ASI- Guía de Preguntas Unidad 3

1. Define: Proceso, Proyecto y Producto. Dá un ejemplo relacionando los tres conceptos.

Un proceso es una secuencia de tareas relacionadas y organizadas en el tiempo, tienen una secuencia lógica y se llevan a cabo para un propósito dado. Otra definición es que es lo que la gente hace, usando procedimientos, métodos, herramientas y equipos, para transformar materia prima (entrada) en un producto (salida) que tenga valor para un cliente.
Un proceso de desarrollo de software es una secuencia de actividades que conducen a la elaboración de un producto de software.Las entradas son los requisitos de los usuarios y la salida es el software desarrollado.Otra definición es que un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software.
Según “El proceso de desarrollo de software” un proceso es una plantilla para crear proyectos, y un proceso de ing. de software es una definición del conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto.Osea, un proceso es una definición de un conjunto de actividades, no su ejecución.Un proceso cubre no solo el primer ciclo de desarrollo (primera versión) sino también los ciclos posteriores.

Los proyectos de desarrollo de software son instanciaciones del proceso definido organizacionalmente. Otra definición es que es un elemento organizativo a través del cual se gestiona el desarrollo de software, que tiene como resultado una versión de un producto. El primer proyecto dentro del ciclo de vida desarrolla y obtiene el sistema o producto inicial.
“Podemos pensar en una iteración como un miniproyecto”.
Según “El proceso de desarrollo de software” la idea de proceso es proporcionar un patrón dentro del cual las personas en su papel de trabajadores ejecutan un proyecto. Este patrón o plantilla indica los tipos de trabajadores que el proyecto necesita y los artefactos por los cuales hay que trabajar.
El producto son los artefactos que se crean durante la vida del proyecto.El producto software debería no ser solo el código máquina ejecutable, sino incluir los modelos, documentación, requisitos, casos

de uso, especificaciones no funcionales y casos de prueba.

Un ejemplo puede ser el desarrollo de un juego. El proceso abarca toda la vida del misma desde su primera versión hasta la última, los proyectos en cambia serán uno para la creación del juego original y luego otros para nuevas versiones. En cada uno de estos proyectos el juego que se obtiene y se ofrece a los clientes será el producto.

2. Defina los conceptos de herramienta, método, metodología.

Las herramientas en un proceso de software son programas usados para apoyar las actividades del proceso de la ingeniería de software. Otra definición es que son software que se utiliza para automatizar las actividades definidas en el proceso.
Es impensable hoy en día desarrollar software sin utilizar un proceso soportado por herramientas, son esenciales. Son buenas para automatizar procesos repetitivos, mantener las cosas estructuradas, generar grandes cantidades de información y para guiarnos a lo largo de un camino de desarrollo concreto.El proceso, ya sea explícita o implícitamente, especifica la funcionalidad de las herramientas, sus casos de uso. Que exista un proceso es el único motivo para necesitar herramientas.

Método
Enfoque al diseño de software donde se definen los modelos gráficos que hay que desarrollar, como parte del proceso de diseño.

Metodología
Conjunto de técnicas y métodos que se utilizan para diseñar un software.

3. ¿Qué es la ingeniería de software?

La ISO la define como:
(1) La aplicación sistemática de conocimientos científicos y tecnológicos, métodos y experiencias para el diseño, implementación, prueba y documentación de software;
(2) La aplicación de un enfoque sistemático, disciplinado cuantificable al desarrollo, operación y mantenimiento de software; o sea la aplicación de la ingeniería al software.

Otra definición es que la ingeniería de software es una disciplina de ingeniería que se interesa por todos los aspectos de la producción de software, desde las primeras etapas de la especificación del sistema hasta

el mantenimiento del sistema después de que se pone en operación.

4. ¿Qué define el proceso de desarrollo de software?

En el libro “El proceso de desarrollo de software” un proceso de desarrollo de software es una definición de las actividades necesarias para convertir los requisitos de usuario en un conjunto consistente de artefactos que conforman un producto de software, y para convertir los cambios sobre esos requisitos en un nuevo conjunto consistente de artefactos.

A esto podemos agregar que definen la funcionalidad de las herramientas (sus casos de uso),productos (que son los resultados de una actividad del proceso), roles (que reflejan las responsabilidades de la gente que interviene en el proceso),Precondiciones y postcondiciones (que son declaraciones válidas antes y después de que se realice una actividad del proceso o se cree un producto).

También definen cuales son los requisitos que el sistema debe cumplir, es decir, cuál será su funcionalidad o casos de uso, cómo se lo evaluará, cómo será la arquitectura del proceso y en algunos casos se planifican las iteraciones.

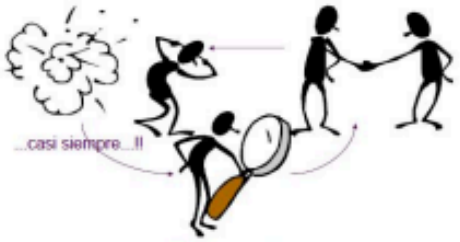
En las diapositivas dice que en la actividad de definición va la planificación del proyecto y el análisis de requisitos.

5. Explique el marco genérico para el desarrollo de software.

Ciclo de vida del software

15

Ciclo de vida del software



16

Obsolescencia del producto

Actividades del proceso

Actividades genéricas del proceso:

- Definición: el Qué. Incluye: planificación del proyecto y análisis de requisitos.
- Desarrollo: el Cómo. Incluye : diseño del sw, generación del código y prueba del sistema.
- Mantenimiento: el Cambio.
  - Corrección de errores.
  - Adaptaciones por evolución del entorno.
  - Mejoras en el negocio. Aumento de la capacidad del producto.

desde cero en un lenguaje de programación estándar como Java o C. Sin embargo, las aplicaciones de negocios no se desarrollan precisamente de esta forma. El nuevo software empresarial con frecuencia ahora se desarrolla extendiendo y modificando los sistemas existentes, o configurando e integrando el software comercial o componentes del sistema.

Existen muchos diferentes procesos de software, pero todos deben incluir cuatro actividades que son fundamentales para la ingeniería de software:

1. *Especificación del software* Tienen que definirse tanto la funcionalidad del software como las restricciones de su operación.
2. *Diseño e implementación del software* Debe desarrollarse el software para cumplir con las especificaciones.
3. *Validación del software* Hay que validar el software para asegurarse de que cumple lo que el cliente quiere.
4. *Evolución del software* El software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

Explayado en resumen del cuaderno.

6. Mencione los distintos modelos de proceso de desarrollo de software. Describa brevemente cada uno de ellos. Ventajas y desventajas.

Modelo lineal secuencial/en cascada:

Encadenamiento secuencial de las actividades. Cada etapa produce documentos que son la entrada a la siguiente. Para desarrollar una etapa debe concluirse la anterior (en la práctica se traslapan y hay retroalimentación entre ellas). Es un ejemplo de proceso dirigido por un plan, en principio debe planear y programar todas las actividades del proceso antes de empezar a trabajar con ellas.

Sus principales **etapas** son: Análisis y definición de requerimientos; Diseño del sist. y SW; Implementación y Prueba de unidad; Integración y prueba del sistema;Operación y mantenimiento.

En principio este modelo debe usarse cuando los requerimientos se entiendan bien y sea improbable el cambio radical durante el desarrollo del sist., así como cuando se trabaja con productos no novedosos.

Ventajas:

-Planificación sencilla.

-Una plantilla estructurada para ingeniería de software.

-Al producirse la documentación en cada fase el proceso se hace visible, de modo que los administradores monitorizan el proceso contra el plan de desarrollo.

