que el equipo de desarrollo los entienda)
- Scrum master (es quien sabe usar scrum, el facilitador, asegura que se cumpla todas las prácticas y reglas de Scrum)

Eventos:

- Sprint:
    - ➢ Solo puede ser cancelado por el Product Owner (el objetivo del Sprint se volvió totalmente obsoleto, corto y planificar de nuevo)
    - ➢ El cliente canceló
    - ➢ Condiciones de mercado cambian
    - ➢ El equipo de desarrollo se da cuenta que no hay ninguna chance de que puedan terminar el Sprint (alcanzar el objetivo)
    - ➢ Soft potencialmente instalable/usable

    - ➢ Criterio de DONE, tiene que ser el mismo para todos, depende de empresas, procesos, etc)
- Sprint planning
    - ➢ Fijar el objetivo del Sprint, participa el equipo de Scrum, Scrum master.
    - ➢ Lleva la reunión aproximadamente 8 horas (se sugiere para un Sprint de un mes)
    - ➢ Se detallan más las funcionalidades
- Sprint review
    - ➢ Scrum Team y personas interesadas en el desarrollo
    - ➢ Máximo dura 4 horas para Sprint de un mes.
- Daily Scrum
    - ➢ 15 min
    - ➢ Sincronizan actividades(que hice ayer, qué haré hoy y si estoy con problemas)
    - ➢ Planifican el próximo día de trabajo
    - ➢ Scrum master y desarrollo
- Sprint retrospective
    - ➢ Participa solo el equipo de desarrollo, a veces el P.O o el SM
    - ➢ 3 horas para un Sprint de un mes, cada miembro responde 3 preguntas. Que funcionó bien? Que podemos mejorar? A qué nos comprometemos para el siguiente Sprint?

Artefactos:

- Product Backlog
    - ➢ Lista ordenada de los requerimientos del producto (que está priorizada por el P.O)
    - ➢ Evoluciona con el producto
    - ➢ Listar requerimientos, funcionalidades, mejoras, etc
    - ➢ Responsabilidad del equipo de desarrollo estimar cuánto tiempo va a llevar.
- Sprint Backlog
    - ➢ Conjunto de ítems del P.O que se seleccionaron para el Sprint
    - ➢ El equipo de desarrollo refina durante el Sprint. Va agregando info a medida que surjan dudas
    - ➢ Solo el equipo de desarrollo puede inferir
- Incremento
    - ➢ Suma de todas las PBIs completadas durante un Sprint y el valor de todos los incrementos pasados
    - ➢ El incremento está DONE

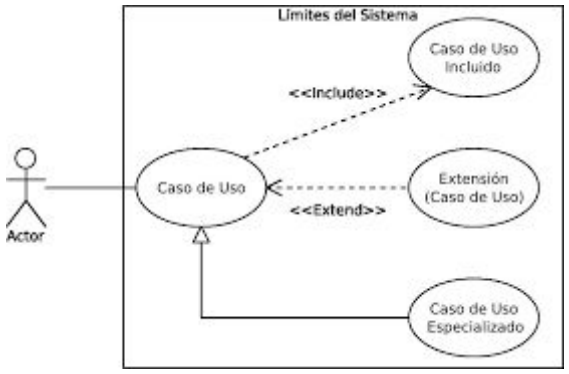
User stories:

Son una forma de especificar PBIs, son fáciles de entender, para gente sin background técnico. Describe qué se debe implementar, no como.

- Tarjeta: Como <usuario> quiero <objetivo> para asi <beneficio>(si es obvio para todo el equipo de desarrollo, no se pone).
- Conversación: Los requerimientos siempre son detallados a partir de una reunión.
- Confirmacion: Condiciones de la satisfacción (por ejemplo, como usuarios de un foro, quiero subir archivos, asi los puedo compartir en un post. Verificar con archivos .txt, .png, etc. Verificar con archivos menores o igual a 1gb).

Casos de uso:

Son importantes para visualizar, especificar y documentar el comportamiento de un sistema, un subsistema o una clase. Modela las necesidades a satisfacer y los límites del sistema.



