Politécnico Internacional de Occidente PIO

Curso Metodologías Ágiles para la Programación

Profesor Julio Cesar Castaño

Estudiante Laura Barona Saavedra

Grupo 6

Agosto 2024

Taller:

1. Investigar que son los requerimientos de software.  
   Un requerimiento es una característica que debe incluirse en un nuevo sistema y puede consistir en una forma de captar o procesar datos, producir información, controlar una actividad o dar apoyo a una tarea. Para identificar requerimientos es menester comenzar por un adecuado relevamiento de información.
2. Concepto general  
   Los requerimientos de software son simplemente una descripción de lo que un programa de software en particular debe hacer. Actúan como pautas para que los desarrolladores creen un producto funcional que satisfaga las necesidades de los usuarios.
3. Definición de requerimiento Funcional  
   En general, los requerimientos funcionales describen acciones específicas que el ingeniero de software debe ser capaz de realizar durante el desarrollo de software. Los requerimientos funcionales a menudo se dividen en reglas de negocio y casos de uso. Las reglas de negocio son declaraciones de alto nivel que definen lo que un sistema debe hacer, mientras que los casos de uso son descripciones más detalladas de cómo debe funcionar el sistema.
4. Definición de requerimiento No funcional  
   Los requerimientos no funcionales describen características específicas que el software debe poseer durante el desarrollo de la aplicación. Por lo general, se dividen en tres categorías: rendimiento, seguridad y calidad.
5. Ejemplos.  
   Algunos de los requerimientos funcionales más comunes son:

* Las características y funcionalidad deseadas del producto
* Plataformas para desarrollar aplicaciones, por ejemplo, iOS, Android y web
* Especificaciones de diseño en términos de tema, colores y fuentes
* Funcionalidad de back-end: integración APl y bases de datos
* Plazos de finalización

Algunos de los requerimientos no funcionales más comunes son:

* Fácilmente Modificable
* Seguridad
* Portabilidad
* Confiabilidad
* Fácil de probar
* Usabilidad
  + Tiempo de Capacitación
  + Número de Slecciones
  + Número de Clics
* Desempeño
* Escalabilidad
* Eficiencia
  + Tiempo
  + Transacciones por segundo
  + Tiempo de Respuesta
  + Tiempo de Operaciones Completas
* Espacio
  + Memoria Principal
  + Memoria Auxiliar
  + Caché

**¿Qué es UML?**

UML son las siglas de “Unified Modeling Language” o “Lenguaje Unificado de Modelado”. Se trata de un estándar que se ha adoptado a nivel internacional por numerosos organismos y empresas para crear esquemas, diagramas y documentación relativa a los desarrollos de software (programas informáticos).

**Conceptos**

El lenguaje de modelado unificado (UML) es un estándar para la representación visual de objetos, estados y procesos dentro de un sistema. Por un lado, el lenguaje de modelado puede servir de modelo para un proyecto y garantizar así una arquitectura de información estructurada; por el otro, ayuda a los desarrolladores a presentar la descripción del sistema de una manera que sea comprensible para quienes están fuera del campo. UML se utiliza principalmente en el desarrollo de software orientado a objetos. Al ampliar el estándar en la versión 2.0, también es adecuado para visualizar procesos empresariales.

**Ejemplos**

**Diagramas de Estructura**

Los diagramas de estructura representan los elementos individuales de un sistema. Por lo tanto, son especialmente adecuados para la representación de la arquitectura de software. La representación estática no representa un cambio, sino estados y dependencias en un momento determinado. Los elementos individuales u objetos están relacionados entre sí. Por ejemplo, un objeto pertenece a una clase. Otros componentes son nodos de ordenador o artefactos –un artefacto representa un resultado, por ejemplo, un archivo de script terminado.

Los diagramas UML de esta categoría representan un sistema completo o una subestructura. Esto último ayuda, por ejemplo, a clarificar la estructura en detalle. En UML 2.x, el lenguaje asigna siete tipos de diagramas a la categoría Estructura:

**Diagrama de Clases**

Si los objetos tienen un comportamiento común o la misma estructura, pueden clasificarse (asignarse a una clase). La clase es, por tanto, un elemento simplificador (abstracción) para la representación visual. Las clases y los objetos están conectados entre sí mediante interfaces. El diagrama de clase muestra todos estos componentes y sus interrelaciones. Una clase representa el diagrama con un rectángulo. Contiene el nombre de la clase en negrita, como se muestra a continuación.