Desventajas:

- Las iteraciones son costosas y aunque son pocas es normal, luego de un pequeño número de estas, congelar parte del desarrollo y continuar con las siguientes fases.

- Los problemas se dejan para su posterior resolución, lo que lleva a que estos sean ignorados o corregidos de una forma poco elegante.

- Existe una alta probabilidad de que el software no cumpla con los requisitos del usuario por el largo tiempo de entrega del producto.

- Es inflexible a la hora de evolucionar para incorporar nuevos requisitos. Es difícil responder a cambios en los requisitos. Esto se debe a los compromisos que se deben realizar temprano en el proceso por su partición inflexible en distintas etapas.

Modelo de construcción de prototipos:

Es un proceso que facilita al programador la creación de un modelo del software a construir. El prototipo es un modelo a escala del real que permite realizar un estudio sobre el mismo, pero no es tan funcional como el producto final. Es una versión inicial de un sistema de software usado para demostrar conceptos, tratar opciones de diseño y encontrar más sobre el problema y sus posibles soluciones.

Los prototipos no tienen que ser ejecutables para ser útiles.

Algunos tipos de prototipo son prototipos en papel o un modelo basado en PC que describa la interacción hombre-máquina, prototipo que implemente algunos subconjuntos de la función requerida del programa deseado, o programa existente que ejecute parte o toda la función deseada pero que tenga otras características que deban ser mejoradas en el nuevo trabajo de desarrollo. También están los prototipos mago de oz, donde solo se desarrolla la interfaz de usuario.

Ventajas:

-En la ingeniería de requerimientos ayuda con la selección y validación de requerimientos del sistema.

-En el proceso de desarrollo del sistema sirve para buscar soluciones específicas de software y apoyar el diseño de interfaces de usuario (la creación de prototipos es parte esencial de este proceso).

-Permiten al usuario ver que tan bien el sistema apoya su trabajo (pueden aparecer nuevas ideas de requerimientos y detectarse errores y omisiones de los mismos, así como descubrir áreas de fortalezas y debilidades en el software).

-Sirve para comprobar la factibilidad de un diseño propuesto.

Desventajas:

-Un problema puede ser que el prototipo quizá no se use necesariamente en la misma forma que el sistema final (ya sea porque el revisor no es un usuario típico, hay poco tiempo de capacitación en su uso o se evitan las partes lentas,etc.).

-A veces los administradores presionan para entregar un prototipo desechable (sobre todo si hay demoras en la entrega de la versión final), sin embargo esto no es recomendable y puede complicar el resto del proceso.

-Puede que sea necesario reaprendizaje cuando se libere el producto de software.

Modelo incremental:

Se diseña una implementación inicial, se la expone al comentario del usuario, y luego se desarrolla en sus diversas versiones hasta producir un sistema adecuado.La especificación, desarrollo y validación están entrelazadas en vez de separadas, hay rápido retroalimentación entre actividades.Cada versión del sistema incorpora alguna de las funciones que el cliente necesita.

Este enfoque es el más común para el desarrollo de sistemas de aplicación y suele ser el mejor que el desarrollo en cascada para la mayoría de los sistemas empresariales, de e-commerce y personales.Puede basarse en un plan, ser ágil, o más comúnmente una mezcla de dichos enfoques. Es parte fundamental de los enfoques ágiles.

Los incrementos pueden ser entregados al cliente. Cada incremento es diseñado, codificado, probado, integrado y entregado por separado. Los incrementos se desarrollan uno después de otro.

Ventajas:

- La especificación puede desarrollarse de forma creciente.

-Reduce el costo de adaptar los requerimientos cambiantes del cliente (menos análisis y documentación para reelaborar).

- Los usuarios y desarrolladores logran un mejor entendimiento del sistema.Se obtiene una rápida retroalimentación del usuario de forma más sencilla (es menos difícil que juzgar avances a partir de documentos de diseño de software), ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.

- Ideal cuando es difícil establecer todos los requerimientos por anticipado.

- Puede ser más rápida la entrega de software útil al cliente, aún si no tiene toda la funcionalidad.

Desventajas:

-El proceso no es visible (documentar cada versión puede ser inefectivo económicamente). Los administradores necesitan entregas regulares para medir el avance.

-La estructura del sistema tiende a degradarse con los nuevos

incrementos. Incorporar más cambios de Software se vuelve cada vez más costoso.

-Este modelo sólo es efectivo en proyectos pequeños o medianos con poco tiempo para su desarrollo y sin generar documentación para cada versión.

- Si los requerimientos crecen, la arquitectura y el diseño puede cambiar drásticamente.

Modelo de ingeniería de software orientada a la reutilización:

Se apoya en una gran base de componentes (que pueden ser sistemas por derecho propio) de SW reutilizables y la integración de marcos para la composición de los mismos.

En la mayoría de procesos de SW se da informalmente cierta reutilización de software. Las etapas de especificación y validación son similares a otros procesos, cambian las etapas intermedias. Los tipos de componentes de SW son: sistemas web, colecciones de objetos y sistemas de software independiente.

Ventajas:

-Menos software a desarrollar (menos costo y riesgo).

-Suele conducir a entregas más rápidas del software.

Desventajas:

-Inevitables los compromisos de requerimientos.

-Se pierde control de la evolución del sistema (las nuevas versiones de los componentes reutilizables no dependen de la organización).

Modelo espiral:

Se desarrolla en ciclos, en cada uno de los cuales se realiza:

-Se define el objetivo.

-Se analizan los riesgos.

-Desarrollo y verificación de la solución obtenida.

-Revisión de resultados y planificación del siguiente ciclo (si se decide que haya otro).

Cada ciclo en la espiral representa una fase del proceso de software.

Por ende, el ciclo más interno puede relacionarse con la factibilidad del sistema, el siguiente ciclo con la definición de requerimientos, el ciclo que sigue con el diseño del sistema, etcétera. El modelo en espiral combina el evitar el cambio con la tolerancia al cambio. Se distingue por su reconocimiento explícito del riesgo (siendo riesgo, algo que podría salir mal).

Ventajas:

- Resolución temprana de riesgos.
- Definición de arquitectura en sus fases iniciales.
- Basado en un proceso continuo de verificación de la calidad.
- Ideal para productos con un nivel alto de inestabilidad de los requerimientos.

Desventajas:

- No aplicable a proyectos bajo contrato.
- No recomendable en proyectos simples por su alto costo.

Modelos de Procesos Evolutivo:

Los modelos evolutivos son iterativos e incrementales y se caracterizan por la forma en que permiten desarrollar versiones cada vez más completas del software.

Cada una de las versiones es entregada al cliente, quien comienza a utilizarlo y probarlo.

7. ¿Qué es la calidad en el marco de un proceso de desarrollo de software?

La calidad del producto obtenido está fuertemente afectada por la calidad del proceso utilizado para producirlo.

■ Calidad es cumplir con los requerimientos de alguien.

■ Calidad es el valor para una persona Valor es aquello que se está dispuesto a pagar para obtener sus requerimientos.

■ Calidad es satisfacción de las necesidades y expectativas de los clientes y usuarios – consumidores “a menor costo”.

La calidad no tiene que ver sólo con lo que hace el software, sino también su comportamiento al ejecutarse, estructura y organización de los programas y la documentación asociada. Esto se refleja en los llamados calidad o atributos no funcionales del software. Ejemplos de dichos atributos son el tiempo de respuesta del software ante la duda de un usuario y la compresibilidad del código del programa.

