**Parcial 2 IS**

**Tema 4: Introducción a la Ingeniería del Software.**

**Introducción**

Primera fase del ciclo de vida del software en la que se produce una especificación a partir de ideas informales. Deben obtenerse y documentarse:

* Los requisitos de información
* Los requisitos funcionales
* Los requisitos no funcionales
* Los criterios para medir el grado de su consecución

El proceso de desarrollo y especificación de requisitos se llama Ingeniería de Requisitos.

**Definición de requisitos**

Un requisito es una característica del sistema que satisface una necesidad o expresa una restricción que afecta al software desarrollado.

**¿Qué son los Requisitos?**

Describen los servicios que debe ofrecer el sistema y sus restricciones operativas. Un requisito puede ser desde una declaración abstracta de alto nivel hasta una especificación formal y detallada de una función del sistema. Debemos distinguirlas en:

* Requisitos del Sistema
* Requisitos de Usuario

La comunicación juega un papel fundamental en la ingeniería de requisitos, añade complejidad a ella ya que interviene el factor humano. Introduce aspectos sociales y culturales, no solo técnicos.

**¿Qué describe un requisito?**

* Una utilidad para el usuario
* Una propiedad general del sistema
* Una restricción general del sistema
* Cómo llevar a cabo cierto cálculo
* Una restricción sobre el desarrollo del sistema

**Requisitos vs Restricciones**

Dada una lista de deseos y necesidades, ¿cómo se distinguen los requisitos de las restricciones? Las restricciones limitan las posibles soluciones a un problema.

Lo que es tangible o visible para el usuario generalmente se considera un requisito.

**Relevancia de los requisitos**

Una definición inadecuada de los requisitos puede generar problemas. Requisitos ambiguos pueden ser interpretados de manera distinta por el usuario y el desarrollador.

Idealmente, los requisitos deberían ser:

* **Completos:** Recogen todos los servicios esperados por el usuario.
* **Consistentes:** No deben existir contradicciones entre las especificaciones.

Sin embargo, en la práctica, es casi imposible crear un documento de requisitos que sea completamente completo y consistente.

**Consejos para la definición de requisitos**

Se sugiere establecer un formato más o menos estándar y adherirse a él en los requisitos.

Asimismo, es fundamental utilizar el lenguaje de manera consistente:

* **Uso del presente** para los requisitos obligatorios, indica que estas características son imprescindibles para el sistema.
* **Uso del modo condicional** para los requisitos deseables, indica que estas características son útiles, pero no imprescindibles.

Además, se recomienda subrayar las partes relevantes del texto para destacar aspectos clave y facilitar la rápida identificación de información crucial.

Es esencial evitar la jerga técnica o términos que puedan no ser familiares para todos los interesados, esto puede llevar a malentendidos y falta de claridad en los requisitos.

**Tipos de Requisitos**

Los requisitos en un proyecto de desarrollo se dividen en: requisitos del sistema y requisitos del usuario..

**Requisitos del Sistema** (descripción detallada del sistema): Estos requisitos describen en detalle las funciones, servicios y restricciones operativas que debe cumplir el sistema. Son específicos y deben definir con exactitud lo que se implementará, ya que pueden formar parte del contrato entre el comprador y el desarrollador. Se dividen en:

* Los **Requisitos Funcionales** se refieren a las funciones y necesidades que el sistema debe satisfacer. Describen los cálculos que realiza el sistema, los datos que maneja y cómo los procesa.

Estos requisitos especifican cómo debe comportarse el sistema en respuesta a los estímulos externos.

* Los **Requisitos No Funcionales** no describen funciones específicas del sistema, imponen restricciones sobre los servicios o funciones que este ofrece.

No incluyen comportamiento. Su finalidad es especificar criterios que evalúen la calidad general del sistema. Suelen aplicarse al sistema en su conjunto y derivan de las necesidades del usuario. Estos requisitos definen propiedades y restricciones del sistema. También pueden referirse al proceso de desarrollo.

En general, son más críticos que los requisitos funcionales, ya que, si no se cumplen, el sistema puede no ser útil.

Los tipos de requisitos no funcionales incluyen:

* + **Requisitos del Producto:** Especifican el comportamiento del producto.
  + **Requisitos Organizacionales:** Derivan de políticas y procedimientos existentes en la organización del cliente y del desarrollador.
  + **Requisitos Externos:** Derivan de políticas y procedimientos existentes en la organización del cliente y del desarrollador

**Metas y Requisitos**

Cuando se trata de requisitos no funcionales, puede ser difícil definirlos con precisión. Los requisitos imprecisos resultan complicados de verificar.

* **Meta:** Un propósito general del usuario.
* **Requisito no funcional verificable:** Una declaración que incluye una medida objetiva.

Las metas son útiles para los desarrolladores, ya que reflejan los deseos generales del usuario.

**Requisitos del Usuario** (Descripción de alto nivel): Son descripciones, en lenguaje natural, tablas o diagramas, que detallan lo que se espera que el sistema proporcione, y sus restricciones. Estos requisitos deben incluir tanto requisitos funcionales como no funcionales y redactarse para ser comprensibles para los usuarios del sistema que no tienen conocimientos técnicos.

Problemas Comunes:

* **Falta de claridad:** Es difícil ser preciso sin que sea complicado de leer.
* **Mezcla de tipos de requisitos:** Se combinan requisitos funcionales y no funcionales, lo que puede generar confusión.
* **Fusión de requisitos:** Es habitual agrupar varios requisitos en la definición de uno solo, lo que puede dificultar la identificación de necesidades específicas.

Los **requisitos de interfaz** describen cómo la aplicación interactúa con su entorno, tanto en términos de comunicación con los usuarios como con otras aplicaciones o sistemas. Estos requisitos son cruciales para garantizar que la aplicación sea intuitiva, eficiente y capaz de integrarse con otros componentes del sistema.

**Gestión de Requisitos**

Tras la identificación de los requisitos, deben seguir un proceso estructurado que incluya las siguientes etapas:

* **Incorporación en un Catálogo:** Todos los requisitos deben ser documentados en un catálogo fácil de consultar. Este sirve como una base de datos centralizada que permite tener acceso a la información relevante.
* **Análisis Exhaustivo:** Los requisitos deben ser analizados para identificar posibles inconsistencias, ambigüedades, duplicidades o falta de información. Este análisis es esencial para asegurar que los requisitos son claros y evitar interpretaciones erróneas que afecten al desarrollo del sistema.
* **Validación con Clientes/Usuarios:** Los requisitos deben ser validados por los clientes o usuarios para asegurarse de que reflejan sus necesidades y expectativas. Esto garantiza que el producto cumpla con lo que se espera y que se minimicen los cambios posteriores en el desarrollo.

Además de las etapas anteriores, se deben realizar actividades que aseguren que los requisitos se están cumpliendo a lo largo del ciclo de vida del proyecto.

**Características de un Requisito**

Cada requisito debe poseer ciertas características que faciliten su gestión y seguimiento. Estas incluyen:

* **Identificador:** Un código único para cada requisito.
* **Autor:** La persona responsable de redactar o proponer el requisito.
* **Tipo de requisito:** Clasificación del requisito
* **Descripción:** Un texto claro y conciso que explique el requisito.
* **Prioridad:** Una clasificación de la importancia del requisito.
* **Estado:** La fase en la que se encuentra el requisito.
* **Fecha de Creación/Revisión:** Fechas que indican cuándo fue creado o revisado el requisito, lo que ayuda a mantener el control de versiones.

**Organización de un Requisito**

Los requisitos deben organizarse de manera lógica y estructurada. Algunas formas de organización incluyen:

* **Por Subsistemas:** Agrupar requisitos según los diferentes subsistemas que componen el sistema total.
* **Por Tipo:** Clasificar los requisitos en funcionales (RFX) y no funcionales (RNFX).
* **Jerárquicamente:** Establecer una estructura jerárquica que muestre la relación entre requisitos, como RF1, RF1.1, RF1.1.1, donde cada nivel representa una mayor especificidad.

**Actividades a Realizar en la Especificación de Requisitos**

Las actividades fundamentales involucradas en la especificación de requisitos son las siguientes:

1. **Extracción de Requisitos:** Proceso en el que los clientes descubren, revelan y comprenden los requisitos que desean.
2. **Análisis de Requisitos:** Proceso de razonamiento sobre los requisitos obtenidos, detectando y resolviendo posibles inconsistencias o conflictos.
3. **Especificación de Requisitos:** Proceso de redacción o registro de los requisitos. Para este proceso puede recurrirse al lenguaje natural, formal, etc.
4. **Validación de Requisitos:** Proceso de confirmación por parte de los usuarios/clientes de los requisitos.

La ejecución de estas actividades a menudo se apoya en diversas técnicas. Como los métodos de recolección de información como observación o técnicas.

Las siguientes técnicas son ampliamente empleadas para la captura y análisis de requisitos:

* **Entrevistas:** Conversaciones estructuradas con usuarios y partes interesadas para obtener información detallada.
* **Desarrollo conjunto de aplicaciones (JAD):** Un enfoque colaborativo que involucra a usuarios, desarrolladores y otros interesados en sesiones de trabajo conjunto.
* **Prototipado:** Creación de modelos iniciales del sistema para facilitar la comprensión de los requisitos y obtener retroalimentación temprana.
* **Observación:** Análisis del entorno en el que se utilizar á el sistema para identificar requisitos en función de cómo los usuarios interactúan con sus herramientas.
* **Estudio de documentación:** Revisión de documentos existentes relacionados con el sistema para identificar requisitos y necesidades previas.
* **Cuestionarios:** Herramientas de recopilación de datos que permiten obtener información de un amplio número de usuarios de manera estructurada.
* **Tormenta de ideas (Brainstorming):** Sesiones creativas en las que los participantes generan ideas y requisitos sin restricciones.
* **ETHIC (Effective Technical and Human Implementation of Computerbased System):** Un enfoque que se centra en la implementación efectiva tanto de aspectos técnicos como humanos en sistemas computacionales.

**Especificación de Requisitos del Software (ERS)**

El objetivo de la Especificación de Requisitos es generar un documento conocido como ERS. Este documento tiene la finalidad de definir de manera completa, precisa y verificable los requisitos que el sistema debe cumplir, incluyendo los requisitos funcionales, los no funcionales y las restricciones aplicables al diseño de software y hardware. El contenido de este documento incluye 4 grandes puntos:

1. Introducción
2. Descripción General
3. Requisitos Específicos
4. Apéndices

Características deseables en una **BUENA** especificación de requisitos:

* **No ambigua:** Los requisitos deben estar redactados de manera clara.
* **Completa:** Deben incluir todos los requisitos necesarios para el funcionamiento del sistema, sin omitir funcionalidades relevantes.
* **Fácil de verificar:** Es esencial que los requisitos puedan ser comprobados de forma sencilla para asegurar que se cumplen en el desarrollo del sistema.
* **Consistente (coherente):** La especificación debe ser interna y externamente coherente, sin contradicciones entre los requisitos.
* **Clasificada por importancia o estabilidad:** Los requisitos deben organizarse según su prioridad o la estabilidad de sus necesidades.
* **Fácil de modificar:** La especificación debe permitir cambios sin complicaciones, adaptándose a nuevas necesidades o correcciones de manera eficiente.
* **Fácil identificación del origen y las consecuencias de cada requisito**: Debe ser sencillo rastrear de dónde proviene cada requisito y cuáles son sus implicaciones en el sistema.
* **De fácil utilización durante la fase de explotación y mantenimiento:** La especificación debe ser práctica y accesible para su uso en etapas posteriores del ciclo de vida del sistema, como la explotación y el mantenimiento.

