UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

CENTRO DE CIENCIAS BÁSICAS

INGENIERÍA EN SISTEMAS COMPUTACIONALES

04-03-2022

METODOLOGÍAS PARA POO

MATERIA: SEMINARIO DE SISTEMAS COMPUTACIONALES I

DOCENTE: ARTURO ELIAS RAMIREZ

ALUMNOS:

BRAYAN DE JESÚS CAPETILLO MACIAS ID: 253311

GERARDO MARTINEZ MARTINEZ ID: 252410

GRADO y GRUPO: 8 A

Contenido

A. Metodologías dirigidas por datos 3

I. OMT 3

a) Objetivos del OMT 3

b) Fases del OMT 3

II. Fusión 5

a) Objetivos de fusión 5

b) Fases de fusión 5

B. Metodologías dirigidas por las responsabilidades 7

I. RDD 7

a) Objetivos del RDD 7

b) Términos de diseño en RDD 8

c) Estereotipos de roles en RDD 8

II. OBA 9

a) Objetivos del OBA 9

b) Orden de análisis y diseño del OBA 9

c) Pasos del OBA 10

C. Metodologías dirigidas por los casos de uso 13

I. Objectory (Jacobson et al., 1992) 13

a) Historia de Objectory 14

b) Etapas de Objectory 14

II. Proceso unificado 14

a) Características principales del proceso unificado 15

D. Metodologías dirigidas por estados (state-driven) 16

I. Metodologías de shlaer y Mellor (shlaer y Mellor, 1992) 16

a) Características 17

b) Procesos 17

Referencias 19

A. Metodologías dirigidas por datos (Driven – data): se basan en la parte estructural de los objetos y son una extensión del modelo conceptual en el modelo Entidad/Relación.

I. OMT (Object Modeling Technique) (Rumbaugh et al., 1991): OMT es una de las metodologías de análisis y diseño orientadas a objetos, más maduras y eficientes que existen en la actualidad. La gran virtud que aporta esta metodología es su carácter de abierta (no propietaria), que le permite ser de dominio público y, en consecuencia, sobrevivir con enorme vitalidad. Esto facilita su evolución para acoplarse a todas las necesidades actuales y futuras de la ingeniería de software.

La Técnica de Modelado de Objetos (OMT) es un enfoque de modelado basado en el mundo real para el modelado y diseño de software. Fue desarrollado básicamente como un método para desarrollar sistemas orientados a objetos y para apoyar la programación orientada a objetos. Describe la estructura estática del sistema.

La técnica de modelado de objetos es fácil de dibujar y usar. Se utiliza en muchas aplicaciones como telecomunicaciones, transporte, compiladores, etc. También se utiliza en muchos problemas del mundo real. OMT es una de las técnicas de desarrollo orientadas a objetos más populares utilizadas hoy en día. OMT fue desarrollado por James Rambaugh.

a) Objetivos del OMT:

1. Probar la entidad física antes de la construcción de los mismos.
2. Facilitar la comunicación con los clientes.
3. Presentar la información de una manera alternativa, es decir, la visualización.
4. Reducir la complejidad del software.
5. Resolver los problemas del mundo real.

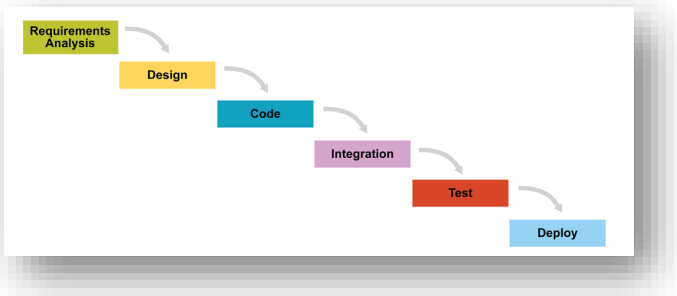
b) Fases del OMT:

1. Análisis. El analista construye un modelo del dominio del problema, mostrando sus propiedades más importantes. El modelo de análisis es una abstracción resumida y precisa de lo que debe de hacer el sistema deseado y no de la forma en que se hará. Los elementos del modelo deben ser conceptos del dominio de aplicación y no conceptos informáticos tales como estructuras de datos. Un buen modelo debe poder ser entendido y criticado por expertos en el dominio del problema que no tengan conocimientos informáticos.

2. Diseño del sistema. El diseñador del sistema toma decisiones de alto nivel sobre la arquitectura del mismo. Durante esta fase el sistema se organiza en subsistemas basándose tanto en la estructura del análisis como en la arquitectura propuesta. Se selecciona una estrategia para afrontar el problema.

3. Diseño de objetos. El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis, pero incorporando detalles de implementación. El diseño de objetos se centra en las estructuras de datos y algoritmos que son necesarios para implementar cada clase. OMT describe la forma en que el diseño puede ser implementado en distintos lenguajes (orientados y no orientados a objetos, bases de datos, etc.).

4. Implementación. Las clases de objetos y relaciones desarrolladas durante el análisis de objetos se traducen finalmente a una implementación concreta. Durante la fase de implementación es importante tener en cuenta los principios de la ingeniería del software de forma que la correspondencia con el diseño sea directa y el sistema implementado sea flexible y extensible. No tiene sentido que utilicemos AOO y DOO de forma que potenciemos la reutilización de código y la correspondencia entre el dominio del problema y el sistema informático, si luego perdemos todas estas ventajas con una implementación de mala calidad.