8. Investigue los siguientes modelos de Calidad ISO 90003 y CMMI y realice el siguiente cuadro comparativo:

Modelo	Descripción	Ventajas	Desventajas	Gráfico
ISO	Norma internacional que establece los requisitos para un	-Mejora de la calidad del software	- Se centra principalmente en la gestión de la calidad	

	sistema de gestión de calidad para el software. Se basa en los principios de la norma ISO 9001, pero está específicamente adaptada para aplicarse al desarrollo, suministro y mantenimiento de software. Proporciona orientación sobre cómo establecer un sistema de gestión de calidad efectivo para garantizar que los procesos relacionados con el software cumplan con los requisitos de calidad y satisfagan las necesidades de los clientes, así como para detectar y corregir una serie de problemas de los productos software.	-Cumplimiento de normativas y requisitos legales -Mayor eficiencia y productividad (al establecer procesos claros y definidos. -Mejora de la reputación y la competitividad -Mejora de la comunicación y la colaboración dentro de la organización. -Reducción de los costos asociados con la corrección de errores y defectos en el software. -Aumento de la confianza de los clientes y socios comerciales en la capacidad de la organización para ofrecer software de calidad. -Facilitación del proceso de auditoría y evaluación de proveedores. - Mejor documentación de los sistemas. - Promueve la eficiencia y la mejora continua en los procesos de desarrollo de software.	y puede no abordar aspectos específicos de la ingeniería de software. - Requiere tiempo y recursos para implementar y mantener un sistema de gestión de calidad. - Puede ser percibida como demasiado burocrática por algunas organizaciones. -Permite una evaluación objetiva de la madurez y capacidad de los procesos de una organización. - Facilita la identificación de áreas de mejora y la implementación de acciones correctivas.	
CMMI	Es un modelo de mejora de procesos que proporciona una guía para el desarrollo y mejora de los procesos que se utilizan en una organización.Provee a las organizaciones de los elementos esenciales para un proceso efectivo. Con este modelo se establecen cinco niveless de madurez de una empresa y de sus procesos, dependiendo de una serie de	-Mejora la comunicación (interna y externa) al proporcionar un marco común. -Proporciona a los trabajadores una idea clara y global, lo que les permite centrarse en sus objetivos. -Aumenta la calidad de productos y servicios y reduce los tiempos de	- Puede requerir una curva de aprendizaje significativa para entender y aplicar correctamente el modelo. - La implementación completa y la mejora continua pueden ser costosas y requerir una inversión significativa de recursos. - Puede ser percibido	

	características.	entrega. -Ayuda a reducir los costes. -Mejora la satisfacción al cliente. -Es un modelo que cuenta con muchos años de experiencia. -Fiabilidad e imagen de la empresa	como complejo y difícil de entender para algunas organizaciones. -Su implementación no es sencilla y lleva bastante tiempo. -Requiere de equipos especializados y experimentados-El proceso de evaluación y optimización también tiene un coste elevado.	

Ventajas ISO explicadas:

1. Mejora de la calidad del software: Al establecer un sistema de gestión de calidad sólido, las organizaciones pueden mejorar la calidad de su software, lo que lleva a una mayor satisfacción del cliente y una menor tasa de errores y defectos.
2. Cumplimiento de normativas y requisitos legales: La conformidad con la norma ISO 90003 ayuda a las organizaciones a cumplir con las normativas y requisitos legales relacionados con la calidad del software, lo que puede ser un requisito para participar en licitaciones o contratos gubernamentales y comerciales.
3. Mayor eficiencia y productividad: Al establecer procesos claros y definidos para el desarrollo, suministro y mantenimiento de software, las organizaciones pueden mejorar su eficiencia y productividad, reduciendo el tiempo y los recursos necesarios para llevar a cabo estas actividades.
4. Mejora de la reputación y la competitividad: La certificación en ISO 90003 puede mejorar la reputación de una organización y su posición competitiva en el mercado al demostrar su compromiso con la calidad y la satisfacción del cliente.

Cuáles son los niveles de madurez en CMMI?

- Nivel de madurez 1 (Inicial). Los procesos se encuentran inestables. Se refiere al momento en que la empresa no tiene definido el Ciclo de Vida de Desarrollo de sus proyectos; en

ocasiones, entrega sus proyectos con defectos, tarde o excediendo el presupuesto, y es el cliente quien se lo señala. La compañía no es consciente de los motivos porque no hay un grupo de trabajo dedicado a analizar los fallos o un proceso estándar para detectarlos y subsanarlos.

- Nivel de madurez 2 (Gestionado). Los proyectos se "planifican, ejecutan, miden y controlan", pero sigue habiendo errores, si bien ahora es posible saber de dónde provienen para corregirlos.
- Nivel de madurez 3 (Definido): la empresa ha definido el ciclo de vida de desarrollo de sus proyectos. Cuenta con estándares de evaluación y es consciente de sus deficiencias. De hecho, ya no es el cliente quien los detecta, sino un equipo designado internamente a tal efecto, y los desarrolladores de la compañía aplican el sistema de resolución de forma idéntica.
- Nivel de madurez 4 (Gestionado cuantitativamente). Existe un ciclo de vida de desarrollo de los proyectos de software establecido, los errores son mínimos, se entrega en fecha y forma, y el tamaño de los proyectos llega definido con criterios cuantificables. Además, se evidencian los resultados de monitorear los procesos para mejorar sus propios criterios de calidad.
- Nivel de madurez 5 (Optimización): el ciclo de vida de desarrollo de sus proyectos de software está perfectamente definido, entrega sus proyectos a tiempo y con pocos o ningún fallo, cumple con los criterios de calidad internacionales, detecta errores internamente, existe un protocolo de corrección estándar, etc.

9. ¿Qué es el Proceso Unificado de Desarrollo (PUD)?

El Proceso Unificado de Desarrollo es un proceso de desarrollo de software. Sin embargo, el PUD es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.

10. Mencione y explique las características del PUD.

-El Proceso Unificado está **basado en componentes**. El sistema de software en construcción está formado por componentes software interconectados a través de interfaces.

-El PUD **utiliza UML** para preparar todos los esquemas de un sistema de software. De hecho UML es parte esencial del PUD, sus desarrollos fueron paralelos.

Sin embargo los **verdaderos aspectos definitorios del PUD son**:

-Dirigido por casos de uso:

Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante, representan los requisitos funcionales. Todos los casos de uso constituyen el modelo de casos de uso el cual describe la funcionalidad total del sistema.

Puede decirse que la estrategia de casos de uso busca responder ¿Qué debe hacer el sistema para cada usuario? Forzandonos a pensar en la importancia para el usuario y no solo en funciones que estaría bueno tener.

Los casos de uso también guían el proceso de desarrollo (diseño, implementación y prueba). A partir del modelo de casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo los casos de uso.

Dirigido por casos de uso quiere decir que el proceso de desarrollo sigue un hilo, avanza a través de una serie de flujos de trabajo que parten de los casos de uso.

Los casos de uso finales son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba.

Los casos de uso guían la arquitectura del sistema y la arquitectura del sistema influye en la selección de los casos de uso.

-Centrado en la arquitectura:

El concepto de arquitectura de software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado. En un sistema de SW la arquitectura se describe mediante diferentes vistas del sistema en construcción.

Cada producto tiene tanto una función como una forma, en este caso la función corresponde a los casos de uso y a la arquitectura la forma,debe haber interacción entre ellos.Los casos de uso deben encajar en la arquitectura al llevarse a cabo y esta debe permitir el desarrollo de los casos de uso requeridos ahora y en el futuro. Tanto la arquitectura como los casos de uso deben evolucionar en paralelo.

Los arquitectos moldean al sistema para darle una forma, esta forma es la arquitectura, que debe diseñarse para que el sistema evolucione no solo en su desarrollo inicial sino también a lo largo de futuras generaciones. Para encontrar esta forma los arquitectos deben trabajar sobre la comprensión general de las funciones claves, es decir, sobre los casos de uso claves del sistema.