**Especificación de Requisitos Agiles**

Las **historias de usuario** son una herramienta clave en los métodos ágiles para describir de forma general e informal las expectativas del software desde la perspectiva del usuario final. No deben confundirse con los requisitos, las historias de usuario representan un objetivo o resultado deseado, en lugar de una función específica del sistema.

El propósito principal de las historias de usuario es definir cómo la función que se implementará generará valor para el usuario, proporcionando un contexto claro al equipo de desarrollo. Estas historias están estructuradas de manera narrativa.

Cada historia debe ser breve y fácil de recordar, lo que facilita su escritura en una tarjeta. Antes de su implementación, se complementan con conversaciones adicionales con los usuarios y la definición de criterios de validación que confirman si se han cumplido los objetivos planteados.

**Información en una Historia de Usuario**

Es recomendable evitar formatos rígidos al definir historias de usuario y, hay que adaptarlos a las características específicas de la empresa y del proyecto.

Hay 4 campos esenciales (se pueden añadir más):

* **ID:** Identificador único de la historia de usuario, correspondiente a una tarea.
* **Descripción:** Debe responder a tres preguntas clave: ¿quién se beneficia?, ¿qué se desea lograr? y ¿cuál es el beneficio esperado? Se recomienda seguir el siguiente patrón: “Como [rol del usuario], quiero [objetivo], para poder [beneficio].”
* **Estimación:** Indica el esfuerzo necesario en términos de tiempo ideal para implementar la historia de usuario.
* **Prioridad:** Este campo permite establecer el orden en que deben ser implementadas las historias de usuario.

Dependiendo del tipo de proyecto, la dinámica del equipo y la organización, pueden ser útiles otros campos, como:

* Título
* Criterio de Validación: pruebas de aceptación acordadas con el cliente o usuario
* Valor: normalmente numérico que junto con otros elementos, ayuda en la priorización de las historias.
* Dependencias

Un **epic** es una superhistoria de usuario caracterizada por su gran tamaño y alta granularidad, en contraste con las historias de usuario que tienen una baja granularidad. Esta etiqueta se aplica a historias que requieren un esfuerzo significativo y que no pueden completarse en una sola vez o dentro de un solo sprint. Suelen tener un flujo asociado que permite su descomposición en historias de usuario; es decir, las historias resultantes están estrechamente relacionadas entre sí. A medida que un epic aumenta en prioridad y se aproxima a su implementación, el equipo lo descompone en historias de usuario de un tamaño más manejable, facilitando la aplicación de los principios y técnicas ágiles, como la estimación y el seguimiento cercano (normalmente diario).

Por encima de los epics se encuentran los **temas**, que representan una colección de epics y/o historias de usuario relacionadas. Los temas sirven para describir un sistema o subsistema en su totalidad.

Por debajo de las historias de usuario están las **tareas**. Estas son el resultado de la descomposición de las historias de usuario en unidades de trabajo más pequeñas y adecuadas para gestionar y seguir el avance de su ejecución.

**Pila del Producto en Scrum**

En Scrum y en metodologías ágiles en general, la pila del producto puede contener tanto historias de usuario como epics. Esta pila debe estar ordenada por prioridad, y el nivel de detalle de cada elemento debe corresponder a su posición en la lista.

Para elementos de baja prioridad que se encuentran al final de la lista, no tiene sentido mantener un alto nivel de detalle, ya que es probable que cambien a lo largo del proyecto o incluso que no se desarrollen en absoluto.

A medida que nos acercamos a los elementos más prioritarios, el detalle debe aumentar. Por lo tanto, las historias de usuario deben encabezar la lista, mientras que los epics ocuparán posiciones más bajas.

**Tema 5: Técnicas de Especificación y Modelado**

**Introducción**

**El Rol de los Modelos en la Ingeniería del Software**

“Los modelos son los planos del software”, una representación abstracta que permite visualizar y razonar sobre un sistema antes de proceder a su implementación en código. Ofrecen una visión estructurada del software, ayudando a los desarrolladores a comprender cómo funcionará el sistema antes de construirlo, reduciendo riesgos y mejora la planificación. Además, permite evaluar decisiones de diseño sin escribir código

Entre las ventajas de los modelos es que describen aspectos clave, su **estructura** define los componentes y su interrelación; su **comportamiento**, que explica cómo estos interactúan y reaccionan ante diferentes estímulos; y su **despliegue** detalla cómo el sistema se distribuirá y ejecutará en diferentes entornos.

Además, los modelos facilitan la **abstracción** del sistema completo, permitiendo a los desarrolladores y diseñadores manejar la complejidad de los proyectos sin detalles innecesarios. Al simplificar el sistema en conceptos clave, se mejora la comunicación entre los miembros del equipo, y se reduce la ambigüedad y los malentendidos que suelen surgir en la fase de diseño y desarrollo.

Estos modelos son útiles en todas las etapas del desarrollo del software.

Otra de sus grandes ventajas es que pueden ser interpretados de forma ambigua, los modelos mejoran la precisión y claridad en la representación del sistema. Cumplen con los criterios establecidos por Stachowiak, quien establece que los modelos deben:

* Representar un objeto o fenómeno real.
* Actuar como un “espejo” parcial de la realidad, capturando los aspectos más relevantes para el propósito en cuestión.
* Reemplazar al original para cumplir alguna función específica, como análisis, simulación o predicción del comportamiento del sistema.

**Ingeniería de Sistemas Basada en Modelos (MBSE)**

Es un enfoque formal que utiliza modelos como principal herramienta para el desarrollo de sistemas complejos. Emplea modelos que representan diferentes aspectos del sistema a lo largo de todo su ciclo de vida. Incluye desde el análisis de los requisitos iniciales hasta la implementación y operación del sistema en funcionamiento.

Uno de los pilares de la MBSE es que los modelos se convierten en el principal medio de intercambio de información entre los diferentes equipos involucrados en el desarrollo del sistema. Esta estrategia permite mejorar la coherencia, precisión y comprensión del sistema al eliminar posibles ambigüedades que podrían surgir en la documentación tradicional. Los modelos actúan como una fuente de información, asegurando que los involucrados en el proyecto estén trabajando con una representación compartida y actualizada del sistema.

**Ingeniería del Software Guiada por Modelos (MDE)**

Es una metodología de desarrollo de software que se basa en la creación de modelos que representan el dominio específico de la aplicación. Está enfocada en la abstracción del conocimiento y las actividades del dominio, es decir, capturar los conceptos y procesos clave de una manera que sea comprensible y manipulable a través de modelos formales.

Una de las ventajas principales de la MDE es que aumenta la productividad al permitir una mayor automatización en la generación de software, reduciendo la cantidad de código a escribir. Fomenta la reutilización, ya que los modelos pueden ser adaptados y extendidos para futuros proyectos, mejorando la eficiencia del desarrollo a largo plazo. Además simplifica el diseño del software centrándose en una representación de más alto nivel, donde los detalles técnicos quedan encapsulados dentro del modelo.

Un aspecto clave es la capacidad de utilizar los modelos no solo para diseñar el sistema, sino también para generar código automáticamente, desarrollo dirigido por modelos (Model-Driven Development, MDD). Esta capacidad permite que los modelos se traduzcan en implementaciones, acortando el ciclo de desarrollo y asegurando que el código esté alineado con el diseño del sistema. Además, los modelos pueden utilizarse para probar el sistema, en pruebas basadas en modelos (Model-Based Testing, MBT), facilitando la detección de errores y la validación del comportamiento del sistema desde el inicio del desarrollo.

**Técnicas Orientadas a Objeto: Lenguaje Unificado de Modelado**

A mediados de los años noventa, el desarrollo de software orientado a objetos estaba marcado por la existencia de múltiples métodos de análisis y diseño, cada uno con su propia notación y enfoque. Aunque compartían muchos de los mismos conceptos, las diferencias en notación y metodología generaban confusión y fragmentación en la industria. Este fenómeno fue conocido como la “guerra de los métodos”, ya que cada propuesta competía por ser la dominante, lo que dificultaba la comunicación y el avance en el desarrollo de software.

En 1994, tres de los principales autores de estos métodos decidieron unificar sus enfoques, lo que dio lugar a la creación del **Unified Modeling Language (UML)**. Este lenguaje unificado buscaba proporcionar una notación estándar para modelar sistemas orientados a objetos, abordando tanto el análisis como el diseño. Su estandarización fue promovida por el Object Management Group (OMG), consolidando su posición como el estándar de facto en el modelado de software.

Las raíces técnicas de UML provienen de la combinación de los tres enfoques:

* **Object Modeling Technique (OMT) de Rumbaugh:** centrada en el análisis de datos, especialmente en sistemas de información.
* **Método-Booch:** era ideal para sistemas concurrentes y de tiempo real.
* **Object-Oriented Software Engineering (OOSE) de Jacobson:** introdujo el concepto de desarrollo basado en casos de uso.

La creación de UML unificó estos enfoques, combinando sus fortalezas y mejorando su capacidad de modelado. De esta manera, no solo estandarizó la notación, sino que también permitió unificar los procesos de análisis y diseño de sistemas orientados a objetos.

**Evolución Histórica de UML**

* En 1994 Rumbaugh y Booch crean el Método Unificado.
* En 1995 se incorpora Jacobson y los tres autores publican Unified Method v0.8.
* El Método Unificado se reorienta hacia la definición de un lenguaje universal para el modelado de objetos, transformándose en UML
* En 1996 se crea un consorcio de colaboradores para trabajar en la versión 1.0.
* En 1997 se produce la estandarización de UML 1.0 por la OMG.
* En julio de 1998 aparece una revisión interna de UML.
* En junio de 1999 aparece OMG UML 1.3 con cambios significativos.
* En septiembre de 2001 aparece UML 1.4 y en enero de 2005 OMG UML 1.4.2.
* En julio de 2005 se libera UML 2.0 y en 2017 la última versión 2.5.1

**Ventajas de la Unificación**

La unificación de los métodos orientados a objetos en UML ofreció diversas ventajas. Reunió los puntos fuertes de cada, permitiendo combinar enfoques estructurados, concurrentes y basados en requisitos. Además, facilitó la introducción de nuevas mejoras, sentando las bases para futuras innovaciones en el modelado de software. Esta unificación proporcionó estabilidad al mercado y ofreciendo un lenguaje maduro y ampliamente aceptado. También impulsó la creación de potentes herramientas de modelado y redujo la confusión entre usuarios, promoviendo una notación estándar que mejoró la colaboración.

**Objetivos en el Diseño de UML**

El primero fue modelar sistemas desde los requisitos hasta los artefactos ejecutables mediante técnicas orientadas a objetos, abarcando todo el ciclo de vida del software. Se planteó abordar la complejidad de sistemas grandes y críticos, haciéndolo lo suficientemente flexible para grandes proyectos. UML fue diseñado para ser útil tanto para personas como para máquinas, facilitando comunicación y automatización. Finalmente, se buscó equilibrar expresividad y simplicidad, logrando un lenguaje potente sin ser complicado, lo que favoreció su adopción masiva.