Diagrama

Descripción generada automáticamente

*Figura 1. La clase "Persona" tiene los atributos nombre y apellido. Las operaciones determinan si la instancia se muestra (display) u oculta (hide).*

**Diagrama de Objetos**

El diagrama de objetos tiene una estructura similar a la del diagrama de clases. Cuando el nombre aparece en el diagrama de clase (ver Persona arriba), el diagrama de objeto especifica el nombre de la instancia junto con el nombre del clasificador/categoría. De acuerdo con el pliego de condiciones, se subraya lo siguiente (por ejemplo: Helga:Persona)

**Diagrama de Componentes**

Un componente es un módulo que está aislado del sistema externo e interactúa con otros componentes mediante interfaces definidas. Es un subformulario de la clase. Por lo tanto, las características estructurales, como las operaciones y los atributos, definen el componente con mayor precisión. El componente contiene varios componentes. Estos pueden ser de nuevo componentes, pero también clases, subsistemas o partes. Hay dos opciones de visualización diferentes para el modelado: Black Box o caja negra (el contenido está oculto) y White Box o caja blanca (el contenido es visible).

Diagrama

Descripción generada automáticamente

*Figura 2. Vista de caja blanca en un componente. Las interfaces necesarias se muestran como cajas ("Socket") y se ofrecen como círculos ("Lollipop").*

**Diagrama de Estructura Compositiva**

Los objetos pertenecen a clases. Estos, a su vez, también pueden clasificarse. Estas metaclases se denominan clasificadores en UML. El diagrama de estructura de la composición representa las partes y conectores de un clasificador. Las partes son siempre parte del todo, incluso si no son necesarias para completar el clasificador. Los conectores son las conexiones entre las partes. Las características o servicios que requieren componentes externos al clasificador envían las piezas a través de una interfaz.

**Diagrama de Paquete**

Un paquete agrupa elementos como interfaces o clases en un espacio de nombres. Los paquetes (packages) también pueden fusionarse con otros paquetes (fusión de paquetes), importarlos (importación de paquetes) o contener otros paquetes (subpaquetes). Los paquetes estructuran el contenido del diagrama jerárquicamente como en un diagrama de árbol. El diagrama de paquete se utiliza, por ejemplo, en el metamodelo de UML 2. En los sistemas de software representa las subáreas de forma modular. Según la especificación, un paquete consta de una cabecera y un área de contenido

Diagrama

Descripción generada automáticamente

*Figura 3. Paquete con clase A y B y una interfaz*

*En la cabecera el paquete de palabras clave describe el elemento, "paquete" es el nombre. En el área de contenido, el paquete incluye una interfaz y dos clases.*

**Diagrama de Distribución**

El diagrama de distribución modela la distribución física de los artefactos en nodos. Los nodos pueden ser hardware (device nodes), que puede proporcionar memoria, o software (execution environment nodes), que proporciona un entorno para ejecutar procesos. Se representan como cuboides tridimensionales. Los artefactos se dibujan como rectángulos conteniendo el nombre del archivo. Para distinguirlo de una clase, añade el estereotipo <<artefact>>. El diagrama es adecuado para visualizar dependencias entre nodos y artefactos, las llamadas relaciones de distribución.

**Gráfica de Perfil**

Los diagramas de perfil se utilizan a nivel de metamodelo. Se utilizan para asignar un estereotipo a las clases o un perfil a los paquetes. En el metanivel tienes la posibilidad de adaptar el modelo para otra plataforma o dominio. Por ejemplo, si restringes la semántica UML dentro de un perfil, transfieres las especificaciones a las clases subordinadas.

**Diagramas de Comportamiento**

Los diagramas de comportamiento cubren las especificaciones restantes bajo UML. A diferencia de los diagramas estructurales, no son estáticos, sino que representan procesos y situaciones dinámicas. Los diagramas de comportamiento también incluyen los diagramas de interacción (ver abajo).

**Diagrama de Casos de Uso**

El diagrama de casos de uso muestra el comportamiento que se esperará de un sistema más adelante. Este modelo no solo es adecuado para sistemas de software, sino también, por ejemplo, para procesos esperados en las relaciones comerciales. El caso de uso involucra a un actor (humano o sistema) con un objetivo. El diagrama normalmente tiene como nombre el objetivo. Los diferentes casos de uso dentro del sistema cumplen el objetivo del actor.