El trabajo del arquitecto se explaya un poco más en el libro.

-Iterativo e incremental:

Al ser el desarrollo de software tan largo, es práctico dividir el trabajo en parte más pequeñas o mini proyectos, cada uno de los cuales es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos al crecimiento del producto. Para mayor efectividad las iteraciones deben estar controladas (deben seleccionarse y ejecutarse de una forma planificada, por esto son miniproyectos).

En primer lugar la iteración trata un grupo de casos de uso que juntos amplían la utilidad del producto desarrollado hasta ahora, y en segundo lugar tratan los riesgos más importantes.Las iteraciones sucesivas se construyen sobre los artefactos tal como quedaron al final de la última iteración. En cada una de estas los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes, y verifican que los componentes satisfacen los casos de uso.

Un incremento no es necesariamente aditivo, los diseñadores pueden tener que reemplazar un diseño superficial por uno más detallado o sofisticado (sobre todo en las primeras fases del ciclo de vida).

Algunos beneficios de la iteración controlada son que se reduce el coste del riesgo a los costes de un solo incremento y que reconoce que las necesidades y requerimientos de los usuarios no pueden definirse completamente al principio, sino que se refinan en iteraciones sucesivas, facilitando la adaptación a los requisitos cambiantes.

Otros beneficios y detalles explayados en el libro.

Estos 3 conceptos son de igual importancia y la eliminación de uno de ellos reduciría drásticamente el valor de del PUD. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada

iteración.

11. Defina los conceptos de Flujo de Trabajo, Artefacto, Actividad y Trabajador. Ejemplifique.

Artefacto es un término general para cualquier tipo de información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema.Hay dos tipos de artefacto: de ingeniería y de gestión (análisis de negocio, plan de desarrollo, plan para asignación de personas concretas a trabajadores, diseño de las actividades de los trabajadores en el plan, etc. Un ejemplo serían los documentos de requisitos.

La palabra trabajador denomina a los puestos a los cuales se pueden asignar personas, y los cuales esas personas pueden aceptar. Un tipo de trabajador es un papel que un individuo puede desempeñar en el desarrollo de software. Cada trabajador tiene un conjunto de responsabilidades y lleva a cabo un conjunto de actividades en el desarrollo de software.. Ejemplos son un especificador de casos de uso, un arquitecto, un ingeniero de componentes, o un ingeniero de pruebas de integración.Una persona puede ser varios trabajadores dentro de un proyecto, o puede ser muchos trabajadores durante la vida de un proyecto.A su vez, un trabajador puede representar a un conjunto de personas que trabajan juntas.

Un flujo de trabajo es un conjunto de actividades, describimos un proceso entero en términos de ellos.En términos de UML un flujo de trabajo es un estereotipo de colaboración, en el cual los trabajadores y los artefactos son los participantes. Los trabajadores y artefactos que participan en un flujo de trabajo pueden participar también en otros flujos de trabajo.

En el libro “El proceso unificado de desarrollo de software”, y en el contexto de los diagramas de actividades y flujos de trabajo del PUD, se menciona que las actividades por trabajador, son trabajos significativos para una persona que actúe como trabajador.

12. ¿Cuáles son las actividades protectoras o de soporte en todo proceso de desarrollo?

En “Ingeniería de software” se menciona como actividades de soporte al proceso la documentación y el manejo de la configuración del software. En “Ingeniería de requisitos” se dice que si utilizamos como referencia el proceso unificado y su definición de

disciplina, las disciplinas de apoyo son: Gestión de la configuración y del cambio, Gestión de Proyectos, Ambiente.
Además, en las diapositivas de la unidad se mencionan como actividades de soporte (protectoras) las siguientes:

- Seguimiento y control de proyectos
- Gestión de riesgos
- Aseguramiento de la calidad del software
- Revisiones técnicas formales
- Medición
- Gestión de la configuración del software
- Gestión de la reutilización
- Preparación y producción del producto de trabajo

13. ¿Qué es UML? ¿Cómo surgió?
UML es un lenguaje de modelado visual.

UML fue desarrollado en un esfuerzo para simplificar y consolidar el gran número de métodos de desarrollo orientado a objetos que habían surgido.

...
UML son una serie de normas y estándares
que dicen cómo se debe representar algo.
...

Era evidente desde hace algún tiempo la necesidad de un lenguaje de modelado visual y consistente, en el cual expresar los resultados de las bastante numerosas metodologías de orientación a objetos existentes a principios de los noventa.

14. ¿Para qué se utiliza UML?

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos.

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. La estructura estática define los tipos de objetos importantes para un sistema y para su implementación, así como las relaciones entre los objetos. El comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos. El modelar un sistema desde varios puntos de vista, separados pero relacionados, permite entenderlo para diferentes propósitos.

...

especializada con un lenguaje especial. UML es un lenguaje de modelado discreto. No se creó para modelar sistemas continuos como los basados en ingeniería y física. UML quiere ser un lenguaje de modelado universal, de propósito general, para sistemas discretos, tales como los compuestos por software, firmware o lógica digital.

15. Explique brevemente la función de las vistas de UML:

Vistas de UML

No hay ninguna línea entre los diferentes conceptos y las construcciones en UML, pero, por conveniencia, nosotros los dividimos en varias vistas. Una vista es simplemente un subconjunto de UML que modela construcciones que representan un aspecto de un sistema. La división en diversas vistas es algo arbitraria, pero esperamos que sea intuitiva. Una o dos clases de diagramas proporcionan una notación visual para los conceptos de cada vista.

En el nivel superior, las vistas se pueden dividir en tres áreas: clasificación estructural, comportamiento dinámico, y gestión del modelo.

La clasificación estructural describe los elementos del sistema y sus relaciones con otros elementos. Los clasificadores incluyen clases, casos del uso, componentes, y nodos y elementos proporcionan la base sobre la cual se construye el comportamiento dinámico. La clasificación de las vistas incluye la vista estática, la vista de casos de uso, y la vista de implementación.

El comportamiento dinámico describe el comportamiento de un sistema en el tiempo. El comportamiento se puede describir como serie de cambios a las fotos del sistema dibujadas a partir de la visión estática. Las vistas de comportamiento dinámico incluyen vista de la máquina de estados, la vista de actividad, y la vista de interacción.

La gestión del modelo describe la organización de los propios modelos en unidades jerárquicas. El paquete es la unidad genérica de organización para los modelos. Los paquetes especiales incluyen a los modelos y a los subsistemas. La vista de gestión del modelo cruza las otras vistas y las organiza para el trabajo de desarrollo y el control de configuración.

Tabla 3.1 Vistas y diagramas de UML

Área	*Vista*	*Diagramas*	*Conceptos Principales*
estructural	vista estática	diagrama de clases	clase, asociación, generalización, dependencia, realización, interfaz
	vista de casos de uso	diagrama de casos de uso	caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso
	vista de implementación	diagrama de componentes	componente, interfaz, dependencia, realización
	vista de despliegue	diagrama de despliegue	nodo, componente, dependencia, localización
dinámica	vista de máquina de estados	diagrama de estados	estado, evento, transición, acción
	vista de actividad	diagrama de actividad	estado, actividad, transición de terminación, división, unión
	vista de interacción	diagrama de secuencia	interacción, objeto, mensaje, activación
		diagrama de colaboración	colaboración, interacción, rol de colaboración, mensaje
gestión del modelo	vista de gestión del modelo	diagrama de clases	paquete, subsistema, modelo
extensión de UML	todas	todos	restricción, estereotipo, valores etiquetados

Vista estática

La vista estática modela los conceptos del dominio de la aplicación, así como los conceptos internos inventados como parte de la implementación de la aplicación. Esta visión es estática porque no describe el comportamiento del sistema dependiente del tiempo, que se describe en otras vistas. Los componentes principales de la vista estática son las clases y sus relaciones: asociación, generalización, y varias clases de dependencia, tales como realización y uso. Una clase es la descripción de un concepto del dominio de la aplicación o de la solución de la aplicación. Las clases son el centro alrededor del cual se organiza la vista de clases; otros elementos pertenecen o se unen a las clases. La visión estática se exhibe en los diagramas de clases, llamados así porque su objetivo principal es la descripción de clases.