**Definición de UML**

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos (modelos) de un sistema que involucra una gran cantidad de software, desde una perspectiva orientada a objetos (OO).

Su principal función es proporcionar una notación para modelar diferentes aspectos del sistema, pero no es un proceso. Sirve como una herramienta para representar de manera clara y consistente los componentes y relaciones en el sistema, facilitando la comprensión y comunicación entre los actores del desarrollo.

Se han definido muchos procesos que utilizan UML como base para el desarrollo de software. Aunque inicialmente fue creado para proyectos de software, UML y RUP son utilizables en sistemas más allá del software, incluyendo áreas como ingeniería de sistemas y otras disciplinas donde es necesario modelar sistemas.

**El Modelo Conceptual de UML**

**Diagrama

Descripción generada automáticamente**Para comprender UML, es fundamental familiarizarse con sus elementos clave, que incluyen los **bloques básicos** compuestos por elementos, relaciones y diagramas. Es importante tener en cuenta las **reglas** de combinación, que dictan cómo ensamblar los bloques de manera coherente y significativa, y los **mecanismos comunes** que se aplican en todo el lenguaje UML, garantizando la consistencia y la interoperabilidad entre diferentes modelos y diagramas.

**Bloques básicos de construcción**

UML se compone de tres bloques de construcción:

* **Elementos**: bloques básicos de la programación orientada a objetos, abstracciones de primera clase dentro de un modelo.

Tipos: Elementos estructurales, de comportamiento, de agrupación y de anotación

* **Relaciones**: Conectan los diferentes elementos entre sí
* **Diagramas**: Representación gráfica de un conjunto de elementos y sus relaciones

**Elementos estructurales**

También llamados clasificadores, son los nombres de los modelos UML.

* En su mayoría son las partes estáticas de un modelo.
* Representan conceptos o cosas materiales.
* Elementos que representan cosas conceptuales o lógicas: clase, interfaz, colaboración, caso de uso, clase activa y componente.
* Elementos que representan elementos físicos: artefactos y nodos.

**Clase**

Es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Actúa como un clasificador que define la estructura y el comportamiento de dichos objetos. Una clase puede implementar una o más interfaces.

Gráficamente se representa como un rectángulo que contiene su nombre, atributos y operaciones

**Caso de uso**

Es una descripción de un conjunto de secuencias de acciones que ejecuta un sistema y producen un resultado observable de interés para un actor particular. Se utiliza para estructurar los aspectos de comportamiento en un modelo y es realizado por una colaboración. Gráficamente se representa como una elipse de borde continuo, y contiene solo su nombre.

**Interfaz**

Es un conjunto de operaciones que especifican un servicio de una clase o componente. Una interfaz describe el comportamiento visible externamente de dicho elemento. Puede representar el comportamiento completo de una clase o componente, o solo una parte. Una interfaz define un conjunto de especificaciones de operaciones, pero no incluye la implementación de dichas operaciones.

La declaración de una interfaz es similar a la de una clase, pero usa la palabra clave <<interface>> antes del nombre. Los atributos no son relevantes en una interfaz, salvo en ocasiones, para definir constantes. Normalmente, una interfaz no se encuentra aislada. Cuando una clase expone una interfaz al mundo exterior, se representa mediante un pequeño círculo conectado al rectángulo que simboliza la clase mediante una línea. Si una clase requiere una interfaz que será proporcionada por otra, dicha interfaz se representa como un semicírculo unido al rectángulo de la primera clase por una línea.

**Componente**

Es una unidad modular del diseño de un sistema que oculta su implementación detrás de un conjunto de interfaces externas definidas, que se exponen a través de puertos. Los componentes que comparten las mismas interfaces pueden ser sustituidos en un sistema siempre que mantengan el mismo comportamiento lógico. Su interior permanece oculto y solo es accesible mediante las interfaces que proporciona. Un componente puede indicar dependencias a través de interfaces requeridas.

La implementación de un componente puede describirse mediante la conexión de partes y conectores, donde las partes pueden incluir componentes más pequeños. Gráficamente, un componente se representa como una clase con un icono especial en la esquina superior derecha.

**Artefacto**

Un artefacto es una parte física y reemplazable de un sistema que contiene información física (“bits”). En un sistema hay diferentes tipos de artefactos de despliegue. Representa típicamente el empaquetamiento físico de código fuente o información en tiempo de ejecución. Gráficamente se representa como un rectángulo con la palabra clave <<artifact>> sobre el nombre.

**Nodo**

Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, generalmente con memoria y capacidad de procesamiento. Pueden alojar un conjunto de artefactos, los cuales pueden migrar de un nodo a otro. También pueden estar compuestos por otros nodos internamente.

Los nodos representan dispositivos de hardware o entornos de software de ejecución, sirviendo como el recurso computacional sobre el cual se despliegan los artefactos para su funcionamiento. Gráficamente, un nodo se representa como un cubo, usualmente con solo su nombre.

**Elementos de comportamiento**

Los elementos de comportamiento son las partes dinámicas en los modelos UML. Actúan como los “verbos” del modelo y representan el comportamiento a lo largo del tiempo y espacio. Semánticamente, estos elementos están vinculados a elementos estructurales.

**Mensaje**

Una interacción es un comportamiento compuesto por mensajes intercambiados entre objetos en un contexto específico, con el fin de lograr un objetivo. Tanto el comportamiento de un conjunto de objetos como el de una operación individual pueden especificarse mediante una interacción, que incluye elementos como mensajes, acciones y enlaces.

Gráficamente, un mensaje se representa como una línea con dirección, generalmente acompañada del nombre de la operación asociada. Los mensajes actúan como el medio de comunicación para intercambiar información o expresar interacciones entre elementos, y pueden variar en tipo y propiedades, lo cual afecta ligeramente su notación.

**Estado**

Una máquina de estados modela el comportamiento de un objeto o interacción a través de las secuencias de estados que atraviesa durante su vida y sus reacciones ante estos. Puede describir el comportamiento de una clase o la colaboración entre varias.

La máquina de estados se compone de elementos como estados, transiciones, eventos y actividades. Gráficamente, un estado se representa como un rectángulo con esquinas redondeadas que incluye su nombre y, si los tiene, sus subestados.

**Acción**

Una actividad es un comportamiento que especifica la secuencia de pasos en un proceso computacional. Una actividad se enfoca en los flujos entre pasos, sin considerar qué objeto ejecuta cada uno.

Cada paso dentro de una actividad se llama acción. En UML, las acciones son unidades fundamentales de comportamiento, generalmente con un conjunto de entradas y salidas que modifican el estado del sistema. Gráficamente, una acción se representa como un rectángulo con esquinas redondeadas, identificado por un nombre que indica su propósito.

**Elementos de agrupación**

Un paquete es un mecanismo general para organizar y estructurar el diseño en UML, en contraste con las clases, que estructuran la implementación. Puede incluir elementos estructurales, de comportamiento e incluso otros paquetes. Un paquete es conceptual y solo existe durante el desarrollo.

Los paquetes actúan como “espacios de nombres” que delimitan el alcance de sus elementos. Además, se pueden establecer relaciones entre ellos. Gráficamente, un paquete se representa como una carpeta, que suele mostrar su nombre y, a veces, su contenido.

**Elementos de anotación**

Los elementos de anotación son componentes explicativos en los modelos UML, usados para describir, aclarar o hacer observaciones sobre cualquier elemento. El principal tipo de elemento de anotación es la nota, un símbolo que muestra restricciones y comentarios relacionados con un elemento o grupo de elementos. Sirven únicamente para aclarar y no afectan la semántica del modelo.

Gráficamente, una nota se representa como un rectángulo con una esquina doblada y contiene texto o gráficos explicativos.

**Relaciones**

Las relaciones en UML permiten modelar los enlaces entre diferentes elementos estructurales del sistema, proporcionando una manera de representar sus interacciones y asociaciones. Estas relaciones también aportan información adicional, que resulta clave para comprender cómo se vinculan y operan los distintos elementos del modelo.

Una de las principales características que se puede especificar en una relación es la multiplicidad. Que indica la cantidad de instancias de una clase que pueden estar asociadas con instancias de otra. La multiplicidad se expresa mediante notaciones específicas, tales como:

* ‘1’: única instancia.
* ‘0..1’: cero y una instancia.
* ‘0..\*’: entre cero y muchas instancias.
* ‘1..\*’: al menos una instancia.
* ‘\*’: un número indefinido de instancias.

Otra característica importante son los nombres de roles, identifican los extremos de una asociación para entender el papel específico de cada elemento en la relación.

En UML, se manejan cuatro clases principales de relaciones: dependencia, asociación, generalización y realización. Cada una de estas representa un tipo de conexión particular entre elementos, permitiendo definir de manera precisa cómo interactúan y dependen entre sí dentro del sistema modelado.

**Relación de Dependencia**

Es una relación semántica entre dos elementos, en la cual un cambio a un elemento (independiente) puede afectar a la semántica del otro elemento (dependiente). Gráficamente, una dependencia se representa como una línea discontinua, posiblemente dirigida, que incluye a veces una etiqueta.

**Relación de Asociación**

Es una relación estructural entre clases que describe un conjunto de enlaces, los cuales son conexiones entre objetos que son instancias de clases. Gráficamente, una asociación se representa como una línea continua, que a veces incluye una etiqueta.

**Relación de Agregación**

Es un tipo especial de asociación en UML que representa una relación estructural entre un todo y sus partes. Modela una conexión donde una clase está compuesta por una o más instancias de otra clase. Las partes pueden existir independientemente del todo.

Gráficamente, la agregación se representa como una línea de asociación con un rombo vacío en el extremo que indica el “todo” (la clase contenedora).

**Relación de Composición**

Es un caso más fuerte de agregación, donde también se representa una relación “todo-parte”, pero con una dependencia de vida más estricta entre las partes y el todo. En una composición, las partes no pueden existir independientemente del todo; si el objeto contenedor se destruye, sus partes también dejan de existir.

Gráficamente, la composición se representa con una línea de asociación que tiene un rombo sólido en el extremo del “todo”

**Relación de Generalización**

Una generalización es una relación de especialización/generalización en la cual el elemento especializado (el hijo) se basa en la especificación del elemento generalizado (el padre). El hijo comparte la estructura y el comportamiento del padre.

Gráficamente, una relación de generalización se representa como una línea continua como una punta de flecha vacía apuntando al padre.

**Relación de Realización**

Una realización es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización en dos contextos: entre interfaces y las clases o componentes que las realizan, y entre los casos de uso y las colaboraciones que los ejecutan.

Gráficamente, una relación de realización se representa como una mezcla entre una generalización y una relación de dependencia.

**Diagramas**

Un diagrama es una representación gráfica que muestra un conjunto de elementos conectados, permitiendo visualizar sus relaciones y la estructura general del sistema. Estos forman grafos conectados en los que los vértices representan los elementos del sistema y los arcos muestran las relaciones entre ellos. Mediante esta estructura, los diagramas permiten visualizar el sistema desde diferentes perspectivas, lo que facilita la comprensión de su arquitectura y su funcionamiento.