Relaciones entre CU:

- Herencia: cuando hay varias formas de hacer lo mismo.(pero con línea punteada)
- Inclusión:→creó un nuevo caso de uso con el comportamiento que comparten A y B. Es siempre explícita, para siempre que se activan esos caso de uso, por eso la flecha hacia el caso de uso C.
- Extensión:← Entre las guardas va la condición en negativo, puede haber más de una condición. Es implícita, se activa bajo una condición. Cuando esa condición se cumple, ahi se activa. La flecha va hacia el caso de uso A (al revez de que la inclusión).

Especificación en un caso de uso:

- Nombre: nombre del caso de uso
- Actor Primario: culpable del que el caso de uso se active
- Actor secundario: uno o muchos que ayudan al actor primario a activar el caso de uso.
- Descripción: breve descripción del caso de uso
- Precondición: es algo que el caso de uso tiene que controlar antes, que después se tiene que describir en el curso básico.

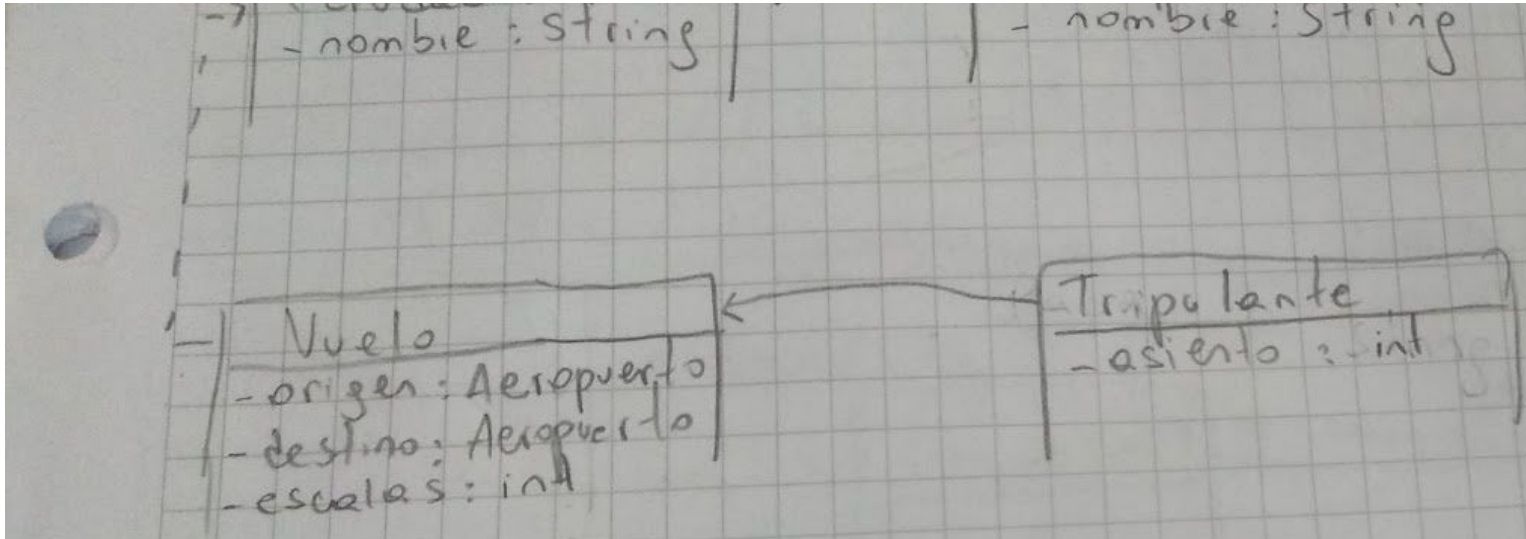
- Post condición: cuando finaliza, cumplo esa condicion.
- CU Curso Básico: lo que me gustaría que pasara
- CU Curso Alternativo: lo que hago si algo falla
- Trigger: es el disparador del caso de uso
- CU que extiende: extensión
- CU Incluido: inclusión, pongo <<include>> “nombre del caso de uso”
- Suposición: algo que tiene que ser verdadero, pero el CU no lo cumple. SI hay suposiciones, no se verifican en curso básico. (supongo que el usuario ya está logueado).
- Terminación: todas las condiciones por el cual el CU termina.