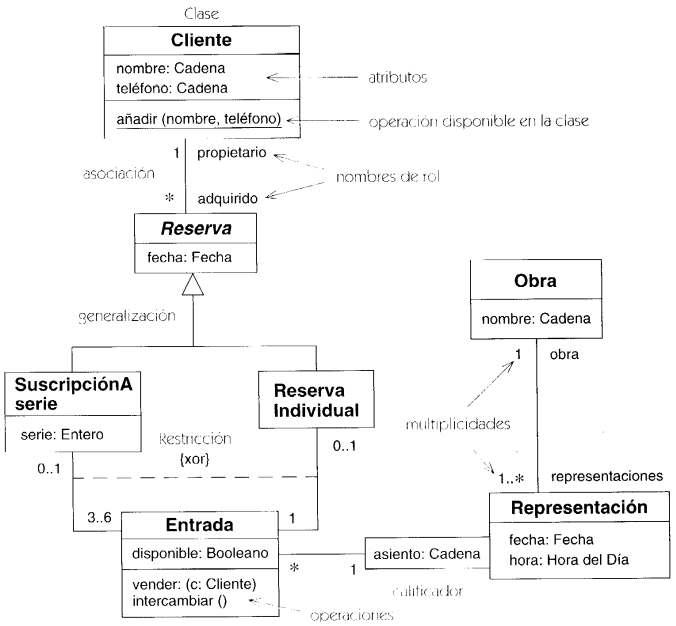


Figura 3.1 Diagrama de clases

Vista de los casos de uso

La vista de los casos de uso modela la funcionalidad del sistema según lo perciben los usuarios externos, llamados actores. Un caso de uso es una unidad coherente de funcionalidad, expresada como transacción entre los actores y el sistema. El propósito de la vista de casos de uso es enumerar a los actores y los casos de uso, y demostrar qué actores participan en cada caso de uso.

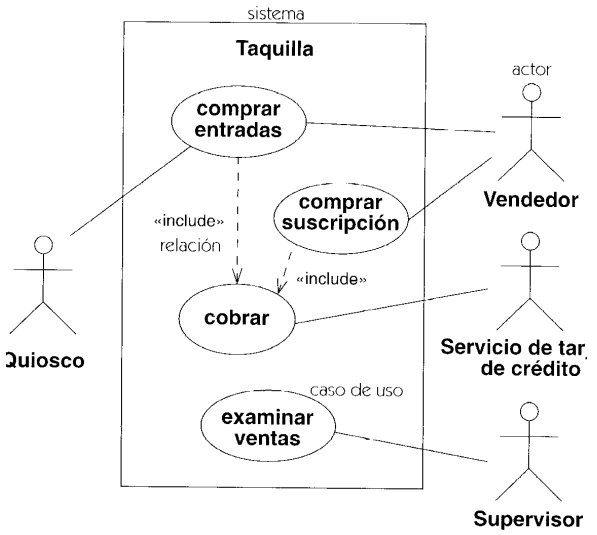


Figura 3.2 Diagrama de casos de uso

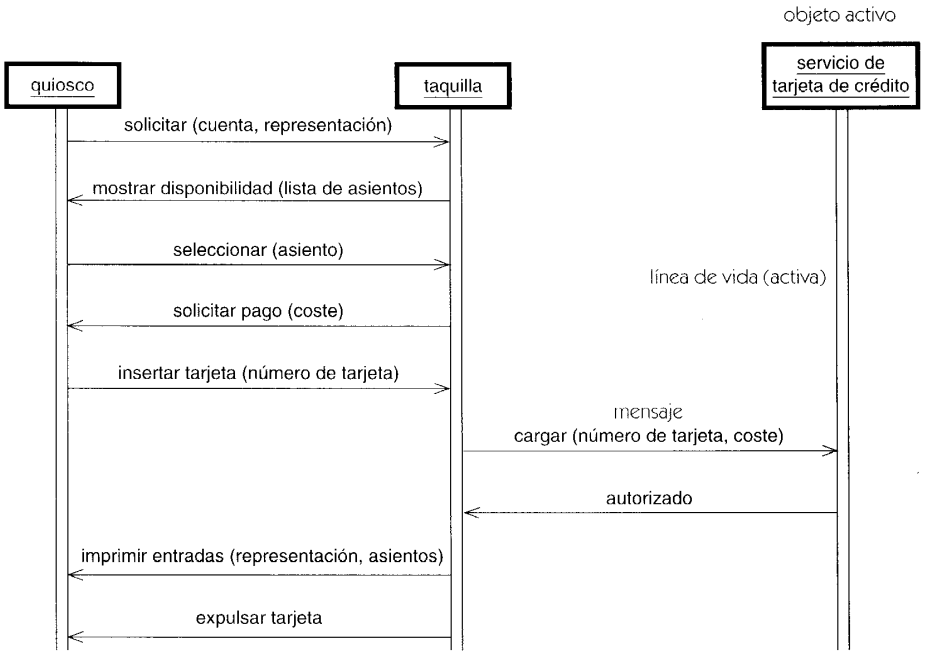
Los casos de uso se pueden también describir en varios niveles de detalle. Se pueden sacar partes como factor común y ser descritos en términos de otros casos de uso más simples. Un caso del uso se implementa como una colaboración en la vista de interacción.

Vista de interacción

La vista de interacción describe secuencias de intercambios de mensajes entre los roles que implementan el comportamiento de un sistema. Un rol de clasificador, o simplemente "rol", es la descripción de un objeto, que desempeña un determinado papel dentro de una interacción, distinto de los otros objetos de la misma clase. Esta visión proporciona una vista integral del comportamiento de un sistema —es decir, muestra el flujo de control a través de muchos objetos—. La vista de interacción se exhibe en dos diagramas centrados en distintos aspectos: diagramas de secuencia y diagramas de colaboración.

El diagrama de secuencia

Un diagrama de secuencia muestra un conjunto de mensajes, dispuestos en una secuencia temporal. Cada rol en la secuencia se muestra como una línea de vida, es decir, una línea vertical



se muestran como flechas entre las líneas de vida. Un diagrama de secuencia puede mostrar un escenario, es decir, una historia individual de una transacción.

Un uso de un diagrama de secuencia es mostrar la secuencia del comportamiento de un caso del uso. Cuando está implementado el comportamiento, cada mensaje en un diagrama de secuencia corresponde a una operación en una clase, a un evento disparador, o a una transición en una máquina de estados.

El diagrama de colaboración

Una colaboración modela los objetos y los enlaces significativos dentro de una interacción. Los objetos y los enlaces son significativos solamente en el contexto proporcionado por la interacción. Un rol describe un objeto, y un rol en la asociación describe un enlace dentro de una colaboración. Un diagrama de colaboración muestra los roles en la interacción en una disposición geométrica (Figura 3.4). Los mensajes se muestran como flechas, ligadas a las líneas de la relación, que conectan a los roles. La secuencia de mensajes, se indica con los números secuenciales que preceden a las descripciones del mensaje.