La metodología OMT emplea tres clases de modelos para describir el sistema:

Modelo de objetos. Describe la estructura estática de los objetos del sistema (identidad, relaciones con otros objetos, atributos y operaciones). El modelo de objetos proporciona el entorno esencial en el cual se pueden situar el modelo dinámico y el modelo funcional. El objetivo es capturar aquellos conceptos del mundo real que sean importantes para la aplicación. Se representa mediante diagramas de objetos.

Modelo dinámico. Describe los aspectos de un sistema que tratan de la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos) y la organización de sucesos y estados. Captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en que están implementadas. Se representa gráficamente mediante diagramas de estado.

Modelo funcional. Describe las transformaciones de valores de datos (funciones, correspondencias, restricciones y dependencias funcionales) que ocurren dentro del sistema. Captura lo que hace el sistema, independientemente de cuándo se haga o de la forma en que se haga. Se representa mediante diagramas de flujo de datos

II. Fusión (Coleman et al., 1994): Fusión es un método sistemático de desarrollo de software para el desarrollo de software orientado a objetos. Desarrollado en Hewlett-Packard Laboratorios en Bristol, Inglaterra, el método integra y extiende las mejores características de los métodos anteriores orientados a objetos. La fusión es un método de cobertura completa, que proporciona una ruta directa desde la definición de requisitos a través del análisis y el diseño hasta un lenguaje de programación y proporciona una implementación de un método de desarrollo de software orientado al objeto que abarca desde la definición de requisitos a la implementación en un lenguaje de programación, se considera metodología de segunda generación porque proviene de OMT, CRC, BOOCH y los métodos formales.

a) Objetivos de fusión:

* Proporciona un proceso para el desarrollo de software. Divide este proceso en fases y dice lo que se debe hacer en cada fase. Brinda orientación sobre el orden en que se deben hacer las cosas dentro de las fases para que el desarrollador sepa cómo avanzar.

* Proporciona criterios que le indican al desarrollador cuándo pasar a la siguiente fase.

* Proporciona una notación completa, simple y bien definida para todos sus modelos. Debido a que esta notación se basa en prácticas existentes, es fácil de aprender.

* Proporciona herramientas de gestión para el desarrollo de software. Los resultados de las diferentes fases están claramente identificados y se realizan verificaciones cruzadas para garantizar la coherencia dentro de las fases y entre ellas. Cada fase tiene sus propias técnicas y
* aborda diferentes aspectos de la traducción de un documento de requisitos en un código ejecutable.

* Es adaptable. Se puede usar una versión liviana en proyectos que no pueden permitirse el esfuerzo requerido para usar la versión completa, o se pueden usar partes del proceso o la notación dentro de otros procesos de desarrollo para abordar sus puntos débiles.

b) Fases de fusión:

1. Análisis: se describe lo que hace un sistema no como lo hace, en esta fase se capturan los requisitos del sistema viendo al sistema desde la perspectiva del usuario, casa con el dominio del problema y se preocupa por el comportamiento visible externamente, en esta fase se producen modelos: modelo de objetos y modelo de la interfase (modelo de funcionamiento y modelo del ciclo de vida).

2. Diseño: consiste en desarrollar un modelo abstracto de como el sistema lleva a cabo el comportamiento especifico en el análisis. El diseñador elige como construirlo y como los objetos se desarrollan entre ellos, esta fase se basa en CRC y Booch, en esta fase se

desarrollan cuatro modelos: gráficos de interacción de objetos, gráficos de visibilidad y descripciones de clases.

3. Implementación: El implementador debe convertir el diseño en código en un lenguaje de programación particular. Fusión brinda orientación sobre cómo se hace esto de las siguientes maneras: Los atributos de herencia, referencia y clase se implementan en clases de lenguaje de programación.

   Las interacciones de objetos se codifican como métodos que pertenecen a una clase seleccionada. Las secuencias de operaciones permitidas son reconocidas por máquinas de estado. Fusión también mantiene un diccionario de datos, un lugar donde se pueden nombrar y describir las diferentes entidades del sistema. Se hace referencia al diccionario de datos durante todo el proceso de desarrollo. En resumen, Fusión es un método de desarrollo completo pero ligero que se puede adaptar para satisfacer las diferentes necesidades de los proyectos de software.

B. Metodologías dirigidas por las responsabilidades (responsability – driven): representan el enfoque más purista de la orientación al objeto centrándose en las "responsabilidades" de los objetos, esto es, las acciones que puede llevar a cabo un objeto.

I. RDD (Responsability Driven Design) (Wirfs- Brock) et al., 1990) Relacioandas con las CRC, Responsibility-Driven Design (RDD) es una técnica de diseño orientado a objetos que se basa, pone un método, en las CRC, dice los pasos para trabajar con las CRC. Los autores de RDD son Rebecca Wirfs - Brock (muy popular en aquellos tiempos en que la OO estaba de moda) y Brian Wilkerson.