En la práctica, un mismo elemento puede aparecer en varios diagramas si es relevante en diferentes contextos. Los diagramas se agrupan por:

* **Diagramas Estructurales:** visualizan, especifican, construyen y documentan los aspectos estáticos de un sistema. Estos representan la organización y disposición de los elementos que conforman la estructura del sistema proporcionando una vista detallada de la arquitectura subyacente.
* **Diagramas de Comportamiento:** están enfocados en los aspectos dinámicos del sistema. Su objetivo es visualizar, especificar, construir y documentar el comportamiento del sistema en tiempo real. Así, cada tipo de diagrama cumple un rol específico en la representación y el análisis de diferentes facetas del sistema modelado.

**Reglas de combinación**

Los bloques de construcción de UML no se pueden combinar de manera específica. Un **modelo bien formado** es aquel que es semánticamente autoconsistente y está en armonía con sus modelos relacionados.

Existen reglas semánticas que especifican a qué debe parecerse un buen modelo:

* **Nombres:** Especifican cómo llamar a los elementos, relaciones y diagramas.
* **Alcance:** Define el contexto que da un significado específico a un nombre.
* **Visibilidad:** Indica cómo se pueden ver y utilizar esos nombres por otros.
* **Integridad:** Asegura que los elementos se relacionen de manera apropiada y consistente.
* **Ejecución:** Establece lo que significa ejecutar o simular un modelo.

Según Booch [3], es habitual que el equipo de desarrollo no sólo construya modelos bien formados sino que los modelos sean:

* **Abreviados:** algunos elementos se ocultan para simplificar la vista.
* **Incompletos:** pueden estar ausentes ciertos elementos.
* **Inconsistentes:** no se garantiza la integridad del modelo.

Estos modelos que no llegan a ser bien formados son inevitables según aparecen detalles del sistema y se avanza en el desarrollo. **Las reglas de UML favorecen (pero no obligan)** a considerar las cuestiones más importantes de análisis, diseño e implementación que permiten obtener sistemas bien formados con el paso del tiempo.

**Mecanismos comunes**

UML incorpora ciertos mecanismos comunes que se pueden aplicar en distintos modelos para mejorar su interpretación y claridad:

1. **Especificaciones:** Proporcionan una base semántica que incluye a todos los modelos de un sistema y aseguran la coherencia entre elementos. Se utilizan para detallar las características del sistema; cada elemento gráfico tiene una explicación textual que describe su sintaxis y semántica.
2. **Adornos:** Son notaciones gráficas que añaden detalles adicionales para aclarar o complementar la información de los elementos UML.
3. **Divisiones comunes:** Permiten modelar tanto desde una perceptiva general (abstracción) como particular (concreto). Esta opción está disponible en la mayoría de los bloques de construcción.
4. **Extensibilidad:** Aunque UML proporciona un lenguaje estándar para el modelado de sistemas, admite adaptaciones para cubrir necesidades específicas de cada proyecto. Los mecanismos de extensibilidad incluyen:
   * **Estereotipos:** Crean nuevos tipos de bloques de construcción derivando los existentes.
   * **Valores etiquetados:** Añaden nueva información a las propiedades de los elementos.
   * **Restricciones:** Extienden la semántica de un bloque de construcción para añadir o modificar reglas.

**Arquitectura**

La visualización, especificación, construcción y documentación de sistemas complejos requieren múltiples perspectivas. Las personas involucradas en el desarrollo tienen intereses variados y observan el sistema desde distintas perspectivas a lo largo de su ciclo de vida. La arquitectura de un sistema se describe óptimamente a través de cinco vistas interrelacionadas, cada una enfocada en un aspecto particular del sistema.

**Vista de Casos de Uso**

Describe el comportamiento del sistema desde la perspectiva de los usuarios finales, analistas y responsables de pruebas. Utiliza casos de uso para representar aspectos estáticos y diagramas de interacción, de estados y de actividades dinámicas.

**Vista de Diseño**

Incluye las clases, interfaces y colaboraciones que definen el vocabulario del problema y su solución, enfocándose principalmente en los requisitos funcionales del sistema. Para los aspectos estáticos, utiliza diagramas de clases y de objetos; para los aspectos dinámicos, se apoya en diagramas de interacción, de estados y de actividades.

**Vista de Procesos**

Esta vista abarca los hilos y procesos que gestionan la sincronización y la concurrencia, atendiendo al crecimiento, rendimiento y operación del sistema. Utiliza los mismos diagramas que la vista de diseño, pero centrándose en clases activas que representan hilos y procesos.

**Vista de Implementación**

Describe los componentes y archivos necesarios para ensamblar y ejecutar el sistema físico, con un enfoque en la gestión de versiones de componentes independientes que pueden combinarse de distintas maneras. Utiliza diagramas de paquetes para los aspectos estáticos, y diagramas de interacción, de estados y de actividades para los aspectos dinámicos.

**Vista de Despliegue**

Muestra los nodos que componen la topología de hardware del sistema, centrándose en la distribución, entrega e instalación de los elementos del sistema físico. Para los aspectos estáticos, utiliza diagramas de despliegue, y para los aspectos dinámicos, diagramas de interacción, de estados y de actividades.

**Cada una de estas vistas puede existir de forma independiente y también interactuar entre sí**

**Diagrama de Casos de Uso**

Un **caso de uso** es una descripción de cómo un sistema lleva a cabo una serie de acciones para generar un resultado valioso para un actor. En ellos capturan el comportamiento esperado del sistema de manera que sea comprensible para todos sin especificar detalles técnicos de implementación.

Ayudan a validar la arquitectura y verificar el sistema a medida que evoluciona, asegurando que los requisitos se cumplan en cada etapa. Deben enfocarse en los comportamientos esenciales, evitando descripciones amplias o específicas.

Se pueden utilizar como base para establecer casos de prueba en el desarrollo, son:

* **Casos de uso aplicados a subsistemas:** permiten verificar que las funcionalidades existentes no sean afectadas por nuevos cambios.
* **Casos de uso aplicados al sistema completo:** permiten validar que todas las partes del sistema trabajan correctamente en conjunto.

**Construcción del Modelo de Casos de Uso**

Sigue unos pasos que permiten estructurar y definir el sistema y sus interacciones de manera efectiva:

1. **Establecer el límite del sistema:** Identificar el alcance del sistema y los límites que lo separan de otros sistemas o del entorno externo.
2. **Definir los actores:** Determinar todos los usuarios o sistemas externos que interactuarán con el sistema.
3. **Identificar los casos de uso:**
   * Especificar el caso de uso: Describir detalladamente cada caso de uso para reflejar el comportamiento esperado del sistema
   * Incluir flujos alternativos clave: Identificar posibles variaciones en el flujo de trabajo principal, como excepciones o alternativas.
4. **Repetir los pasos** anteriores hasta que el sistema, los actores y los casos de uso sean estables y reflejen todos los requerimientos identificados.

**Componentes del modelo**

* **Límite del Sistema:** Define el alcance del sistema que se está modelando, delimitado visualmente mediante un rectángulo.
* **Actores:** Son los roles de personas o sistemas externos que interactúan con el sistema. Características:
  + Una misma persona puede adoptar distintos roles, según el contexto.
  + Los **actores principales** activan los casos de uso, los **secundarios** solo interactúan sin activarlos.
  + El nombre del actor debe reflejar claramente su papel en la interacción con el sistema.
  + Se representan comúnmente como un “muñeco de palo” para mayor claridad en el diagrama.
  + Los actores pueden ser especializados mediante relaciones de generalización, permitiendo agrupar roles similares.
* **Casos de Uso:** Los casos de uso representan las acciones o funcionalidades que los actores pueden realizar con el sistema. Se representan como elipses.
* **Relaciones:** Las relaciones conectan a los actores con los casos de uso, mostrando el tipo de comunicación o interacción que existe entre ellos.

Un **diagrama de casos de uso** es una representación gráfica que ilustra un conjunto de casos de uso, actores, y sus relaciones. Puede incluir notas y restricciones para aclarar detalles específicos.

**Especificación de los casos de uso**

La especificación de un caso de uso describe su comportamiento a través de un flujo de eventos claro y comprensible para cualquier persona, incluso aquellas sin conocimientos profundos del sistema.

En este flujo de eventos se debe incluir los siguientes elementos:

* **Inicio y fin del caso de uso:** Indica cuándo comienza y finaliza la interacción.
* **Interacción con actores:** Especifica cómo interactúan los actores con el sistema.
* **Objetos intercambiados:** Detalla los datos o elementos que se intercambian entre el sistema y los actores.

Además, el flujo de eventos debe contener tanto el **flujo principal** (el curso normal de acciones) como los **flujos alternativos** que representan variaciones en el comportamiento.

Los casos de uso se identifican analizando las secuencias de interacción con los actores desde la perspectiva del usuario final.

Aunque no existe un formato estandarizado para documentar los casos de uso, es común describir la especificación en formato tabular con los siguientes campos:

* **Nombre** del caso de uso
* **ID** del caso de uso
* **Objetivo:** una breve descripción del propósito del caso de uso.
* **Contexto o precondiciones:** las condiciones previas necesarias.
* **Actores implicados:** especificando el actor principal y, si es necesario, los secundarios.
* **Flujo principal:** el escenario principal de los eventos.
* **Flujos alternativos:** extensiones del flujo principal.
* **Postcondiciones:** resultados que se esperan una vez concluido el caso de uso.

Para estructurar un caso de uso, es importante analizar cada paso y diferenciar el flujo principal de los alternativos.

* **Flujo principal:** Representa el escenario ideal en el que el caso de uso se ejecuta sin errores, desvíos o interrupciones. Puede tener desviaciones:
  + Simples: Ramificaciones que ocurren dentro del flujo principal.
  + Complejas: Escenarios alternativos.
* **Flujos alternativos:** Se enfocan en la gestión de errores y excepciones, y generalmente no regresan al flujo principal. Para una especificación clara:
  + Seleccionar solo los flujos alternativos más relevantes.
  + Agrupar los flujos similares para evitar redundancias y simplificar.

**Tipos de relaciones en UML**

Los diagramas de casos de uso en UML representan distintas relaciones entre actores y casos de uso que ayudan a organizar y simplificar el modelo. Relaciones principales:

1. **Comunicación**

* Descripción: Relación básica entre los actores y el sistema, que define las entradas y salidas de información en el diagrama.
* Uso: Especifica cómo los actores se comunican con el sistema.

1. **Herencia**

* **Entre Actores:** Permite abstraer similitudes en la interacción de actores-sistema.
* **Entre Casos de Uso:** Los casos de uso hijos pueden heredar, modificar o añadir características al caso de uso padre.
  + En UML 2, la herencia entre casos de uso afecta solo a relaciones, precondiciones, postcondiciones y flujos, estos no tienen atributos ni operaciones.
  + **Recomendación:** Usar herencia únicamente cuando simplifique el modelo, puede facilitar comprensión y mantenimiento.

1. **Inclusión**

* **Descripción:** Un caso de uso base incluye el comportamiento de otro caso de uso en un punto específico, fomentando la reutilización.
* **Evita duplicación:** Facilita la reutilización de flujos de eventos comunes mediante un caso de uso independiente.
* **Representación:** La relación de inclusión se denota con la etiqueta <<include>>.
* El caso de uso incluido siempre debe formar parte de otro caso de uso base.
* Debe figurar en la especificación de todos los casos de uso que lo incluyan.