Un uso de un diagrama de colaboración es mostrar la implementación de una operación. La colaboración muestra los parámetros y las variables locales de la operación, así como asociaciones más permanentes. Cuando se implementa el comportamiento, la secuencia de los mensajes corresponde a la estructura de llamadas anidadas y el paso de señales del programa.

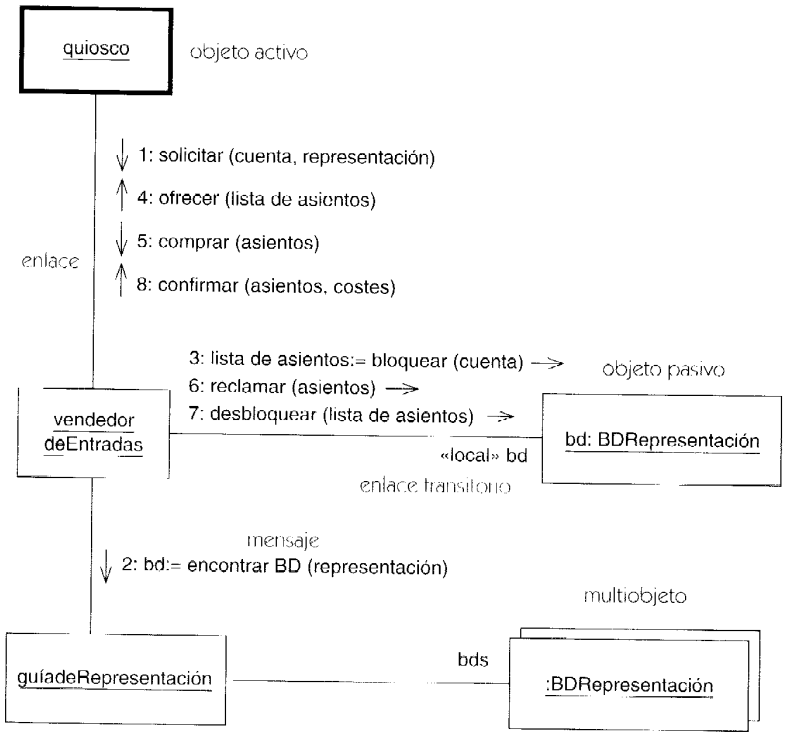


Figura 3.4 Diagrama de colaboración

Tanto los diagramas de secuencia como los diagramas de colaboración muestran interacciones, pero acentúan aspectos diferentes. Un diagrama de secuencia muestra secuencias en el tiempo como dimensión geométrica, pero las relaciones entre roles son implícitas. Un diagrama de colaboración muestra las relaciones entre roles geométricamente, y relaciona los mensajes con las relaciones, pero las secuencias temporales están menos claras, porque vienen dadas por los números de secuencia. Cada diagrama debe ser utilizado cuando su aspecto principal es el foco de atención.

Vista de la máquina de estados

Una máquina de estados modela las posibles historias de vida de un objeto de una clase. Una máquina de estados contiene los estados conectados por transiciones. Cada estado modela un período de tiempo, durante la vida de un objeto, en el que satisface ciertas condiciones. Cuando ocurre un evento, se puede desencadenar una transición que lleve el objeto a un nuevo estado. Cuando se dispara una transición, se puede ejecutar una acción unida a la transición. Las máquinas de estados se muestran como diagramas de estados.

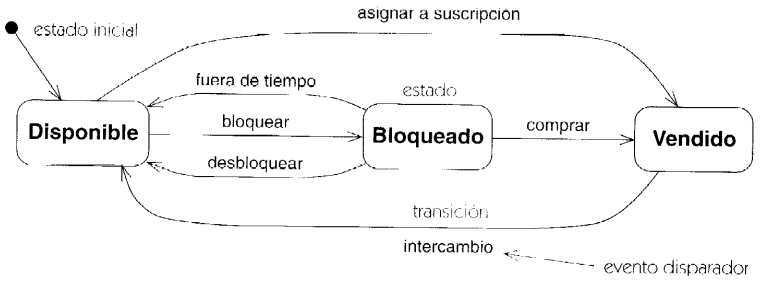


Figura 3.5 Diagrama de estados

Las máquinas de estados se pueden utilizar para describir interfaces de usuario, controladores de dispositivo, y otros subsistemas reactivos. También pueden usarse para describir los objetos pasivos que pasan por varias fases cualitativas distintas, durante su tiempo de vida, cada una de las cuales tiene su propio comportamiento especial.

Vista de actividades

Un grafo de actividades es una variante de una máquina de estados, que muestra las actividades de computación implicadas en la ejecución de un cálculo. Un estado de actividad representa una actividad: un paso en el flujo de trabajo o la ejecución de una operación. Un grafo de actividades describe grupos secuenciales y concurrentes de actividades. Los grafo de actividades se muestran en diagramas de actividades.

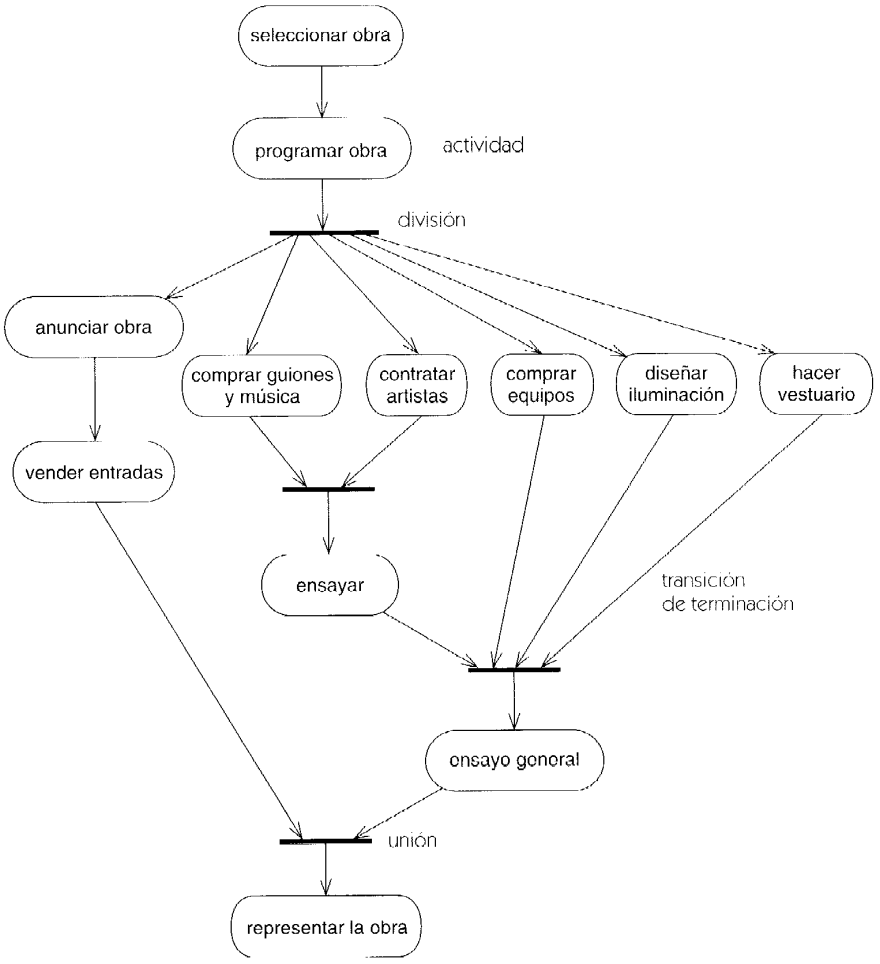


Figura 3.6 Diagrama de actividades

Este ejemplo muestra un diagrama de actividades, cuyo propósito es modelar los procesos reales de una organización humana. El modelado de tales negocios es un propósito importante de los diagramas de actividades, pero los diagramas de actividades se pueden también utilizar para modelar actividades software. Un diagrama de actividades es provechoso para entender el comportamiento de alto nivel de la ejecución de un sistema, sin profundizar en los detalles internos de los mensajes, lo que requeriría un diagrama de colaboración.