El objetivo es convertir requisitos en clases. El método se basa en las ideas "cliente – servidor", hay clases (u objetos para ser más exactos) que harán de cliente y otros de servidor. La idea es preguntarse "de qué acciones es responsable este objeto" y "qué información debe proporcionar".Lo importante del método es que obliga a pensar en los objetos según su responsabilidad y evitar pensar en datos.

a) Objetivos del RDD :

Según Rebecca Wirfs-Brock, RDD tiene 3 principales objetivos:

* Maximizar la abstracción
* Comportamiento de distribución
* Preservar el diseño de flexibilidad

A diferencia del Data-Driven Design, el Responsibility-Driven Design se centra en el contrato considerando las acciones de las que el objeto es responsable y la información que el objeto comparte y trata de evitar tratar con detalles, delegando el control para asignar responsabilidades a los elementos y por lo tanto cada elemento se reutilizaría muy rápidamente.

b) Términos de diseño en RDD :

Primero, echemos un vistazo a la terminología común cuando se habla de Diseño Impulsado por la Responsabilidad.

* Aplicación: conjunto de objetos que interactúan.
* Objeto: implementación de uno o más roles.
* Rol: conjunto de responsabilidades relacionadas.
* Responsabilidad: obligación de realizar una tarea o conocer información.
* Colaboración: interacción de objetos o roles (o ambos).
* Contrato: acuerdo que describe los términos de una colaboración.

Los objetos deben tener un propósito específico para jugar dentro de un sistema de software. Mientras tanto, los roles se pueden implementar utilizando interfaces y composición. Hoy analizamos algunos estereotipos de roles que han demostrado ser útiles a lo largo del tiempo.

c) Estereotipos de roles en RDD :

Definir responsabilidades es crucial para tener un buen diseño de software. Rebecca Wirfs-Brock describe algunos roles para describir diferentes tipos de implementaciones responsables para ayudarlo a definir elementos de software.

Una sola responsabilidad es mayor que una operación o atributo. Al definir las clases, viene a la mente el Principio de Responsabilidad Única. Cada clase debe hacer solo una tarea / responsabilidad, pero ¿cómo establecemos las responsabilidades?

Aquí es donde los estereotipos de roles son útiles.

Exploremos las categorías de estereotipos de roles:

* Controlador: toma decisiones y dirige de cerca la acción de otros objetos.
* Coordinador: reacciona a los acontecimientos delegando tareas a otros.
* Titular de la información: conoce y proporciona información.
* Proveedor de información: Una ligera variación de un titular de información que asume un papel más activo en la gestión y el mantenimiento de la información.
* Interfacer: transforma información y solicitudes entre distintas partes de una aplicación.
* External Interfacer: se comunica con otras aplicaciones en lugar de con las suyas propias.
* Internal Interfacer: actúa como un puente entre los vecindarios de objetos.
* User Interfacer: se comunica con los usuarios respondiendo a los eventos generados en la interfaz de usuario y, a continuación, pasándolos a objetos más apropiados.
* Proveedor de servicios: realiza trabajos y ofrece servicios informáticos.
* Estructurador: mantiene relaciones entre objetos e información sobre esas relaciones.

II. OBA (Object Behavior Analysis) (Rubin y Goldberg, 1992) El análisis orientado a objetos se esfuerza por modelar una situación en términos de una colección de entidades que interactúan, cada una de las cuales proporciona un conjunto bien definido de comportamientos y atributos. La mayoría de los enfoques publicados describen definiciones conceptualmente similares, aunque adoptan terminologías alternativas (2, 3, 10). Existe un alto grado de acuerdo sobre la estructura deseada del resultado final; diferimos en cómo llegar al resultado final.

Muchos enfoques recomiendan primero buscar los objetos tangibles, en particular buscando los sustantivos en una especificación de requisitos y verbos y adjetivos aplicables. Con los sustantivos como objetos, la interfaz del mensaje se determina a partir de los verbos, y las propiedades lógicas se derivan de los adjetivos. Aunque este enfoque básico puede funcionar para sistemas pequeños, es nuestra experiencia que simplemente no se ampliará. En primer lugar, asume que existe una especificación de requisitos completa, formal y correcta. Es casi seguro que esto no es cierto para los sistemas grandes. Además, este enfoque tiene un fuerte sesgo hacia los aspectos tangibles de un problema (es decir, aquellas cosas que se pueden ver, escuchar, sentir, oler y probar). Los objetos tangibles a menudo son importantes para reconocer y capturar. Pero, con la misma frecuencia, los objetos conceptuales tienen una influencia significativa en la estructura de los resultados del análisis. Los sustantivos y verbos son a menudo una guía insuficiente para localizar este tipo de objetos.

a) Objetivos del OBA

   El objetivo de OBA es, en primer lugar, comprender la descripción del problema y, en segundo lugar, formular esta descripción en términos de múltiples objetos que interactúan. Estos objetos cumplen funciones y responsabilidades del sistema al proporcionar y contratar servicios bien definidos que llevan a cabo comportamientos del sistema. El resultado del análisis debe ser comprensible para el usuario final, prestarse a un mayor diseño e implementación, y ser rastreable a las metas y objetivos del sistema. Sobre la base de OBA, se pueden formular estimaciones para el desarrollo restante del proyecto en función del número de comportamientos identificados, participantes e iniciadores, y las relaciones entre estas diversas partes.

b) Orden de análisis y diseño del OBA

   La cuestión de por dónde empezar está en el corazón de la distinción entre varios enfoques de análisis. Nuestro objetivo es poder responder a la pregunta: ¿Qué roles y responsabilidades se necesitan para realizar las tareas requeridas? Además, debemos responder por qué existe un objeto en particular, por qué está vinculado a otro objeto, por qué un objeto proporciona un servicio en particular y cómo el objeto participa en el cumplimiento de los requisitos funcionales. Cada paso de OBA contribuye a la explicación o es una fuente de información utilizada para determinar los resultados finales. Como tal, cada uno de los pasos de OBA en última instancia debe completarse.