1. **Extensión**

* **Descripción:** Un caso de uso origen añade un comportamiento opcional al caso de uso destino, permitiendo modelar subflujos o características opcionales.
* **Independencia:** A diferencia de la inclusión, el caso de uso destino es funcional por sí mismo.
* **Condiciones:** Útil para representar comportamientos que se activan solo bajo ciertas condiciones.
* **Representación:** La relación de extensión se indica con la etiqueta <<extend>>.
* **Especificación:** La extensión debe mencionarse en la especificación de los casos de uso afectados, indicando los puntos donde se extiende el flujo principal.

**Trazabilidad**

El modelo de casos de uso está estrechamente relacionado con la especificación de requisitos, aspecto clave en la ingeniería de requisitos.

Es fundamental vincular los requisitos funcionales con los casos de uso para que:

* Cada **requisito funcional** está cubierto por al menos un caso de uso
* Cada **caso de uso** esté asociado a al menos un requisito funcional

Algunas herramientas de modelado o de gestión de requisitos permiten establecer vínculos automáticos entre los casos de uso y los requisitos.

Como alternativa, se puede crear una **matriz de trazabilidad**, con beneficios como:

* Detección de inconsistencias, como requisitos sin casos de uso o viceversa.
* Facilitación del seguimiento de los requisitos durante el proyecto.

Para utilizar matrices de trazabilidad de manera efectiva, es importante que tanto los requisitos como los casos de uso tengan un ID único.

**Diagrama de Actividades**

Modela un proceso como una secuencia de nodos conectados, como un diagrama de flujo, pero orientado a objetos. Es una herramienta versátil en el modelado de sistemas que ofrece una representación visual clara de distintos tipos de flujos de trabajo.

Principales Usos

* Modelar visualmente el flujo de un caso de uso: Permite representar las secuencias de acciones en un caso de uso, facilitando su comprensión.
* Representar detalles de operaciones o algoritmos: Desglosa y documenta el comportamiento específico de algoritmos o métodos.
* Flexibilidad para diferentes tipos de flujos: Permite diagramar tanto flujos secuenciales como condicionales o paralelos.

La **actividad** es el elemento principal en un diagrama de actividades y se asocia a un elemento específico, como un caso de uso, clase o interfaz, para explicar su comportamiento. La actividad define el contexto del flujo de trabajo y puede hacer referencia a propiedades u operaciones del elemento asociado, proporcionando mayor precisión y claridad en la modelación del proceso.

Las actividades en un diagrama de actividades se representan como una red de nodos conectados, cada tipo de nodo cumple una función específica. Tipos:

* **Nodos de Acción:** Representan unidades atómicas dentro de la actividad, que llevan a cabo acciones específicas.
* **Nodos de Control:** Gestionan y dirigen el flujo de la actividad, determinando cómo se transiciona entre nodos.
* **Nodos de Objeto:** Representan los objetos utilizados dentro de la actividad, facilitando la visualización de los recursos involucrados.

Además, existen dos tipos de flujo dentro de un diagrama de actividades:

* **Flujos de Control:** Conectan los nodos para establecer el orden en que se ejecutan las actividades.
* Forma, Polígono

  Descripción generada automáticamente**Flujos de Objeto:** Conectan los nodos de objeto, representando la interacción y el intercambio de datos entre ellos.

Las actividades y sus nodos pueden tener precondiciones y postcondiciones, que establecen los requisitos que deben cumplirse antes de iniciar la actividad y los resultados esperados una vez finalizada.

Las **transiciones** conectan un estado de actividad o acción con el siguiente, permitiendo representar el flujo dentro del diagrama. Existen varios tipos de transiciones:

* **Transiciones Secuenciales:** Representan un flujo simple entre actividades, donde cada actividad sigue a la anterior de manera directa.
* **Bifurcaciones:** Definen caminos alternativos en el flujo, determinados por expresiones booleanas.
  + Una bifurcación cuenta con una transición de entrada y dos o más de salida.
  + Cada transición de salida está asociada a una condición booleana que se evalúa una sola vez.
  + Es importante que las condiciones en cada salida no se solapen y cubran todas las opciones posibles para garantizar un flujo continuo y eficiente.

**Nodos de Control**

Administran y dirigen el flujo de actividades, estableciendo cómo se conectan y organizan los distintos elementos en el diagrama. Tipos:

* **Nodos de Inicio:** Permiten iniciar uno o más flujos de manera concurrente dentro de la actividad. Su representación gráfica es un círculo de color negro.
* **Nodo de Fin:** Puede detener todos los flujos activos de la actividad o solo uno, según el contexto. Su representación gráfica es un círculo de color negro dentro de otro círculo blanco.
* **Nodos de Decisión:** Tienen un único punto de entrada y varios puntos de salida, cada uno con una condición exclusiva que dirige el flujo hacia diferentes caminos. Se suele representar como un rombo con una condición.
* **Nodos de Fusión:** Permiten unir varios flujos de entrada en un único flujo de salida, consolidando diferentes trayectorias en un solo flujo. Se representa igual que el de decisión.
* **Nodo “Fork”:** Divide un flujo en varios subflujos paralelos, permitiendo que múltiples actividades se ejecuten simultáneamente. Se representa con una línea continua gruesa que recibe un flujo y devuelve dos o más.
* **Nodo “Join”:** Sincroniza los flujos paralelos generados, combinándolos en un único flujo de salida. Al igual que el “Fork” es una línea continua que recibe más de un flujo y retorna solo uno.

**Nodos de Objeto**

Indican la disponibilidad de una instancia en un punto específico de la actividad. Estos:

* Tienen flujos de entrada y salida de tipo objeto.
* Representan la creación y el uso de objetos a través de los nodos de acción, permitiendo visualizar cómo y cuándo se manipulan los objetos del flujo de la actividad.

**Particiones de Actividad**

También conocidas como carriles o calles (swimlanes), dividen el diagrama de actividades horizontal o verticalmente para mejorar su claridad. Cada partición lleva un nombre único y agrupa acciones relacionadas, facilitando la asociación de acciones con diferentes elementos o responsables. La interpretación de estas particiones varía según el criterio del modelador, ofreciendo flexibilidad en la organización del flujo.

**Diagrama de Máquinas de Estado**

Define la secuencia de estados que un objeto atraviesa a lo largo de su vida, impulsada por eventos internos o externos. Este tipo de diagrama es útil para modelar el comportamiento dinámico de elementos como clases, casos de uso o sistemas completos. Relación con Otros Diagramas:

* **Diagramas de Interacción:** Estos diagramas representan el comportamiento de múltiples objetos interactuando, mientras que la máquina de estados modela el comportamiento de un solo objeto.
* **Diagramas de Actividades:** Se centran en el flujo de control entre actividades, no en los estados. En estos diagramas, la transición entre actividades se da al completarse la actividad anterior.

**Elementos de una Máquina de Estados**

1. **Estados:** Representan las condiciones o situaciones en las que se encuentra un objeto en un momento dado.
2. **Transiciones:** Indican las posibles rutas de cambio entre estados.
3. **Eventos:** Son sucesos que desencadenan cambios de estado en un momento específico.

En el diagrama, los estados de inicio y fin tienen una representación visual distinta de los demás estados, para identificar claramente el inicio y la finalización del flujo.

**Estados**

En una máquina de estados representa una condición o situación en la vida de un objeto. El objeto puede cumplir alguna condición específico. Los estados permiten modelar cómo un objeto responde y se adapta a diferentes circunstancias a lo largo de su ciclo de vida. Un objeto permanece en cada estado durante un período finito antes de realizar una transición a otro estado en respuesta a eventos.

Partes de un Estado

* **Nombre:** Texto que identifica el estado, puede no ser especificado.
* **Acciones de Entrada/Salida:** Se ejecutan al ingresar y salir del estado.
* **Transiciones Internas:** Cambios dentro del estado sin necesidad de salir de él.
* **Actividades Internas:** Operaciones que requieren un tiempo para completarse y que pueden interrumpirse si ocurre un evento.

También pueden existir **estados compuestos**; estados que contienen otros estados anidados en su interior, llamados submáquinas. Los estados anidados heredan todas las transiciones de sus estados contenedores, facilitando la organización y modularidad en el diagrama de estados. Se clasifican en:

* Sencillos: Cuentan con una sola región para los estados anidados.
* Ortogonales: Tienen múltiples regiones, permitiendo el modelado de comportamientos concurrentes.

Los eventos compuestos se pueden anidar a distintos niveles, aunque se recomienda limitar esta profundidad a dos o tres niveles para mantener la claridad y comprensión.

**Transiciones**

Es la relación entre dos estados que indica que un objeto ejecutará ciertas acciones y cambiará al segundo estado cuando ocurra un evento específico y se cumplan las condiciones necesarias. Este proceso de cambio es fundamental para modelar la respuesta de un objeto ante diversos eventos y su evolución durante el ciclo del sistema.

Cuando se produce el cambio de un estado a otro, se dice que la **transición se ha disparado**. Antes, el objeto se encuentra en el estado de origen; después, pasa al estado de destino. En los diagramas, una transición se representa mediante una línea continua dirigida desde el estado de origen hacia el estado de destino.

Un caso especial de transición es la **auto-transición**, donde el estado de origen y el de destino son el mismo. Esta transición es útil para representar cambios internos o actividades dentro de un estado sin pasar a un estado diferente. En una máquina de estados puede haber transiciones con múltiples orígenes (join) o con múltiples destinos (fork). Permiten modelar comportamientos complejos y situaciones donde el objeto puede dividirse o sincronizarse en varios flujos paralelos de actividades.

Una transición puede incluir tres elementos opcionales:

* **Evento(s) de disparo:** Son las ocurrencias internas o externas que activan la transición y desencadenan el cambio de estado del objeto.
* **Condición(es) de guarda:** También conocidas como condiciones de protección, son expresiones booleanas escritas entre corchetes que se evalúan una vez que ocurre el evento de disparo. Si la condición es verdadera, la transición procederá.
* **Acción(es):** Son los procesos a ejecutar cuando se activa la transición. Estas acciones pueden incluir llamadas a operaciones sobre el objeto, cambios en sus propiedades, entre otras modificaciones. Una vez iniciada no puede ser interrumpida y se ejecuta hasta completarse.

**Eventos**

Es un acontecimiento ocurre en un momento y lugar específicos. En las máquinas de estados, los eventos modelan los estímulos que desencadenan un cambio de estado. Estos eventos pueden ocurrir de forma interna o externa.

Tipos de Eventos

* **Evento de Llamada:** Representa la solicitud de una operación específica en un objeto. Cuando una operación se invoca sobre un objeto con una máquina de estados, siguiendo estos pasos:
  + El control pasa del emisor al receptor.
  + El evento activa la transición y la operación se ejecuta.
  + El receptor cambia a un nuevo estado, y el control vuelve al emisor.
  + Para asegurar coherencia, la operación debe incluirse en la lista de operaciones del objeto receptor y debe coincidir con la firma esperada.
* **Evento de Señal:** Consiste en el envío asíncrono de información sin asociarla a una operación específica. Este tipo de evento tiene una representación especial tanto para el envío como para la recepción de señales.
* **Evento de Tiempo:** Representa el paso del tiempo y se modela con la palabra “after” seguida de una expresión que especifica el intervalo temporal. El tiempo se mide a partir del momento en que el objeto entra en el estado actual.
* **Evento de Cambio:** Este evento representa una modificación en el estado o el cumplimiento de una condición y se modela con la palabra “when” seguida de una expresión booleana.