Los parámetros de entrada y de salida de una acción se pueden mostrar usando las relaciones de flujo que conectan la acción y un estado de flujo del objeto.

Vistas físicas

Las vistas anteriores modelan los conceptos de la aplicación desde un punto de vista lógico. Las vistas físicas modelan la estructura de la implementación de la aplicación por sí misma, su organización en componentes, y su despliegue en nodos ejecución. Estas vistas proporcionan una oportunidad de establecer correspondencias entre las clases y los componentes de implementación y nodos.

Hay dos vistas físicas: la vista de implementación y la vista de despliegue.

La vista de implementación modela los componentes de un sistema —a partir de los cuales se construye la aplicación— así como las dependencias entre los componentes, para poder determinar el impacto de un cambio propuesto. También modela la asignación de clases y de otros elementos del modelo a los componentes.

La vista de implementación se representa en diagramas componentes. La Figura 3.7 muestra un diagrama de componentes para el sistema de la taquilla.

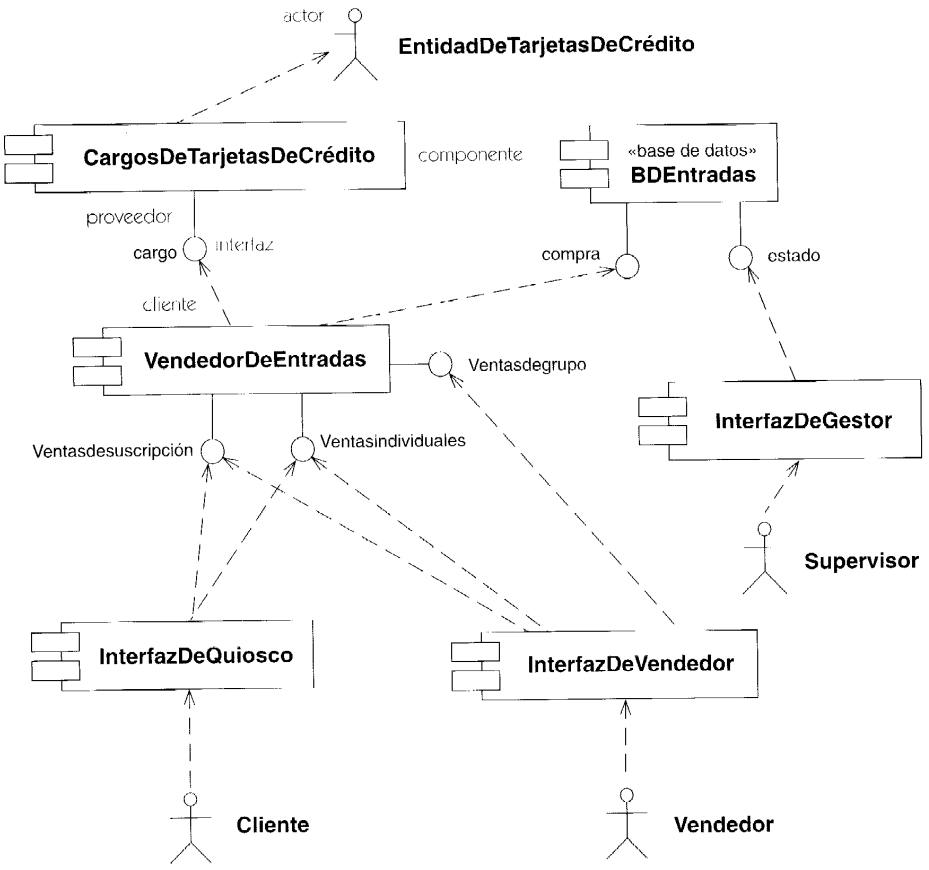


Figura 3.7 Diagrama de componentes

Un círculo pequeño con un nombre es una interfaz —un conjunto coherente de servicios—. Una línea sólida que va desde un componente a una interfaz, indica que el componente proporciona los servicios de la interfaz.

Una flecha de guiones de un componente a una interfaz indica que el componente requiere los servicios proporcionados por interfaz. Por ejemplo, las ventas de suscripciones y las ventas

La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos. Un nodo es un recurso de ejecución, tal como una computadora, un

dispositivo, o memoria. Esta vista permite determinar las consecuencias de la distribución y de la asignación de recursos.

La vista de despliegue se representa en diagramas de despliegue. La Figura 3.8 muestra un diagrama de despliegue del nivel de descriptor para el sistema de taquilla. Este diagrama muestra los tipos de nodos del sistema y los tipos de componentes que contienen. Un nodo se representa como un cubo.

La Figura 3.9 muestra un diagrama de despliegue del nivel de instancia, para el sistema de taquilla. El diagrama muestra los nodos individuales y sus enlaces, en una versión particular del sistema. La información de este modelo es consistente con la información del nivel de descriptor de la Figura 3.8.

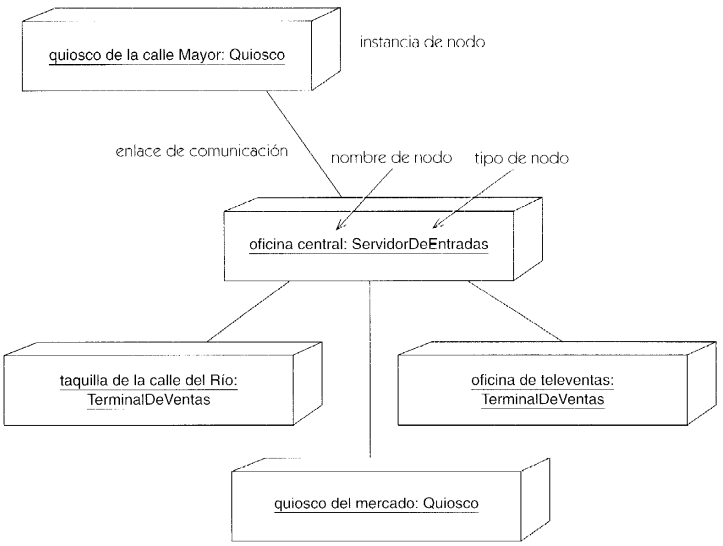


Figura 3.9 Diagrama de despliegue (nivel de instancia)