   OBA se puede caracterizar como un enfoque iterativo, pero uno con múltiples puntos de entrada. Dentro de este artículo, los pasos de OBA se establecen de manera lineal para fines de exposición. En la práctica, se utilizan tanto iterativamente, dentro de una sola parte de un proyecto, como en paralelo, en múltiples partes de un proyecto. Cuando completamos un paso, lo hacemos creyendo que hemos reunido suficiente información o resultados para el siguiente paso. Al pasar al siguiente paso, podemos descubrir que falta información o que surgen nuevas preguntas. Para resolver

estos problemas, volvemos a iterar. Los pasos de OBA están diseñados específicamente para proporcionar esta forma de control y equilibrio, para verificar que el contexto creciente del análisis sea internamente consistente en todos los pasos.

En la sección anterior, argumentamos a favor de iniciar el proceso de análisis con un enfoque en el comportamiento. En varias situaciones, sin embargo, esto podría no ser posible. Por ejemplo, cualquiera de los siguientes podría haberse completado antes de la decisión de aplicar OBA:

- ❖ Un enfoque no orientado a los objetos y orientado a los datos ha producido un modelo de datos.
- ❖ Un análisis a nivel de toda la empresa ha definido un vocabulario común para funciones y/o datos.
- ❖ Se ha completado un análisis de dominio con los objetos básicos que se proponen.

A medida que vayamos explicando cada paso, iremos anotando posibles puntos de entrada alternativos en función de contar con la información de estas situaciones.

El análisis del comportamiento del objeto (OBA) consta de cinco pasos:

- ∗ Establecer el contexto para el análisis
- ∗ Comprender el problema centrándose en los comportamientos
- ∗ Definir objetos que exhiben comportamientos
- ∗ Clasificar objetos e identificar sus relaciones
- ∗ Modelar la dinámica del sistema

c) Pasos del OBA

Etiquetamos los cinco pasos de OBA como Pasos 0 a 4, enfatizando el primer paso con la etiqueta inusual "cero" para llamar la atención sobre el hecho de que este paso a menudo está fuera del alcance de lo que tradicionalmente se llama análisis.

Paso 0: Establecer el contexto de análisis

El paso 0 consta de cuatro subpasos que identifican metas y objetivos, recursos apropiados para el análisis, áreas de actividad centrales y un plan de análisis preliminar. Llevar a cabo estos subpasos constituye la base del contexto en el que llevamos a cabo el análisis. En particular, el subpaso 0.1 identifica metas y objetivos, que son declaraciones del resultado deseado del sistema. El subpaso 0.2 sirve para identificar recursos -documentos, diccionarios, así como expertos en usuarios finales y dominios- que pueden contribuir al esfuerzo de análisis. El siguiente subpaso, el subpaso 0.3, consiste en identificar las áreas de actividad principales. Estas son las principales áreas del sistema que requieren análisis. La identificación de estas áreas proporcionará una base para el proceso de scripting (discutido en el Paso 1) y para la partición de trabajo y el desarrollo paralelo. Una forma de identificar estas áreas es esbozar el ciclo de vida prístino del sistema. Por lo general, se trata de un orden secuenciado en el tiempo de las principales actividades que se producen en un dominio de problema.

La última tarea, Substep 0.4, es generar un plan de análisis preliminar. El plan toma la partición de las áreas de actividad básicas, establece prioridades y hace estimaciones de tiempo y recursos para las actividades de análisis. Este plan está integrado en el plan maestro del proyecto.

Paso 1: Comprender el problema Una vez que se ha establecido el contexto, el siguiente paso es determinar qué se supone que debe hacer el sistema, y para quién y con quién se supone que debe hacerlo. La idea básica es especificar escenarios de uso que cubran todas las vías posibles a través de las funciones del sistema. (El enfoque de la Objeción de Jacobson también sugiere una técnica similar que él llama Casos de Uso)

Un enfoque para obtener los escenarios de uso es a través de un proceso de entrevista estructurado. Por lo general, se entrevista a dos tipos de personas: los usuarios y los expertos en el dominio. Los usuarios son las personas que realizan algunas de las actividades en el sistema actual, o que realizarán actividades en el sistema a construir. Los expertos varían, dependiendo del tipo de sistema. Generalmente, los expertos incluyen a las personas que han patrocinado el trabajo y que tienen expectativas sobre el sistema, consultores que trabajan en el dominio y se consideran conocedores, así como otros desarrolladores que son expertos en la construcción de sistemas del mismo tipo.

Paso 2--Definir objetos

Hasta este punto, hemos creado escenarios de uso que describen actividades básicas, y cuyo contenido se basa en entrevistas, construcciones de ejemplo o referencias (expertos o materiales escritos). Hemos capturado estos escenarios de cómo debería funcionar thing2 y los hemos presentado en forma de scripts. Los scripts están vinculados entre sí haciendo coincidir las condiciones posteriores con las condiciones previas, lo que proporciona una imagen más amplia de cómo podrían progresar las acciones en el sistema. Al hacerlo, identificamos una serie de partes que actúan como iniciadores, participantes o ambos. Además, identificamos los contratos y los servicios requeridos a los participantes, así como las propiedades lógicas de los (4) asumimos que el

análisis se lleva a cabo en el contexto de un modelo de proceso de proyecto que incluye un sistema de revisiones periódicas.