El diagrama del caso de uso representa a UML con un rectángulo con la etiqueta use case. El remitente es el actor (esto se representa como una figura de palo, como en el caso anterior, incluso si se trata de un sistema). El actor está conectado por una relación de dependencia (representada como un guion) con el caso de uso (elipse con etiqueta) dentro de un sistema (rectángulo con etiqueta <<sistema>> y nombre del sistema).

Diagrama

Descripción generada automáticamente

*Figura 4. El diagrama de casos de uso en el ejemplo tiene un actor que puede esperar dos casos de uso dentro del sistema “red social XYZZY”*

**Diagrama de Actividades**

Las actividades consisten en una red de acciones que se relacionan entre sí mediante flujos de datos y de control. Mientras que el Diagrama de casos de uso muestra los requisitos del sistema, el Diagrama de actividades muestra cómo funcionan estos casos de uso. En este tipo de diagrama, por ejemplo, el token juega un papel importante: en los procesos paralelos, es un marcador para el cual se priorizan los procesos y, por lo tanto, se reciben los recursos (por ejemplo, la memoria de trabajo).

**Diagrama de Máquina de Estados**

Una máquina de estados, también llamada autómata finito, representa un conjunto finito de estados en un sistema. Si se cumple una condición definida en el sistema (se detona un desencadenante), se produce una situación correspondiente. Esto puede incluir actividades o interacciones. Bajo UML 2.0, un estado representa esta situación. Los estados se consideran como nodos (inglés: vértices) y se muestran como rectángulos con esquinas redondeadas. Además, el diagrama de máquina de estados modela las transiciones de un estado (nodo fuente) a otro (nodo destino). Los modelos UML presentan las transiciones de estado como bordes. Las acciones son la última piedra angular. Se asignan especificaciones de comportamiento al estado.

El comportamiento está ligado a ciertas situaciones o eventos. El diagrama de máquina de estados UML tiene 4 especificaciones:

Comportamiento de entrada (entry behavior)

Comportamiento en el estado (doActivity)

Comportamiento que ocurre en caso de un evento (event behavior)

Comportamiento de salida (exit behavior)

Además, un estado puede ser interrumpido y continuado en el mismo punto. Esta propiedad se llama estado de la historia.

**Diagramas de Interacción**

Los diagramas de interacción son un subtipo de los diagramas de comportamiento. Por lo tanto, también representan situaciones dinámicas. En particular, son adecuados para modelar el comportamiento en el que los elementos intercambian información. Los diagramas definen el papel de los objetos implicados. También nombran y priorizan los mensajes que se envían de un lado a otro entre los objetos. Los diagramas de interacción también muestran cómo estos mensajes afectan a los elementos del comportamiento. Por ejemplo, puede iniciar o detener actividades.

**Diagramas de Secuencia**

Como diagrama de interacción, el diagrama de secuencia representa el intercambio de mensajes entre objetos. El tipo de diagrama UML modela estos objetos como las llamadas líneas de vida. En este sentido, es similar a otros diagramas de comportamiento como el diagrama de actividad. Sin embargo, a diferencia de estos, el diagrama de secuencia no se utiliza para obtener una visión general del comportamiento de un sistema, sino para presentar un posible comportamiento entre muchos otros en detalle. Prescribe una cronología. Una línea discontinua representa el curso del tiempo.

UML 2.0 muestra mensajes síncronos (UML: flecha con la punta llena) y mensajes asíncronos (UML: flecha con la punta abierta). Los mensajes síncronos son aquellos que bloquean un canal hasta que reciben una respuesta del objeto de destino. Determinan las características de comportamiento en forma de operaciones sincrónicas. Los mensajes asincrónicos controlan el objeto fuente de llamada. Éstas incluyen tanto las operaciones asíncronas como las señales (paquetes de datos enviados entre acciones).

**Diagrama de Comunicación**

Al igual que el diagrama de secuencia, el diagrama de comunicación modela una transferencia de mensajes. Sin embargo, este diagrama UML no utiliza líneas discontinuas para la secuencia de tiempo, sino que numera las secuencias con números y letras. Estos llamados términos de secuencia se encuentran encima de una flecha con la punta apuntando hacia el receptor. Los números representan el orden en que se envían los mensajes, las letras representan el nivel jerárquico (como se muestra en la figura siguiente).