Diagramas de clases:

Modela un conjunto de clases colaboración e interfaces así también como sus relaciones. Representan elementos lógicos de un sistema. Se usa cuando quiero hacer un sistema de bajo nivel.

- Clase: contiene atributos, operaciones y semántica.
- Nombre de la clase: empieza con mayúscula.
- Atributos: empieza con minúscula.
- Operaciones: empiezan con minúscula y se ponen los paréntesis, redimensionar()
- Visibilidad (modificadores de acceso): cuando es package no se pone nada
  private -
  public +
  protected #
- Clase abstracta: se pone el nombre de la clase en cursiva
- Métodos abstractos: se escribe en cursiva
- Tipos de atributos: se pone el atributo dos puntos y el tipo -nombre:String
- Atributos estáticos: se pone una línea subrayada -nombre:String
- Métodos estáticos: en subrayado
- Más info en los métodos: se ponen entre paréntesis
  +mover(pintoX:int, puntoy:int) o sino +mover(int, int)
- Tipos de retornos: -esVisible():boolean , por lo general si es void no hace falta ponerlo, si es abstracto si.
- Adornos:
  - Nombre de la relación, de mayor semántica.
  - Como leerlo: hacia que lado, con una flecha, dirección.
  - El rol: puedo elegir en vez del nombre de la relación. Se usa menos (porque es más difícil de usar).
  - Multiplicidad: Indica con cuantas instancias de una clase, se van a relacionar con otra.

    - Navegabilidad: Da info en tiempo de ejecución, pone una restricción. Se indica solamente con una flecha los objetos de un tipo, puede mandarle msj a otro.
- Relación: conexión entre clases, sirve para indicar cómo los objetos se relacionan entre sí.