Paso 3-- Clasificar objetos e identificar relaciones

Las tareas del paso 3 implican la aplicación de un conjunto de técnicas para identificar las relaciones entre objetos. La aplicación de estas técnicas nos permite rellenar los espacios en blanco sobrantes del Paso 2 para completar las Tarjetas de Modelado de Objetos:

* Servicios Contratados
* Rastro de Tarjeta

Hereda Del Propósito del Subpaso 3.1 es describir las relaciones contractuales entre los objetos. La inclusión de contratos en la Tarjeta de Modelado de Objetos tiene dos propósitos. En primer lugar, nos permite derivar las relaciones entre este objeto y otros objetos del sistema. En términos orientados a objetos, esta relación contractual es esencialmente una declaración de que el objeto envía un mensaje a otro objeto con el fin de obtener información, proporcionar información, solicitar acciones o notificar que ha ocurrido algún evento. El segundo propósito es evitar errores que resultan cuando un objeto espera un servicio de otro, pero ningún objeto ha asumido la responsabilidad de ese servicio.

Para cada tarjeta de modelado de objetos creada en el paso 2, capturamos los contratos que el objeto espera que cumplan otros. Una ilustración de las relaciones contractuales, parcialmente derivada de las tarjetas de modelado de objetos para el ejemplo de la hoja de cálculo, se muestra en el diagrama de la Figura 4. Para proporcionar este y los ejemplos posteriores, asumimos que se han creado más scripts y se ha identificado un conjunto más completo de objetos.
Ahora estamos listos para el Subpaso 3.2 en el que aplicamos varias técnicas a las que nos referimos como técnicas de reorganización. El objetivo es determinar:

* servicios comunes a dos o más objetos, y crear un objeto que capture la descripción compartida de estos servicios
* propiedades lógicas comunes a dos o más objetos (mediante el examen de glosarios de atributos) y, de nuevo, crear un objeto que capture la descripción compartida de estos atributos
* los servicios o propiedades lógicas de un objeto se pueden describir como un refinamiento de los servicios o propiedades lógicas de otro objeto.
* un objeto al que se le asignan múltiples responsabilidades (en términos de sus servicios prestados), y para factorizarlas en un objeto separado para cada área de responsabilidad.

Las dos primeras técnicas se llaman abstracción, la tercera especialización, y el cuarto es la factorización. Como resultado de la aplicación de estas técnicas, creamos nuevos objetos y sus tarjetas de modelado asociadas.

Paso 4: Modelar ciclos

de vida del sistema Hasta este punto, hemos tratado con vistas estáticas del sistema que estamos analizando. Estos identifican la estructura del sistema en un solo punto en el tiempo, es decir, qué comportamientos contiene el sistema, qué objetos son responsables de estos comportamientos y cualquier relación entre los objetos.

El paso 4 de OBA se refiere al modelado de la dinámica del sistema, es decir, aquellos aspectos del sistema que cambian con el tiempo. El sistema llevará a cabo comportamientos en respuesta a eventos, en un orden prescrito. Los estados de objeto, los eventos y el orden en que ocurren los comportamientos deben estar muy representados.

Los estados asociados a un objeto se definen en el subpaso 4.1. Los estados se utilizan para representar una situación o condición de un objeto durante la cual se aplican ciertas leyes, reglas y políticas físicas (esta definición proviene de (12)). Los cambios en el estado suelen dar lugar a cambios en el comportamiento de uno o más objetos del sistema. Supongamos, por ejemplo, que nuestro scripting indica que existe una aplicación para cargar y guardar hojas de cálculo. Cada vez que el usuario intenta salir, la aplicación determina si la hoja de cálculo se modificó o no desde la última vez que se guardó y, de ser así, ofrece guardar antes de salir. Por lo tanto, el estado de la hoja de cálculo, modificado o no, afecta el comportamiento de la aplicación.

Los estados de los objetos se determinan a partir de expresiones de pre y postcondición de script. Las expresiones se componen de una colección de cláusulas, cada una de las cuales se compone a su vez de una descripción de estado y un par de objetos. El primer paso para determinar los estados interesantes de un objeto es buscar en todas las expresiones previas y posteriores condiciones las cláusulas que contienen el objeto. La definición estatal no está completa hasta que se hayan considerado todas esas cláusulas. Por el contrario, si tenemos conocimiento del dominio del problema que indica que no se ha tenido en cuenta una condición de estado, tenemos evidencia para creer que el scripting no se ha completado, y debemos iterar de nuevo al Paso 1.

En el Subpaso 4.3, determinamos la secuenciación de las operaciones dentro del sistema, también conocida como flujo de control. Definimos el flujo de control como el aspecto de un sistema que describe las secuencias de operaciones que ocurren en respuesta a un evento. Hasta este punto, hemos estado trabajando bajo el supuesto de que el orden predeterminado es secuencial. Esto es inherente a la notación que elegimos para los guiones, que pasa a presentarlos de una manera que conduce a una interpretación secuencial. Sin embargo, hay muchos otros pedidos que pueden ser apropiados y / o requeridos. Por ejemplo, las líneas de un script podrían ejecutarse de forma simultánea, repetitiva, selectiva u opcional.