Diagrama

Descripción generada automáticamente

*Figura 5. Los mensajes asíncronos 1a y 1b se envían en paralelo. El objeto :B envía el mensaje con la etiqueta 1a.1 antes del mensaje 1a.2b. El objeto :D envía el mensaje 1a.2b.1 solo después de recibir el mensaje 1a.2b.*

**Diagrama de Tiempos**

El diagrama de tiempos permite mostrar el comportamiento de los sistemas en detalle en función de la secuenciación temporal. Los sistemas en tiempo real, por ejemplo, tienen que manejar ciertos procesos dentro de un cierto período de tiempo. UML 2.0 modela el diagrama de temporización como un diagrama bidimensional con un eje x y un eje y para representar mejor el plano temporal. Con este subformulario del diagrama de secuencia, los estados de los objetos se encuentran en el eje y y las secuencias de tiempo asignadas a ellos se ejecutan a lo largo del eje y.

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

*Figura 6. Diagrama de temporización de una lámpara LED con cambiador de color: las instancias de color se alternan y están activas durante el mismo intervalo de tiempo.*

**Diagrama de Interacción**

El nuevo diagrama de interacción añadido a UML 2.0 ayuda a mostrar un sistema muy complejo primero en un esquema aproximado, si un diagrama de interacción normal se vuelve demasiado confuso. Un diagrama de secuencia, por ejemplo, es adecuado para la visualización detallada. El diagrama UML es similar al diagrama de actividad con nodos. Representa los flujos de control entre interacciones. La diferencia con el diagrama de actividad es que un diagrama de interacción completo puede anidarse dentro de nodos que representan actividades. Estos nidos se pueden mostrar directamente en el diagrama (en línea) o se puede hacer referencia (palabra clave: ref, del inglés reference) al modelo, que se muestra en detalle en otra parte.

1. ¿Cuáles son los principios orientados a objetos?

**Clase**

Como unidad lingüística, las clases son un aspecto central de UML. Definen lo que constituye una clase y cómo las clases interactúan entre sí. Esta language unit tiene cuatro niveles, que van desde elementos simples hasta relaciones más complejas:

* Núcleo (describe elementos de la infraestructura UML 2.0 como paquetes, espacios de nombres, atributos, etc.)
* AssociationClasses (define clases de asociación)
* Interfaces (define las interfaces)
* Powertypes (clase cuyas instancias son subclases dentro de esta clase)

**Encapsulamiento**

El encapsulamiento es el principio que permite proteger los datos y ocultar la información interna de un objeto, de manera que solo se pueda acceder a ellos a través de métodos públicos y seguros.

Los datos y métodos privados de un objeto no están disponibles para otros objetos, lo que garantiza la integridad de la información y evita que sea alterada o corrompida accidental o intencionalmente.

Cada objeto es responsable de su propia información y de su propio estado, la única forma de que la información se pueda modificar, es mediante los mismos métodos del objeto. por lo tanto, los atributos internos de un objeto deberían ser inaccesibles desde fuera, pudiéndolos modificar solo llamando a las funciones correspondientes.

Con esto conseguimos mantener a salvo el estado de usos indebidos que puedan ocasionar resultados inesperados.

**Objetivo**

El objetivo principal de la encapsulación es proteger los datos de una clase del acceso directo desde el exterior, lo cual ayuda a mantener la integridad de los datos y a reducir la complejidad del sistema.

**Abstracción**

La abstracción es el principio que permite simplificar y clarificar los conceptos y comportamientos complejos mediante la creación de modelos o representaciones abstractas y simplificadas.

Los modelos abstractos se centran en los aspectos esenciales y relevantes del objeto o sistema, y eliminan los detalles irrelevantes o confusos. De esta manera, se simplifica el diseño y se mejora la comprensión y mantenibilidad del programa.

Este principio se puede definir como las características específicas del objeto, los mismos que lo distinguen de los demás tipos de objetos,

La abstracción consiste en separar un elemento de su contexto o del resto de elementos que lo acompañan. Es un principio por el cual se descarta toda aquella información que no resulta relevante en un contexto en particular, enfatizando en alguno de los detalles o propiedades de los objetos. Depende principalmente del interés del observador, permitiendo abstracciones muy diferentes de la misma realidad.