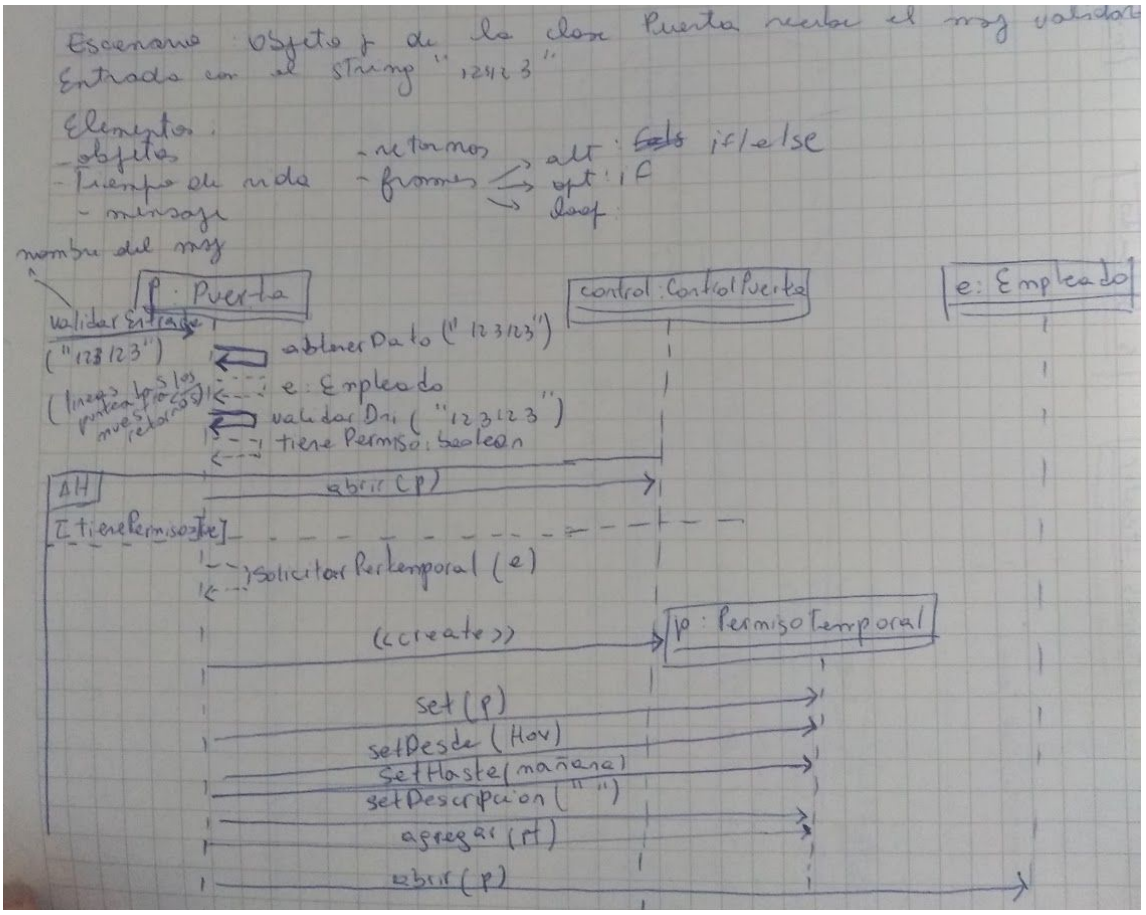


ASOCIACION ——→ conoce A (estructural) adornos (navegado una clase a
GENERALIZACION ——△ ES UN (métodos) herencia (polifirmo un método)
DEPENDENCIA - - -> USA (parametros) pasa su clase por parametro
REALIZACION - - -△ IMPLEMENTA (métodos) (interfaz)
AGREGACION ◇——→ (estructural) es parte de (cuando hay
COMPOSICION ◆——→ adornos una clase dentro de
otra clase (o la misma)

- Relaciones:
    - Generalización(herencia): Una clase extiende de otra, relación padre e hijo. Se como “es un”, “es una categoría de” y se indica con un triángulo vacío (flecha).
    - Asociación: Relación “estructural” entre dos clases. Una clase tiene un objeto de otra. Se lee como “tiene”, “es de”, “conoce a”. Se pone una línea llena →
        - Agregación: es un tipo de asociación, agrega mayor semántica a la relación. Relación del todo con las partes (por ejemplo la facultad se divide en departamentos)
          Lleva adornos, en algún momento voy a tener una lista. “A está formado por B”. Cuando se destruye A no se destruyen los elementos de B. Se lee “es de parte de”⟶ (sin relleno)
        - Composición: Mismo que agregación pero, cuando se destruye A se destruyen los elementos de B relacionados. Ante la duda siempre hacer Asociación, lleva adornos.

    - Realización: Indica que una interfaz va a ser usada por otras clases. Se indica con una flecha entre cortada (B va a implementar los métodos de la interfaz A). Se lee “implementa”
    - Dependencia: Relación débil, es cuando hay una relación entre dos clases. En caso de no ser una asociación. Se indica con una flecha entre cortada (distinta a realización, no es un triángulo) (en sentido de quien depende de quien). Se lee “usa”
      a. Retorno de un método
      b. Variable interna de una operación
      c. Un parámetro de un método
      d. Import

<u>Diagramas de secuencia:</u>
Es un tipo de diagrama de Interacción que resalta el orden temporal de los mensajes enviados entre los distintos objetos.
Escenario: Objeto p de la clase Puerta, recibe el msj “validarEntrada” con el String “12312312”
Elementos: Objetos, tempo de vida, mensaje, retornos, frames(alt: if/else, opt:if y loop)


Escenario: objeto p de la clase Puerta recibe el msj validar Entrada con el String "123123"
Elementos:
- objeto
- Tiempo de vida
- mensaje
- retornos
- frames → alt: if/else
→ opt: if
→ loop
nombre del msj
p: Puerta
control: ControlPuerta
e: Empleado
validarEntrada ("123123")
obtenerDato ("123123")
e: Empleado
validarDni ("123123")
tienePermiso: boolean
alt
[tienePermiso=true]
abrir (p)
solicitarPerTemporal (e)
«create»
p: PermisoTemporal
set (p)
setDesde (Hoy)
setHaste (mañana)
setDescripcion ("")
agregar (pt)
abrir (p)