C. Metodologías dirigidas por los casos de uso (use case – driven): Estos se utilizan para obtener los requisitos funcionales del sistema y así definir el contenido de cada una de las iteraciones. Así, la idea consiste en coger casos de uso o escenarios y desarrollar el proceso a través de las distintas disciplinas (diseño, implementación, pruebas…)

I. Objectory (Jacobson et al., 1992)

Objectory es una metodología orientada a objetos creada principalmente por Ivar Jacobson, quien ha contribuido en gran medida a la ingeniería de software orientada a objetos.El marco de la objeción es una técnica de diseño llamada diseño con bloques de construcción. Con la técnica de bloques de construcción, un sistema se ve como un sistema de bloques de conexión con cada bloque que representa un servicio del sistema.

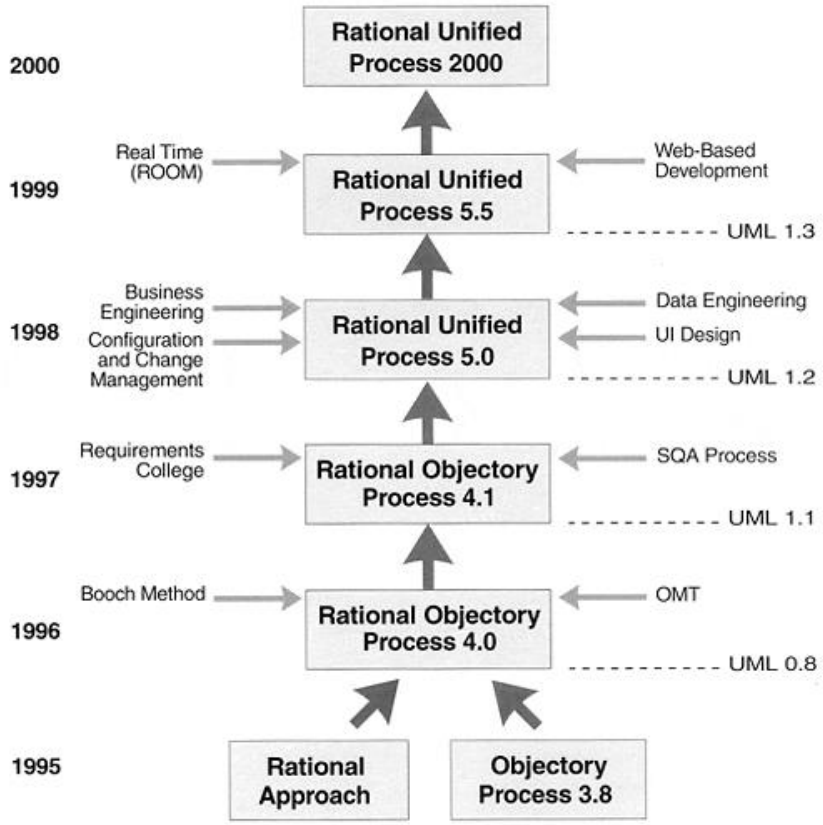
Se considera que es la primera metodología orientada a objetos disponible comercialmente para el desarrollo de sistemas industriales

a gran escala. Este enfoque ofrece una visión global del desarrollo de software y se centra en la rentabilidad. Sus principales técnicas son: modelado conceptual, programación orientada a objetos y una técnica de diseño de bloques.

a) Historia de Objectory

El proceso Objectory fue creado en Suecia en 1987 por Ivar Jacobson como resultado de sus muchos años de experiencia en el fabricante sueco de telecomunicaciones Ericsson AB. Este proceso se convirtió en un producto en su empresa, Objectory AB. Centrado en el concepto de caso de uso y diseño orientado a objetos, rápidamente ganó reconocimiento en la industria del software y ha sido adoptado e integrado por muchas empresas en todo el mundo. Una versión simplificada del proceso Objectory fue publicada como libro de texto en 1992.

b) Etapas de Objectory



II. Proceso unificado

El Proceso Unificado de Desarrollo Software o simplemente Proceso Unificado es un marco de desarrollo de software que se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental. El refinamiento más conocido y documentado del Proceso Unificado es el Proceso Unificado de Rational (RUP) o simplemente UP.

El Proceso Unificado no es simplemente un proceso, sino un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos. De la misma forma, el Proceso Unificado de Rational, también es un marco de trabajo extensible, por lo que muchas veces resulta imposible decir si un refinamiento particular del proceso ha sido derivado del Proceso Unificado o del RUP. Por dicho motivo, los dos nombres suelen utilizarse para referirse a un mismo concepto.