**Objetivo**

El objetivo de la abstracción es permitir que los programadores manejen la complejidad ocultando los detalles innecesarios y mostrando solo la información relevante.

**Herencia**

La herencia es el principio que permite crear nuevas clases a partir de clases existentes, reutilizando el código y los comportamientos de sus ancestros. La nueva clase se conoce como subclase o derivada, mientras que la clase original se llama superclase o base.

La herencia permite crear objetos especializados a partir de objetos más generales, y añadir o modificar sus propiedades y comportamientos de manera independiente. Además, reduce la duplicación de código y aumenta la eficiencia y la legibilidad del programa.

**Objetivo**

El objetivo principal de la herencia es promover la reutilización del código y establecer una relación jerárquica entre las clases.

**Polimorfismo**

En informática el polimorfismo es considerado uno de los elementos más importantes dentro la programación orientada a objetos POO, ya que su definición está fuertemente ligada a la Herencia.

El polimorfismo es el principio que permite a los objetos responder de diferentes maneras a un mismo mensaje o método, según su tipo o contexto. Es decir, un objeto puede comportarse de forma distinta según la situación, sin necesidad de conocer su tipo específico.

El polimorfismo aumenta la flexibilidad y la compatibilidad de los objetos, ya que permite interactuar con ellos de manera genérica y predecible, sin tener que conocer los detalles internos de su implementación. También permite extender y modificar las funcionalidades del programa de forma modular y escalable.

Como definición el polimorfismo es la habilidad de un objeto de realizar una acción de diferentes maneras, utilizando métodos iguales que se implementen de forma diferente en varias clases.

**Objetivo**

El objetivo del polimorfismo es permitir que una interfaz única controle el acceso a una clase genérica de acciones. Esto facilita la extensibilidad y la mantenibilidad del código.

**Implementación**:

El polimorfismo se puede lograr mediante la sobrecarga de métodos y la sobreescritura de métodos.

1. ¿Qué son los casos de uso y cómo se utiliza?

En ingeniería del software, un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Normalmente, en los casos de usos se evita el empleo de jergas técnicas, prefiriendo en su lugar un lenguaje más cercano al usuario final. En ocasiones, se utiliza a usuarios sin experiencia junto a los analistas para el desarrollo de casos de uso.

En otras palabras, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo, por ejemplo, la especialización y la generalización son relaciones. Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona a eventos que se producen en su ámbito o en él mismo.

Tabla

Descripción generada automáticamente con confianza baja

*Figura 7. Caso de ejemplo realizado por MADEJA*

Webgrafía

*Guía para la redacción de casos de uso*. (s/f). Juntadeandalucia.es. Recuperado el 16 de agosto de 2024, de <https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416>

Krall, C. (s/f). ¿Qué es y para qué sirve UML? Versiones de UML (Lenguaje Unificado de Modelado). Tipos de diagramas UML. Aprenderaprogramar.com; aprenderaprogramar.com. Recuperado el 16 de agosto de 2024, de <https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=688:ique-es-y-para-que-sirve-uml-versiones-de-uml-lenguaje-unificado-de-modelado-tipos-de-diagramas-uml&catid=46&Itemid=163>

*Los 4 principios fundamentales de la programación orientada a Objetos POO*. (2022, octubre 29). Diseño de Paginas Web Cusco. <https://webdesigncusco.com/los-4-principios-fundamentales-de-la-programacion-orientada-a-objetos-poo/>

*UML, lenguaje de modelado gráfico*. (2018, octubre 26). IONOS Digital Guide; IONOS. <https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/uml-lenguaje-unificado-de-modelado-orientado-a-objetos/?srsltid=AfmBOoqfXFA8pDP7BMrET8yGmj6jd5gJDGuiO4oxq0B9tNy_dYNrP_L8>

Rodriguez, G. J. (2012, enero 15). Técnicas efectivas para la toma de requerimientos. Northware. <https://www.northware.mx/blog/tecnicas-efectivas-para-la-toma-de-requerimientos/>

(S/f). Grupomontevideo.org. Recuperado el 16 de agosto de 2024, de <http://www.grupomontevideo.org/ndca/ndevaluacioninstitucional/wp-content/uploads/2014/09/5-Requerimientos.pdf>