**Diagrama de Interacción**

Representa el comportamiento dinámico del sistema y el flujo de control durante una operación. Describen cómo los objetos interactúan entre sí mediante intercambio de mensajes para llevar a cabo tareas específicas. Estas interacciones modelan un “comportamiento” y ayudan a “realizar” o implementar un caso de uso.

Los elementos principales en los diagramas de interacción son:

* **Objetos:** participan en la interacción.
* **Enlaces:** establecen conexiones entre los objetos.
* **Mensajes:** representan la comunicación entre objetos.

Los diagramas de interacción incluyen notas y restricciones para mayor claridad.

Existen cuatro tipos de diagramas de interacción, nos centraremos en dos:

* **Diagrama de Secuencia:** Enfatiza el orden temporal de los eventos. Se representa como una tabla donde los objetos se disponen en el eje horizontal (X) y los mensajes en el eje vertical (Y), con los eventos ordenados cronológicamente.
* **Diagrama de Colaboración:** Resalta la estructura y las relaciones entre los objetos que envían y reciben mensajes. Gráficamente, parece como una red de nodos y arcos que muestra cómo los objetos están conectados.

Ambos diagramas son equivalentes, es decir, se puede convertir uno en el otro sin perder información. Sin embargo, visualmente, cada tipo destaca diferentes aspectos de la interacción, algunos detalles pueden percibirse con más claridad en uno que en el otro.

Estos diagramas de interacción reflejan cómo los objetos colaboran para realizar las funciones de una aplicación, representando la comunicación entre ellos en una tarea.

Una **interacción** se define como una unidad de comportamiento de un clasificador que ilustra cómo se lleva a cabo la comunicación a través del intercambio de mensajes. Esta muestra cómo los objetos colaboran para cumplir con un objetivo específico.

Un **mensaje** representa un tipo de comunicación dentro de una interacción. Los mensajes pueden invocar operaciones, crear o destruir instancias, o enviar señales entre los objetos. Son los elementos que facilitan la interacción y permiten colaborar entre sí.

Las interacciones y sus mensajes son fundamentales para modelar el comportamiento dinámico de las colaboraciones, que pueden incluir clases, interfaces, componentes, nodos y casos de uso. Estos aspectos dinámicos se representan mediante flujos de control que varían desde secuencias simples hasta flujos más complejos que incluyen bifurcaciones, iteraciones, recursión y concurrencia.

La **clase** define los atributos y métodos comunes que pueden compartir sus instancias, sirviendo como modelo para la creación de objetos. La **instancia** representa un objeto concreto a partir de esa clase, mientras que la **instancia nombrada** se refiere a una instancia específica que se identifica con un nombre único en el diagrama.

Esta notación es fundamental para visualizar las interacciones entre objetos, mostrando cómo se comunican y colaboran para llevar a cabo funciones en el sistema.

En los diagramas de interacción se representa la definición de un mensaje que se envió entre objetos con la siguiente sintaxis

Tabla

Descripción generada automáticamente con confianza bajareturn := nombreMensaje(parametro : tipoParam,. . . ) : tipoRetorno

Tabla 3: Notación de Mensajes en Diagramas de Interacción

La **secuenciación** es la dinámica en la que, cuando un objeto envía un mensaje a otro, el receptor puede comunicarse a su vez con un tercer objeto, desencadenando una serie de mensajes en forma de secuencia. Esta comienza con un proceso o hilo y se mantiene activa mientras este exista. Además, establece un flujo de control independiente, organizando los mensajes en un orden cronológico. Para facilitar la visualización de esta secuencia, se utiliza un número de secuencia, como 1, 1.1, 1.2, etc., que indica el orden de los mensajes desde el inicio de la interacción.

**Diagrama de Secuencia**

Representan la interacción entre los componentes del sistema desde una perspectiva temporal. La interacción se ilustra mediante el flujo de mensajes entre objetos o actores a Es capaz de describir procesos internos entre diversos módulos y en detallar las comunicaciones tanto con otros sistemas como con actores externos. Estos diagramas se vinculan a los casos de uso, mostrando cómo se llevan a cabo a través de interacciones entre diferentes entidades de objetos.

Imagen que contiene Tabla

Descripción generada automáticamenteNotación de los elementos en un diagrama de secuencia

Cada actor u objeto está representado por un eje vertical, conocido como línea de vida, que ilustra su existencia a lo largo del tiempo.

El intercambio de mensajes se indica mediante líneas horizontales que conectan los objetos, junto con su descripción correspondiente.

Cuando un objeto o actor está activo, se representa mediante un rectángulo sobre la línea de tiempo, cuyo tamaño corresponde a la duración de su actividad.

Los diagramas de secuencia se distinguen de los diagramas de colaboración por dos características principales:

* **Línea de vida:** Se representa como una línea vertical discontinua y señala la duración de un objeto. Esta comienza con el mensaje “create” y finaliza con “destroy”, incluye una señal en forma de X al final.
* **Foco de control:** Se representa como un rectángulo delgado y estrecho sobre la línea de vida. Indica el tiempo de ejecución de una acción y permite la posibilidad de anidar rectángulos adicionales hacia la derecha.

**Operadores de Control y Marcos de Interacción**

Un **marco de interacción** es una sección dentro de un diagrama de secuencia que incluye un operador de control, que define cómo se ejecuta la secuencia de mensajes. Tipos de ejecución:

* **Ejecución Opcional (opt):** Se ejecuta solo si se cumple una condición específica.
* **Ejecución Condicional (alt):** Se divide en varias subregiones, cada una representando una rama de la condición. Estas delimitan con líneas discontinuas horizontales.
* **Ejecución Iterativa (loop):** Indica un comportamiento repetitivo, especificando mínimos y máximos y la condición de guarda. El cuerpo del bucle se ejecuta mientras la condición de guarda sea verdadera antes de cada iteración.
* **Ejecución Paralela (par):** Se divide en varias subregiones separadas por líneas discontinuas horizontales. Cada una indica una computación paralela o concurrente.

**Diagrama de Colaboración**

Ilustra la interacción espacial entre objetos, centrándose en el intercambio de mensajes. Este utiliza los mismos elementos que los diagramas de secuencia, sin “líneas de vida” y de ‘foco de control”.

Este permite identificar los objetos y sus relaciones dentro del sistema y describe el flujo de mensajes entre los objetos o roles, resaltando la organización estructural de los objetos que colaboran. Los objetos se representan como nodos, mientras que los enlaces se muestran como arcos etiquetados con los mensajes. El orden de los mensajes se indica utilizando la numeración decimal de Dewey (1, 1.1, ...).

Los mensajes se representan mediante un enlace entre dos objetos, que se visualiza como una línea. Los mensajes se superponen a este enlace, y el orden se indica junto a su descripción.

Cada mensaje puede expresarse en lenguaje natural o en pseudocódigo, permitiendo incluir condiciones o invocar funciones.

**El Modelo Estructural en UML**

En UML, las partes estáticas del sistema se representan mediante varios tipos de diagramas que permiten capturar su estructura fundamental.

Cada uno de estos diagramas contribuye a una representación clara y organizada de los elementos que componen el sistema y sus relaciones internas.

Los diagramas estructurales son esenciales para visualizar, especificar, construir y documentar la estructura estable del sistema, que a menudo se denomina su “esqueleto”. Estos permiten una vista completa de la arquitectura estática del sistema, que permanece constante durante su operación.

En los diagramas, se incluyen varios elementos clave. Cada uno desempeña un papel específico en la definición de la estructura del sistema, ayudando a saber cómo se interrelacionan los elementos del software para crear una base donde se ejecutar funcionalidades del sistema.

**Diagrama de Clases**

Una **clase** define una categoría de objetos que comparten ciertas características de estructura (atributos) y comportamiento (operaciones). Los **objetos** son instancias específicas de una clase y tienen valores concretos asignados a sus atributos.

Cada clase incluye una serie de operaciones que los objetos pueden ejecutar cuando son invocados. Los objetos se activan al ser creados y están activos hasta que completan su función o son terminados por otro objeto en el sistema.

Mientras están activos, los objetos pueden recibir y responder a mensajes de otros objetos, permitiendo la interacción y colaboración entre ellos dentro del sistema.

**Compartimentos de una Clase**

La especificación de una clase se compone de varias secciones que permiten describir sus características en detalle:

1. **Compartimento de nombre:**

* Nombre de la clase (obligatorio).
* Estereotipos: pueden incluirse si es necesario, como ‘<<abstract>>’ que significa que la clase es abstracta.
* Valores etiquetados: permiten añadir metadatos adicionales.

1. **Compartimento de atributos:** detalla cada atributo de la clase, incluye:

* Nombre del atributo (obligatorio).
* Visibilidad: puede ser publica (‘+’), privada (‘-’), protegida (‘#’) o de paquete (‘→’).
* Tipo de dato, UML soporta tipos como ‘int’, ‘string’, ‘boolean’ y enumeraciones.
* Multiplicidad: define el rango de instancias, como ‘[0. . . 1]’, ‘[3]’, o ‘[2. . . \*]’.
* Valor inicial (opcional): útil en diseño, principalmente indica restricciones.

1. **Compartimento de operaciones:** lista cada operación que la clase puede realizar, indicando:

* Nombre de la operación (obligatorio, en fases tempranas de análisis).
* Visibilidad de la operación (pública, privada, protegida o de paquete).
* Tipo de retorno, debe ajustarse a los tipos soportados en UML.
* Lista de parámetros (más comúnmente detallada en el diseño) con:
  + Dirección del parámetro (in, out, inout).
  + Tipo de cada parámetro
  + Valor predeterminado, si aplica.

Estos elementos pueden simplificarse durante el proceso de análisis y diseño, y no siempre todos se presentan en el diagrama. Además, no tiene por qué haber una correspondencia exacta entre esta especificación y la sintaxis de un lenguaje de programación.

Las clases pueden representarse de diversas maneras, según el nivel de detalle.

**Relaciones**

En un modelo representan conexiones semánticas significativas entre elementos. En el contexto del modelado de clases y objetos, hay tres tipos:

1. **Relaciones entre objetos:** llamadas vínculos, representa una conexión directa entre dos objetos, estos pueden intercambiar mensajes. Cuando un objeto recibe un mensaje, se ejecuta la operación correspondiente.
2. **Relaciones de dependencia:** una relación entre dos o más elementos, donde un cambio en uno de ellos (llamado el proveedor) podría afectar a otro (llamado el cliente. Para especificar diferentes tipos de dependencias, se emplean diversos estereotipos para ajustar la semántica de la relación según el contexto.
3. **Relaciones entre clases:**

* **Generalización:** relación entre un elemento general y una versión más específica de ese. En la mayoría de los casos, una herencia simple es suficiente para representar esta relación. Se describe con la frase “es un tipo de”.
* **Asociación:** indica que pueden existir vínculos entre los objetos de ambas. Los vínculos entre objetos son instancias de las asociaciones entre clases.