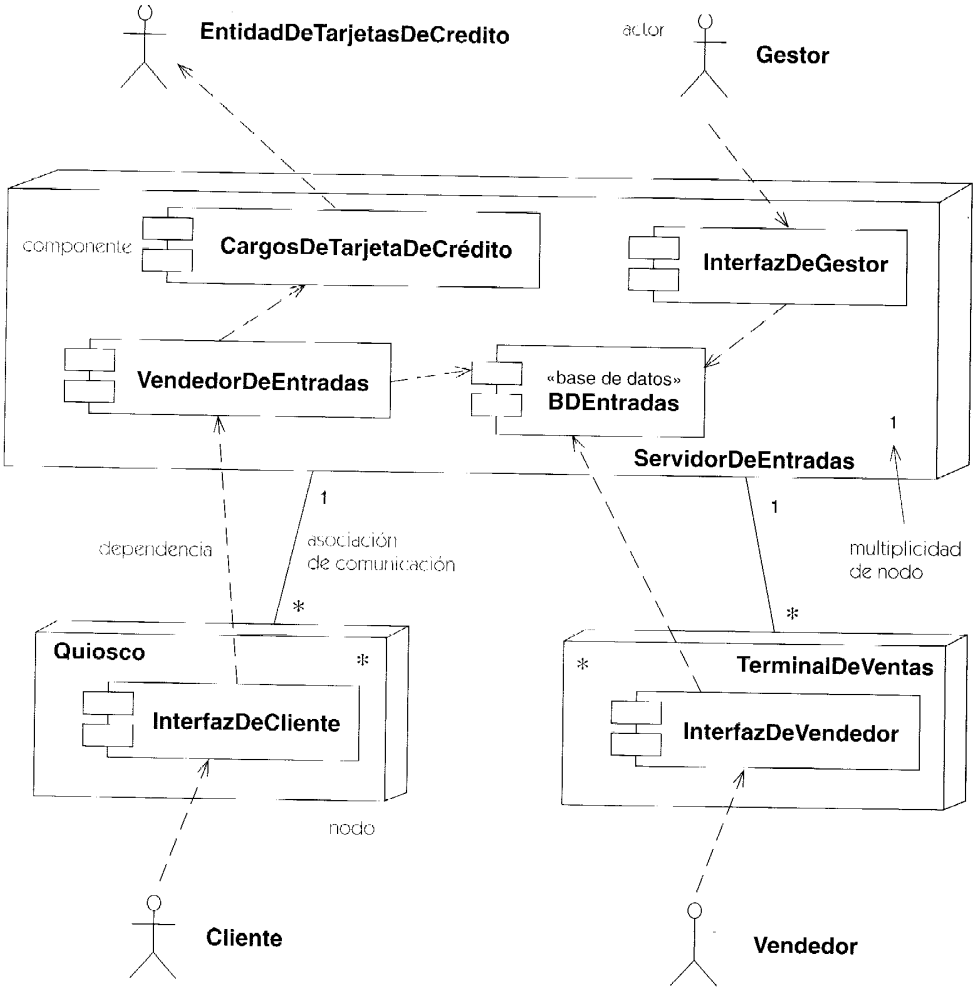


Figura 3.8 Diagrama de despliegue (nivel de descriptor)

Vista de gestión del modelo

La vista de gestión del modelo modela la organización del modelo en sí mismo. Un modelo abarca un conjunto de paquetes que contienen los elementos del modelo, tales como clases, máquinas de estados, y casos de uso. Los paquetes pueden contener otros paquetes: por lo tanto, un modelo señala un paquete raíz, que contiene indirectamente todo el contenido del modelo. Los paquetes son unidades para manipular el contenido de un modelo, así como unidades para el control de acceso y el control de configuración. Cada elemento del modelo pertenece a un paquete o a otro elemento.

Un modelo es una descripción completa de un sistema, con una determinada precisión, desde un punto de vista. Puede haber varios modelos de un sistema desde distintos puntos de vista; por ejemplo, un modelo de análisis y un modelo de diseño. Un modelo se representa como una clase especial de paquete.

Un subsistema es otro paquete especial. Representa una porción de un sistema, con una interfaz perfectamente determinada, que puede ser implementado como un componente distinto.

Generalmente, la información de gestión del modelo se representa en diagramas de clases.

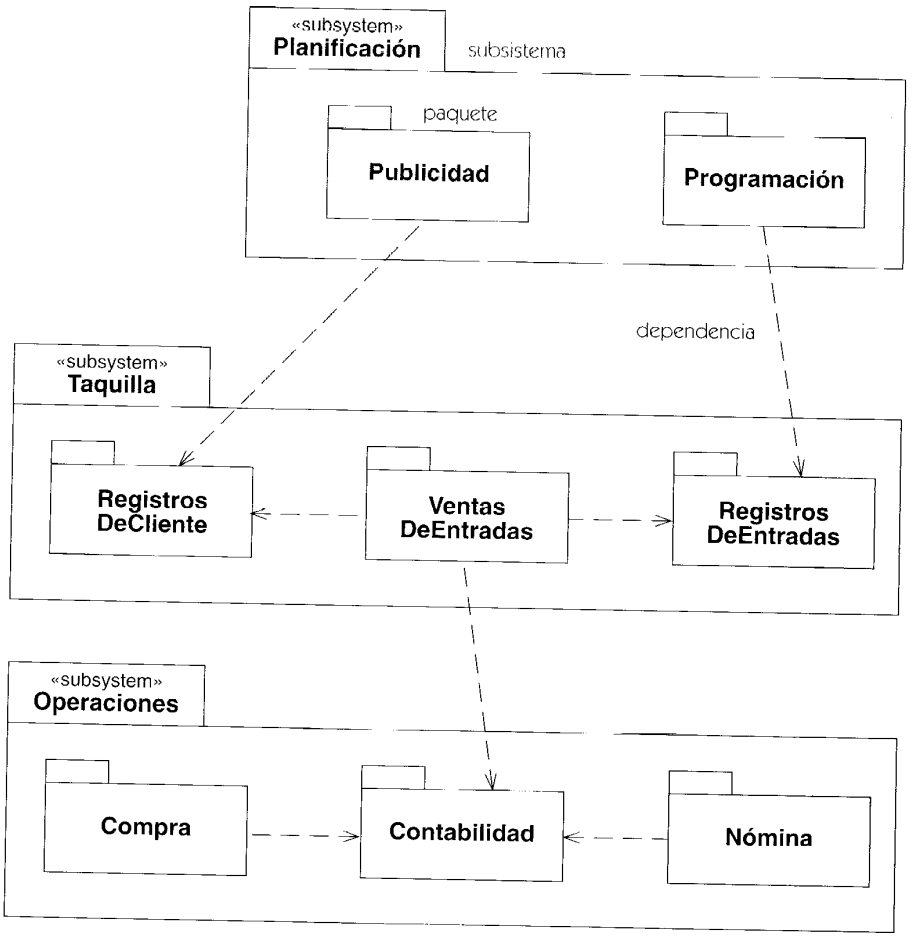


Figura 3.10 Paquetes

1. **Vista de Desarrollo (Development View)**:
   - La vista de desarrollo se centra en los aspectos internos del sistema desde la perspectiva de los desarrolladores.
   - Se utiliza para representar la estructura del sistema en términos de módulos, componentes, clases y sus relaciones.
   - Puede incluir diagramas como el diagrama de clases, diagrama de componentes, diagrama de paquetes, entre otros, para mostrar cómo se organiza y estructura el código del sistema.
2. **Vista de Despliegue (Deployment View)**:
   - La vista de despliegue describe cómo el sistema se despliega y se ejecuta en el entorno de producción.
   - Se centra en los elementos físicos del sistema, como servidores, dispositivos de red y otros recursos de hardware.
   - Utiliza diagramas de despliegue para representar la asignación de componentes del sistema a nodos físicos y la comunicación entre ellos, así como la configuración de hardware y software.

3. **Vista de Procesos (Process View)**:
   - La vista de procesos se centra en la dinámica del sistema y cómo los diferentes componentes interactúan entre sí para lograr los objetivos del sistema.
   - Se utiliza para modelar los procesos o flujos de trabajo que ocurren dentro del sistema, incluyendo interacciones entre objetos y el flujo de control entre componentes.
   - Puede incluir diagramas de secuencia, diagramas de actividad y diagramas de colaboración para representar los diferentes aspectos de la interacción entre los componentes del sistema durante la ejecución.

Estas vistas en UML proporcionan perspectivas específicas que permiten a los desarrolladores comprender y modelar diferentes aspectos del sistema de software, desde su diseño y desarrollo hasta su despliegue y funcionamiento en un entorno de producción.