Trabajo practico N°2

**1. Definición profunda de caso de uso**

Los casos de uso explican lo que un usuario espera del sistema desde una perspectiva externa, sin enfocarse en los detalles técnicos. Este enfoque facilita la comprensión de la interacción del usuario con el sistema y lo que espera ver. Es clave en las primeras etapas del desarrollo, ya que permite identificar necesidades reales sin caer en soluciones prematuras.

**2. Propósito esencial en el contexto de interacción**

El propósito central de un caso de uso es modelar cómo interactúan el sistema y los actores externos (usuarios, otros sistemas). Esto ayuda a definir los requisitos funcionales desde el punto de vista del usuario y comprobar que el sistema hará lo que realmente se necesita.

**3. Representación gráfica y simbolismo**

* Caso de uso: se dibuja como una elipse.
* Actor: se representa con una figura de palo.

Este tipo de diagramas es simple y de alto nivel, por lo que pueden entenderse fácilmente incluso por personas no técnicas. Por su claridad, se usan en etapas tempranas para acordar las funcionalidades principales.

**4. Tipos y roles de actores**

Los actores pueden ser:

* Humanos: como un recepcionista.
* Sistemas externos: como el sistema de registros PRS.
* Dispositivos físicos: como un hardware conectado.

Todos ellos interactúan con el sistema e influyen en su comportamiento.

**5. Notación de flechas y su justificación**

En UML, no se utilizan flechas entre actores y casos de uso, porque no se trata de un flujo de datos sino de una relación de colaboración. Las flechas sí se usan en otros diagramas como los de secuencia, donde indican mensajes. Usarlas en casos de uso podría confundir.

**6. Ampliación y refinamiento de detalles en casos de uso**

Un caso de uso puede detallarse de tres maneras:

* Con una descripción textual sencilla.
* Mediante una tabla estructurada (actores, estímulos, datos, etc.).
* A través de un diagrama de secuencia que muestre paso a paso la interacción.

Esto es necesario porque el diagrama general solo ofrece una vista superficial, mientras que para el desarrollo se requiere mayor profundidad.

**7. Estructura detallada de la descripción textual (ejemplo: “Transferencia de datos”)**

Incluye:

* Actores: recepcionista, PRS.
* Descripción: qué hace el usuario.
* Datos: información personal o resumen médico.
* Estímulos: acción que inicia el caso (por ejemplo, presionar un botón).
* Respuesta: confirmación de que la acción fue completada.

Cada sección ayuda a documentar con claridad el caso de uso y preparar su implementación y pruebas.

**8. Análisis de caso de uso: Ver información del paciente**

* Actor: Recepcionista.
* Objetivo: Ver los datos de un paciente.
* Flujo básico: El recepcionista introduce un ID → el sistema consulta la base de datos → verifica autorización → si es válida, muestra la información del paciente.

**9. Vínculo crucial entre casos de uso y requisitos del usuario**

El modelado con casos de uso permite:

* Comprender lo que el usuario necesita lograr.
* Validar que el sistema lo cumpla.
* Documentar requisitos funcionales claros.

Así, el desarrollo permanece alineado con lo que el usuario espera.

**10. Ventajas estratégicas del modelado de casos de uso**

* Es fácil de entender por usuarios y desarrolladores.
* Permite identificar claramente las funcionalidades.
* Sirve como guía para pruebas y desarrollo incremental.

En resumen, ayuda a planificar, construir y validar software centrado en el usuario.

**Parte 2: Diagramas de Secuencia (10 puntos)**

**1. Objetivo y enfoque de los diagramas de secuencia**

Representan la secuencia cronológica de interacciones entre actores y objetos. A diferencia de los casos de uso que muestran una visión general, aquí se detalla lo que ocurre en cada paso.

**2. Elementos estructurales y líneas de vida**

* Los actores u objetos se colocan en la parte superior.
* Las líneas de vida son líneas punteadas verticales que indican el tiempo durante el cual cada objeto está activo.

Esto permite ver cuándo y cómo participa cada entidad.

**3. Interacciones, mensajes y activaciones**

* Las flechas representan mensajes o llamadas entre objetos.
* Un rectángulo en la línea de vida indica una activación (el momento en que el objeto está procesando).
* La longitud del rectángulo muestra cuánto dura esa actividad.

**4. Principio de lectura y flujo temporal**

Se lee de arriba hacia abajo, representando el paso del tiempo. Esta convención facilita entender el orden de las acciones.

**5. Información adicional en los mensajes**

Las flechas pueden incluir:

* El nombre del método.
* Los parámetros.
* Condiciones.

Ejemplo: verInfo(PID) significa que se está pidiendo la información del paciente con ese ID, lo que da mayor claridad.

**6. Modelado de alternativas y condicionales**

Se usa el contenedor alt para mostrar decisiones.

Las condiciones se colocan entre corchetes [ ], por ejemplo:

* [autorizado] mostrar datos
* [no autorizado] mostrar error

Esto permite reflejar bifurcaciones en la lógica.

**7. Escenario: “Ver información del paciente” – Inicio**

* Actor: Recepcionista.
* Mensaje inicial: ViewInfo(PID).
* Receptor: Interfaz del sistema (instancia P de PatientInfo).

Se busca acceder a los datos del paciente mediante su identificador.

**8. Escenario: “Ver información del paciente” – Autorización**

* El objeto P consulta la base de datos con el PID.
* La base verifica con el sistema de autorización.
* Si la autorización es válida → se devuelven los datos.
* Si no → se muestra un mensaje de error.

Este paso es clave para asegurar la privacidad.

**9. Escenario: “Transferencia de datos” – Alternativas**

Hay dos formas de enviar datos al sistema PRS:

* Enviar directamente los datos personales desde la interfaz.
* Enviar un resumen de datos desde la base.
* La elección depende del tipo de información a transferir.

**10. Impacto estratégico de los diagramas de secuencia en el desarrollo**

* Permiten validar la lógica del sistema antes de programar.
* Ayudan a detectar errores o requisitos faltantes.
* Son fundamentales para diseñar interacciones seguras y precisas.

Contribuyen a que el software sea funcional, coherente y alineado al modelo definido.