Las asociaciones pueden incluir:

* + Nombre: describe la acción que el origen realiza sobre el destino.
  + Nombres de roles: alternativa al nombre, es el “papel” que desempeña.
  + Navegabilidad: restringe la dirección de la relación, señalando si una clase “conoce” a la otra.
  + Multiplicidad: especifica cuántos objetos pueden estar involucrados en la relación en un momento dado, en uno o ambos extremos. Se define mediante un intervalo que indica el mínimo y máximo de objetos permitidos.

Existen tipos especiales de asociación:

* + **Agregación:** relación entre una clase que representa un “todo” y otras que representan sus partes. Cada clase puede existir de forma independiente.
  + **Composición:** es una forma más fuerte de agregación, las partes y el “todo” están estrechamente ligados. Las partes en una composición no pueden existir independientemente del todo. Cada parte pertenece exclusivamente a un único “todo”.

Se pueden crear clases de asociación, son útiles en relaciones de asociación con multiplicidad, si existen atributos que no pertenecen a ninguna de las dos clases. La clase de asociación forma parte de la relación, incluyendo sus propiedades. Una clase de asociación actúa también como clase independiente, pudiendo tener sus propios atributos y operaciones.

Los diagramas de clases muestran un conjunto de clases, interfaces y colaboraciones, junto con sus relaciones. Parten de la idea de que el mundo real puede observarse a través de distintas abstracciones (enfoques subjetivos), empleando diversos mecanismos de abstracción como:

* Clasificación / Instanciación
* Composición / Descomposición
* Agrupación / Individualización
* Especialización / Generalización

La clasificación es uno de los mecanismos de abstracción más utilizados.

**Perspectivas de los Diagramas de Clases**

Hay diferentes perspectivas para abordar estos diagramas, cada una ofrece una visión específica del sistema en sus etapas de desarrollo. Las perspectivas permiten modelar un sistema desde distintos niveles de abstracción, facilitando la comprensión y diseño en función del contexto y los objetivos. Principales perspectivas de los diagramas de clases:

* **Conceptual:** Representa los conceptos del dominio, sin vincularse directamente a la implementación, es independiente del lenguaje de programación.
* **Especificación:** Define tipos de interfaces que pueden tener distintas implementaciones, según el entorno, el rendimiento o el proveedor.
* **Implementación:** Muestra las clases tal como se implementarán en el código. Es la más común, aunque puede ser preferible usar la perspectiva de especificación.

**Diagrama de Objetos**

Proporciona una representación completa o parcial de los objetos en un instante específico de ejecución. Modela las instancias de los elementos definidos en los diagramas de clases.

Tienen una serie de características:

* + Representa una vista estática del diseño y los procesos del sistema.
  + Cada instancia debe tener un nombre único en su contexto.
  + Las operaciones que se pueden ejecutar en el objeto se definen en su abstracción.
  + Comparte la misma notación que los diagramas de clases, pero el nombre del objeto se subraya.

Los diagramas de objetos son útiles para:

* + Ilustra las estructuras de datos y objetos dentro del sistema.
  + Especifica detalles más precisos del modelo.
  + Proporciona una “fotografía” del sistema en un momento de su ejecución.

Los objetos pueden desempeñar **roles** si están conectados mediante vínculos. Tipos:

* **Vínculo Bidireccional:** ambos elementos tienen conocimiento mutuo y pueden interactuar entre sí.
* **Vínculo Unidireccional:** solo uno de los elementos tiene conocimiento del otro y puede interactuar con él, pero no al revés.

Al visualizar el estado de un objeto, muestra el valor de sus atributos en un momento específico, ya que el estado de un objeto es dinámico y puede cambiar con el tiempo.

Existen dos estereotipos en las relaciones de dependencia entre objetos y clases:

* **instanceOf:** indica que el objeto es una instancia de la clase especificada.
* **instantiate:** indica que la clase crea instancias de otra clase.

**Diagrama de Paquetes**

Los **paquetes** se utilizan para organizar los elementos de modelado en unidades más grandes, que pueden ser manipuladas como un conjunto. Constituyen un mecanismo de agrupación dentro de UML, permitiendo organizar de manera estructurada los elementos.

Hay que controlar la visibilidad de los elementos dentro de un paquete, para que algunos sean accesibles desde fuera del paquete, y otros permanezcan ocultos. Los elementos dentro de un mismo paquete suelen estar relacionados semánticamente y tienden a cambiar juntos. Un paquete debe ser cohesivo y con un bajo acoplamiento, con un control adecuado sobre el acceso a su contenido.

Un paquete puede contener otros elementos como clases, interfaces, componentes, nodos, colaboraciones, casos de uso, diagramas y otros paquetes.

Cada elemento pertenece exclusivamente a un único paquete, y la visibilidad de los elementos contenidos en un paquete se controla de la misma manera que en las clases:

* **Público:** el elemento es visible para los contenidos de cualquier paquete que importe el paquete que lo contiene.
* **Privado:** el elemento no es visible fuera del paquete en el que se declara.

Los paquetes pueden estar relacionados mediante generalización y dependencias, estas últimas con semánticas propias como merge o import.

Gráficamente, un paquete se representa como una carpeta y debe tener un nombre que lo distinga de otros paquetes. Puede ser:

* **Nombre simple:** una cadena de texto única.
* **Nombre de camino:** el nombre del paquete precedido por el nombre del paquete contenedor.

**Importación**

Otorga un permiso unidireccional que permite a los elementos de un paquete acceder a los elementos de otro paquete. La relación de importación con el estereotipo import.

Las partes públicas de un paquete se denominan exportaciones. Los elementos que un paquete exporta solo son visibles para aquellos paquetes que lo importan explícitamente.

Las dependencias entre paquetes no son transitivas, facilitando la implementación de una arquitectura por capas. Esto significa que, zsi un elemento es visible dentro de un paquete, será visible en todos los paquetes que estén incluidos dentro de este paquete.

Los paquetes anidados pueden acceder a todos los elementos de los paquetes que los contienen.

**Generalización**

Es otro tipo de relación que puede existir entre paquetes que se utiliza para definir familias de paquetes. Los paquetes involucrados en esta relación siguen el mismo principio de sustitución que se aplica a las clases.

**Diagrama de Componentes**

Muestra cómo un sistema se organiza en componentes y sus relaciones. Es de mayor abstracción que los diagramas de clases, ya que un componente suele estar implementado por una o más clases en tiempo de ejecución.

Un componente es una unidad autónoma dentro de un sistema, definida por sus interfaces. Su propósito es encapsular una parte de las funcionalidades del sistema, accesibles únicamente a través de sus interfaces. Un componente puede ser reemplazado por otro siempre que cumpla con la misma especificación, ya que es autocontenido y reemplazable.

La estructura interna de un componente puede modelarse para delegar la realización de sus interfaces a alguna de sus partes internas. Los componentes pueden depender de otros, lo que significa que dependen de las interfaces de esos otros componentes. Además, pueden tener estereotipos para añadir información semántica.

Existen varios tipos de componentes:

* Ejecutables: Se ejecutan de forma autónoma.
* Librerías: Biblioteca de objetos estática o dinámica.
* Tabla: Representa una tabla en una base de datos.
* Archivo: Contiene código fuente o datos.
* Documento: Representa otro tipo de documento.

El otro elemento esencial del modelo arquitectónico son las interfaces. Que especifican un conjunto de características públicas que otros elementos pueden utilizar. Su propósito principal es separar la especificación (qué hace) de la implementación (cómo lo hace).

Una interfaz puede tener múltiples implementaciones validas, siempre que cumplan las condiciones establecidas en la propia interfaz. Los atributos y operaciones de una interfaz deben estar completamente especificados, actuando como un “contrato”:

* Firma completa de la operación: nombre, parámetros y valor de retorno.
* Semántica de la operación: puede expresarse en texto o pseudocódigo.
* Nombre y tipo de atributos.
* Restricciones de uso o comportamiento.

En los diagramas de componentes se representan las siguientes relaciones, cada una con una función específica en la arquitectura del sistema:

* **Dependencia:** Indica que un componente depende de otro para su funcionamiento. Un cambio en el componente del cual depende podría afectar al componente dependiente.
* **Generalización:** Relación en la cual un componente se considera una especialización de otro. Permite definir componentes y extender funcionalidades.
* **Proporciona (Interfaz):** Esta relación muestra que un componente ofrece una interfaz que otros componentes pueden utilizar. La interfaz proporcionada define el conjunto de operaciones o servicios que se pone a disposición de otros elementos. Proporcionada se representa como un pequeño círculo unido a uno de los lados del componente. Está conectado al componente que proporciona la interfaz e indica las operaciones que el componente pone a disposición de otros.
* **Consume (Interfaz):** Representa que un componente utiliza una interfaz proporcionada por otro componente. Es fundamental para mostrar la interacción entre componentes, donde uno depende de la funcionalidad del otro. Una interfaz consumida se representa mediante una línea que termina en un semicírculo o “socket” que se conecta a la interfaz proporcionada por otro componente. Este semicírculo indica que el componente necesita o consume los servicios definidos por la interfaz del componente al que está conectado.

Este diagrama muestra cómo los componentes se interrelacionan, a través de sus interfaces, para conformar la arquitectura completa de un sistema. Puede incluir, o no, la composición interna de los componentes o la especificación detallada de sus interfaces.

**Diagrama0020de Despliegue**

Representa la topología de hardware de un sistema. Captura la relación entre los elementos de un modelo conceptual o físico y los recursos de información asignados a ellos.

Es útil para:

* Indicar la distribución de los componentes.
* Evaluar el rendimiento y la carga del hardware del sistema.
* Analizar aspectos como redundancia, balance de carga, entre otros.

Los **nodos** son elementos físicos en tiempo de ejecución que representan recursos computacionales, como procesadores o dispositivos. Puede contener objetos o instancias y se utiliza para modelar la topología de hardware donde se ejecuta el sistema. Cada nodo debe contar con un nombre único, que puede ser simple o incluir una ruta de paquete. También pueden incorporar valores etiquetados o compartimentos adicionales .

Las **relaciones** conectan los distintos componentes dentro del diagrama de despliegue. En una relación, es posible representar:

* El tipo de comunicación entre componentes, utilizando una etiqueta.
* La cardinalidad de la relación.

Los **artefactos** representan elementos de implementación en un sistema. Pueden ubicarse de dos maneras:

* **Dentro de los nodos:** especificando el recurso computacional en el que se ejecutarán.
* **Mediante relaciones:** sin indicar un recurso de ejecución específico

Los artefactos son los elementos que se despliegan en los nodos y pueden asociarse con archivos fuente, ejecutables, librerías, scripts, tablas de base de datos y documentos. Estereotipos:

* Se emplea <<artifact>> para denotar un artefacto general.
* También pueden especificarse estereotipos más detallados como <<script>>, <<executable>>, <<library>>, <<document>>, entre otros.

**Técnicas Estructurales**

No existe un criterio definitivo para organizar las técnicas de la metodología estructurada. Aunque si podemos considerar dos enfoques:

* Según el enfoque de **representación**, clasifica las técnicas por la forma en que se representan los componentes y aspectos del sistema.
* Según el enfoque de **modelado**, clasifica las técnicas segun los modelos del sistema que se crean.