El nombre Proceso Unificado se usa para describir el proceso genérico que incluye aquellos elementos que son comunes a la mayoría de los refinamientos existentes. También permite evitar problemas legales ya que Proceso Unificado de Rational o RUP son marcas registradas por IBM (desde su compra de Rational Software Corporation en 2003). El primer libro sobre el tema se denominó, en su versión española, El Proceso Unificado de Desarrollo de Software (ISBN 84-7829-036-2) y fue publicado en 1999 por Ivar Jacobson, Grady Booch y James Rumbaugh, conocidos también por ser los desarrolladores del UML, el Lenguaje Unificado de Modelado. Desde entonces los autores que publican libros sobre el tema y que no están afiliados a Rational utilizan el término Proceso Unificado, mientras que los autores que pertenecen a Rational favorecen el nombre de Proceso Unificado de Rational.

a) Características principales del proceso unificado

Iterativo e Incremental

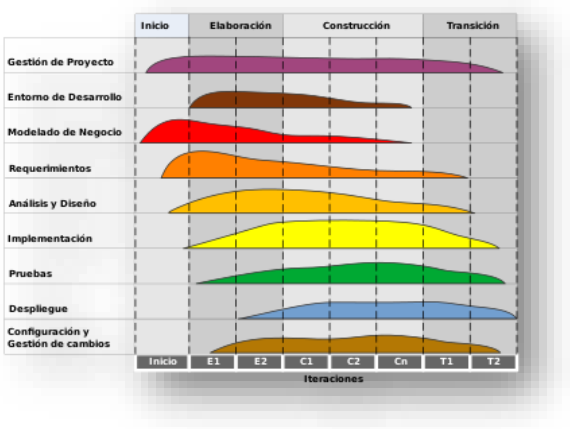
El Proceso Unificado es un marco de desarrollo iterativo e incremental compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones (la de inicio puede incluir varias iteraciones en proyectos grandes). Estas iteraciones ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

Cada una de estas iteraciones se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: Análisis de requisitos, Diseño, Implementación y Prueba. Aunque todas las iteraciones suelen incluir trabajo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto.

Diagrama ilustrando como el énfasis relativo en las distintas disciplinas cambia a lo largo del proyecto.

Dirigido por los casos de uso

En el Proceso Unificado los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc. El proceso dirigido por casos de uso es el rup. Nota: en UP se está Dirigido por requisitos y riesgos de acuerdo con el Libro UML 2 de ARLOW, Jim que menciona el tema.



Centrado en la arquitectura

El Proceso Unificado asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc.

Enfocado en los riesgos

El Proceso Unificado requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero.

D. Metodologías dirigidas por estados (state-driven)

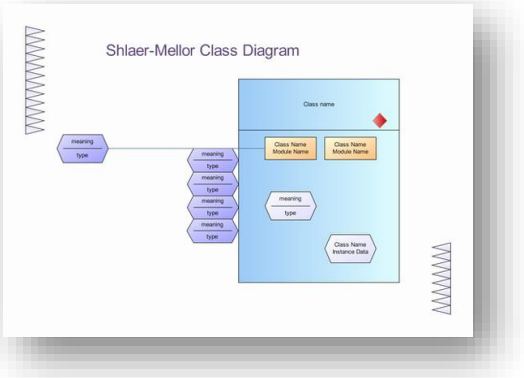
I. Metodologías de shlaer y Mellor (shlaer y Mellor, 1992)

El método Shlaer-Mellor, también conocido como Análisis de Sistemas Orientados a Objetos (OOSA) o Análisis Orientado a Objetos (OOA) es una metodología de desarrollo de software orientada a objetos introducida por Sally Shlaer y Stephen Mellor en 1988. El método hace que el análisis documentado sea tan preciso que es posible implementar el modelo de análisis directamente mediante la traducción a la arquitectura de destino, en lugar de elaborar cambios de modelo a través de una serie de modelos más específicos de la plataforma. En el nuevo milenio el método Shlaer-Mellor ha migrado a la notación UML, convirtiéndose en EXEcutable UML.

a) Características

El método Shlaer-Mellor es una de una serie de metodologías de desarrollo de software que llegaron a finales de la década de 1980. Los más familiares fueron el Análisis y Diseño Orientado a Objetos (OOAD) de Grady Booch, la Técnica de Modelado de Objetos (OMT) de James Rumbaugh, la Ingeniería de Software Orientada a Objetos de Ivar Jacobson y el Análisis Orientado a Objetos (OOA) de Shlaer y Mellor. Estos métodos habían adoptado un nuevo paradigma orientado a objetos para superar las debilidades establecidas en los métodos existentes de análisis estructurado y diseño estructurado (SASD) de las décadas de 1960 y 1970. De estos problemas bien conocidos, Shlaer y Mellor eligieron abordar:

La complejidad de los diseños generados a través del uso de métodos de análisis estructurado y diseño estructurado (SASD).
El problema de mantener la documentación de análisis y diseño a lo largo del tiempo.
Antes de la publicación de su segundo libro en 1991, Shlaer y Mellor habían dejado de nombrar su método "Análisis de sistemas orientados a objetos" en favor de simplemente "Análisis orientado a objetos". El método comenzó centrándose en el concepto de diseño recursivo (RD), que permitió el aspecto de traducción automática del método.



Lo que hace que Shlaer-Mellor sea único entre los métodos orientados a objetos es:

el grado en que se toma la descomposición semántica orientada a objetos,
la precisión de la notación de Shlaer-Mellor utilizada para expresar el análisis, y
el comportamiento definido de ese modelo de análisis en tiempo de ejecución.

b) Procesos

Traducción v. elaboración

El objetivo del método Shlaer-Mellor es hacer que el análisis documentado sea tan preciso que sea posible implementar el modelo de análisis directamente mediante traducción en lugar de mediante elaboración. En la terminología de Shlaer-Mellor esto se llama diseño recursivo. En la terminología actual (2011), diríamos que el método Shlaer-Mellor utiliza una forma de arquitectura basada en modelos (MDA) normalmente asociada con el Lenguaje unificado de modelado (UML).