**Clasificación según el Enfoque de Representación**

El enfoque de representación organiza las técnicas de acuerdo con la forma en que presentan o describen los elementos y componentes del sistema, permitiendo diferentes niveles de detalle y especificación.

* **Gráficas:** Utilizan iconos o símbolos para representar visualmente los componentes de un aspecto específico del modelo. Cada técnica gráfica puede enfocarse en diferentes aspectos del sistema y suelen combinarse con otros tipos de técnicas para ofrecer una visión integral del sistema.
* **Textuales:** Ofrecen una descripción detallada de los componentes definidos en los diagramas gráficos, usando una gramática formal o semiformal. Suelen complementarse con representaciones gráficas, para mayor especificidad y exactitud en la descripción de componentes.
* **Marcos o Plantillas:** Sirven para organizar y especificar información relacionada con componentes de un modelo ya declarados en diagramas o esquemas. Se representan mediante formularios y se utilizan para documentar las propiedades y características de los componentes de forma estructurada.
* **Matriciales:** Se usan para verificar la precisión y la completitud de los modelos. Permiten estudiar las referencias cruzadas entre los componentes, ayudando en la validación y en el aseguramiento de la consistencia entre diferentes modelos.

**Clasificación según el Enfoque de Modelado**

El enfoque de modelado organiza las técnicas según el tipo de modelos del sistema que se crean, distinguiendo entre la funcionalidad, los datos, y los eventos temporales. A continuación, se detallan las técnicas específicas según cada dimensión:

1. **Modelado basado en la Información**

La dimensión de la información se centra en representar las **entidades** del sistema y sus **relaciones**. Se enfoca en describir los datos que el sistema maneja, cómo se transforman durante el procesamiento y cómo se generan como salida.

Su objetivo principal es capturar el dominio de datos del sistema, especificando cada ítem de datos que se acepta, procesa y produce como resultado.

Las principales técnicas que se utilizan son:

* + **Diagramas Entidad-Relación (ER):** Representan las entidades en el sistema y las relaciones entre ellas.
  + **Diagramas de Estructura de Datos:** Detallan la organización interna de los datos del sistema.
  + **Matriz Entidad/Entidad:** Ayuda a identificar y organizar las relaciones entre distintas entidades.

1. **Modelado basado en el Tiempo**

La dimensión del tiempo analiza cómo responde el sistema a los eventos que ocurren en momentos específicos. Esta se enfoca en describir las respuestas temporales del sistema, cuando ciertos eventos activan sus funciones o procesos.

Su objetivo es describir el momento en el que se activa una función o ejecuta un proceso en respuesta a un evento, gestionando el orden de acciones del sistema.

Las principales técnicas que se utilizan son:

* + **Diagramas de Transición de Estados:** Representan los posibles estados del sistema y sus transiciones en respuesta a eventos.
  + **Listas de Eventos:** Enumeran y describen los eventos que desencadenan alguna respuesta en el sistema.
  + **Redes de Petri:** Modelan secuencias y paralelismos de eventos en el sistema.
  + **Diagramas de Historia de la Vida de las Entidades:** Describen el ciclo de vida de cada entidad en función de los eventos que afectan su estado.
  + **Matriz Evento/Entidad:** Relaciona los eventos con las entidades que afectan o dependen de estos eventos.

1. **Modelado basado en la Función**

La dimensión de la función se enfoca en representar las funciones y procesos del sistema e interfaces que operan sobre los datos. Estudia cómo el sistema realiza sus tareas y ejecuta transformaciones sobre los datos.

Busca especificar las funciones, procesos o transformaciones del sistema, detallando cómo opera y qué tipo de procesamiento realiza sobre la información que maneja.

Sus técnicas principales son:

* + **Diagramas de Flujo de Datos (DFD):** Representanlas funciones del sistema, sus entradas y salidas, y las interfaces entre ellas.
  + **Lenguaje Estructurado:** Define procesos o funciones con mayor detalle, permitiendo una especificación formal o semiformal.
  + **Arboles de Decisión:** Representan decisiones complejas en forma jerárquica, mostrando las alternativas posibles.
  + **Tablas de Decisión:** Organizan las reglas de decisión en una tabla, mostrando las acciones a tomar en función de condiciones específicas.
  + **Diagramas de Descomposición Funcional:** Descomponen el sistema en sus subfunciones o módulos principales.
  + **Matriz Función/Entidad:** Relaciona las funciones con las entidades que participan en cada una.
  + **Diccionario de Datos:** Define y documenta los elementos de datos, proporcionando un glosario de términos y especificaciones para cada componente de datos en el sistema.

**Diagramas de Flujo de Datos**

Es una de las técnicas más utilizadas en el análisis estructurado, concebida para ayudar a los analistas a modelar las funciones que debe realizar el sistema y los datos entre ellas. Surge a finales de los 70, la propuesta incluía herramientas de soporte como el Diccionario de Datos y las Especificaciones de Procesos, que complementan el DFD.

Un DFD es una representación gráfica en forma de red que ilustra el flujo de información y las transformaciones desde la entrada hasta la salida del sistema. Esta permite modelar funciones y datos del sistema en distintos niveles de abstracción, proporcionando una visión clara y estructurada del sistema.

Para lograrlo el sistema se modela mediante un conjunto de DFD nivelados. Los superiores describen las funciones generales y las inferiores tareas más detalladas.

Los componentes principales de un DFD son: Procesos, Almacenes, Entidades Externas y Flujo de Datos, cada uno cumple un rol específico en el modelado funcional del sistema.

**Procesos**

Los procesos representan funciones que transforman los flujos de datos de entrada en flujos de salida, y se refieren a operaciones que el sistema debe realizar.

Cada proceso cumple con la regla de conservación de los datos. Si faltan datos necesarios para producir las salidas, se produce un error de conservación de datos; si algunos datos de entrada no se utilizan, se pierde información.

Para identificar los procesos en los DFD, existen algunas normas de nomenclatura:

* Numeración y nombre únicos en el conjunto de DFD.
* Nombre breve y preciso que refleje la función completa del proceso.
* Independencia de la implementación física en los DFD lógicos, que representan solo la lógica del sistema, sin detalles técnicos de realización.

**Almacenes de Datos**

Los almacenes de datos representan información almacenada temporalmente en el sistema, datos “en reposo”. Funcionan como dispositivos lógicos de almacenamiento, y pueden contener cualquier tipo de datos.

La representación gráfica de un almacén de datos varía según la metodología usada. Todos los almacenes de datos deben llevar un nombre que los describa.

Tienen una serie de características:

* **Legibilidad:** Un almacén puede repetirse en un DFD para mejorar su claridad.
* **Ubicación en Niveles:** En DFD nivelados, el almacén debe aparecer en el nivel más alto y ser representado en niveles inferiores donde interactúe directamente con procesos específicos.
* **Almacenes Locales:** Si un almacén se conecta solo con un proceso en un nivel, debe considerarse “local” y representarse en el nivel que detalle el proceso.

Existen dos tipos de estructuras de almacén

* **Estructura Simple:** Similar a un registro, compuesto de atributos donde identifican una ocurrencia.
* **Estructura Compleja:** Representada idealmente mediante un Diagrama Entidad-Relación (ER) para describir su organización interna.

La información de cada almacén debe documentarse en el Diccionario de Datos para asegurar precisión y consistencia.

**Entidades Externas**

Representan elementos que generan o consumen información del sistema sin pertenecer a él. Pueden corresponder a cualquier fuente que proporcione reciba información del sistema. Su representación gráfica puede variar según la metodología.

Como no forman parte del sistema, los flujos de información que emiten o reciben establecen la interfaz entre el sistema y su entorno. Para facilitar su identificación, cada entidad debe tener un nombre claro y representativo que describa su relación o función respecto al sistema. Si mejora la claridad, una entidad externa puede aparecer varias veces en un mismo DFD.

Generalmente, estas entidades se encuentran en el Diagrama de Contexto, el DFD de nivel más alto, donde se delimitan los límites y la interacción del sistema con el entorno. También pueden incluirse en otros diagramas de niveles inferiores para mejorar la comprensión del sistema.

**Flujo de Datos**

Es un camino por la que circulan datos de composición definida entre distintas partes del sistema, representada mediante arcos dirigidos. Según su persistencia, los flujos de datos se dividen en:

* **Flujos de Datos Discretos:** Aquellos que trasladan información en un momento específico.
* **Flujos de Datos Continuos:** Los datos permanecen activos a lo largo del tiempo.

La conexión directa entre dos procesos mediante un flujo de datos es posible si la información es síncrona, es decir, si el proceso receptor comienza su actividad cuando el proceso emisor ha completado la suya.

Las diferentes conexiones que se pueden hacer entre los procesos y almacenes son:

* **Flujo de Consulta:** El proceso puede acceder al almacén para consultar atributos específicos o verificar si se cumplen criterios predefinidos.
* **Flujo de Actualización:** Implica la modificación del almacén mediante la creación, eliminación, o actualización de valores de entidades o relaciones existentes.
* **Flujo de Diálogo:** Combina consulta y actualización en un flujo interrelacionado, permitiendo acceder a información y modificarla en un solo proceso.

Cada flujo de datos debe tener un nombre que refleje claramente el tipo de información que transporta. No es necesario nombrar los flujos de datos que ingresan o salen de almacenes simples, en estos casos su estructura corresponde a la del almacén.

Los flujos de datos no indican el control de la ejecución de los procesos, ni definen cuándo debe comenzar o finalizar el mismo.

**Niveles de Abstracción en los DFD**

En los DFD para sistemas grandes, se recomienda estructurarlos en niveles de abstracción mediante una aproximación descendente donde cada nivel ofrece una vista más detallada de las funciones. Estos se componen de un conjunto de DFD organizados jerárquicamente, donde cada nivel se expande sobre el anterior.

* **Diagrama de Contexto (o Nivel 0):** es el más general, define los límites del sistema e identifica las interfaces de entrada y salida.
* **Diagrama de Niveles:** detalla las funciones principales del sistema y sus relaciones, cada función de un nivel puede descomponerse en niveles inferiores.
* **Procesos Primitivos:** representan las funciones básicas que no se descomponen más, describiéndose con precisión.

Los niveles de descomposición suelen organizarse de la siguiente manera:

* **Nivel 0:** Diagrama de Contexto, que define los límites generales del sistema.
* **Nivel 1:** Diagrama de Subsistemas, representa las principales divisiones funcionales del sistema.
* **Nivel 2:** Diagramas de las Funciones de los Subsistemas, detalla las principales operaciones dentro de cada subsistema.
* **Nivel N:** Diagramas de Procesos, detallando cada subfunción hasta el nivel requerido.

Para asegurar consistencia, los flujos de datos en cada nivel deben coincidir con los flujos de datos en el nivel inferior. Las salidas también deben coincidir, salvo en casos de rechazos triviales.

La numeración de los DFD sigue un esquema estructurado: el Diagrama de Contexto es el 0, los Diagramas de Nivel 1 se numeran del 1 al N, y los niveles sucesivos se numeran según un sistema de numeración anidada o sistema Dewey, facilitando la referencia y coherencia en la documentación.