Descomposición semántica

Shlaer-Mellor propone una descomposición semántica en múltiples dominios (problemáticos). La división entre modelos de análisis y diseño: El dominio de análisis expresa precisamente lo que el sistema debe hacer, el dominio de diseño es un modelo de cómo funciona la máquina virtual

Shlaer-Mellor para una plataforma de hardware y software en particular. Estos modelos son disjuntos, siendo la única conexión la notación utilizada para expresar los modelos.
Descomposición dentro del dominio de análisis donde los requisitos del sistema se modelan y agrupan en torno a temas específicos y disjuntos. Para volver al ejemplo anterior del tren de pasajeros, se pueden crear modelos semánticos individuales basados en actuadores de puertas, controles de motores y sistemas de frenado. Cada agrupación se considera y modela de forma independiente. La única relación definida entre las agrupaciones son las dependencias, por ejemplo, la aplicación de un tren de pasajeros puede depender tanto del accionamiento de la puerta como del control del motor. Los sistemas de frenado pueden depender del control del motor.

Lenguaje de acción preciso

Uno de los requisitos para la generación automatizada de código es modelar con precisión las acciones dentro de las máquinas de estado finito utilizadas para expresar el comportamiento dinámico de los objetos Shlaer-Mellor.
Shlaer-Mellor es único entre los métodos de análisis orientado a objetos en la expresión de un comportamiento secuencial de forma gráfica como los diagramas de flujo de datos de acción (ADFD). En la práctica, las herramientas que soportaban Shlaer-Mellor, proporcionaban un lenguaje de acción preciso. Los lenguajes de acción reemplazaron el enfoque ADFD, por lo que todas las acciones están escritas en forma textual.

Prueba y simulación

El enfoque translativo del método Shlaer-Mellor se presta a entornos automatizados de prueba y simulación (cambiando la plataforma de destino durante la generación de código), y esto puede explicar en parte la popularidad de Shlaer-Mellor y otros métodos basados en MDA al desarrollar sistemas integrados, donde las pruebas en sistemas de destino, por ejemplo, teléfonos móviles o sistemas de gestión de motores, son particularmente difíciles.

Referencias

[1] A.A. (s. f.). Roles in Responsibility-Driven Design. Apium Academy. Recuperado 5 de marzo de 2022, de https://apiumacademy.com/blog/roles-responsibility-driven-design/

[2] Beck, K. and Cunningham, W. A laboratory for teaching object-oriented thinking. In OOPSLA '89 Conference Proceedings, ACM SIGPLAN Note 24, 10 (Oct. 1989).

[3] Booch, G. Object Oriented Design with Applications. Benjamin/Cummings Inc., Redwood City, Calif., 1991.

[4] Coad, P. and Yourdon, E. Object-Oriented Analysis. Yourdon Press, Englewood Cliffs, N.J., 1990.

[5] Gilb, T. Principles of Software Engineering Management. Addison-Wesley, Readin.g, Mass., 1988.

[6] Harel, D. Statecharts: A visual formalism for complex systems. In Science of Computer Programming. Vol. 8, No. 3, North Holland, 1987, pp. 231-274,

[7] 6. IFPUG. International Function Point Users Group: Function Point Counting Practices Manual. Release 3.1, Jan. 1991.

[8] Jacobson, I. Object-Oriented Software Engineering. Addison-Wesley, Reading, Mass., 1992.

[9] Kowal, J.A. Behavior Models: Specifying User's Expectations. Prentice Hall, Englewood Cliffs, N.J., 1992.

[10] ParcPlace Systems. Object-Oriented Methodology Course Notes. ParcPlace Systems, Inc., Sunnyvale, Ca., 1992.

[11] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, N.J., 1991.

[12] Scott, A.C., Clayton, J.E., and Gibson, E.L. A Practical Approach to Knowledge Acquisition. AddisonWesley, Reading, Mass., 1991.

[13] Shlaer, S., and Melior, S.J. Object Lifecycles: Modeling the World in States. Yourdon Press, Englewood Cliffs, N.J., 1992.

[14] Wirfs-Brock, R., Wilkerson, B. and Wiener, L. Designing Object-Oriented Software, Prentice-Hall, Englewood Cliffs, N.J., 1990.

[15] Ivar Jacobson et al., Object-Oriented Software Engineering: A Use-Case-Driven Approach . Reading, MA: Addison-Wesley, 1992.

[16] This software-engineering-related article is a stub. You can help Wikipedia by expanding it.

[17] I.V.A.R.J.A.C.O.B.S.O.N. (s. f.-b). Objectory. https://www.ivarjacobson.com/. Recuperado 5 de marzo de 2022, de https://www.ivarjacobson.com/resource/books

[18] Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener, Designing Object-OrientedSoftware, Prentice Hall, 1990.

[19] Bertrand Meyer, “Programming as Contracting,” Interactive Software Engineering, Inc.Technical Report, 1988.

[20] Kent Beck and Ward Cunningham, “A Laboratory for Teaching Object Oriented Thinking,”OOPSLA ‘89 Conference Proceedings, SIGPLAN Notices, 24(10), pp. 1-6, New Orleans,Louisiana, October 1989