"Estándares para el desarrollo de software orientado a objetos"

Realizado por: Tania A. Acosta H.
Dirigido por: Ing. Raúl Córdova

Abstract

The object oriented is configured as the methodology of software development prevailing for the next years, however we do not have an standard for its development. The present work outlines a standard methodology for objects oriented software development , defined from the study of three major object oriented methods: Booch, OMT and OOSE. The methodology covers the planning, analysis, design, implementation, tests and software maintenance object oriented phases.

Resumen

La orientación a objetos se configura como la metodología de desarrollo de software predominante para los próximos años, sin embargo no se cuenta con estándares para su desarrollo. El presente trabajo plantea una metodología estándar para el desarrollo de software orientado a objetos, definida a partir del estudio de tres principales métodos: Booch, OMT, OOSE. La metodología cubre las fases de planificación, análisis, diseño, implementación, pruebas y mantenimiento orientado a objetos.

Introducción

Desde la década pasada la tecnología orientada a objetos se ha ido perfeccionando y se la considera como la solución a los problemas de esta década para la construcción rápida de sistemas complejos, confiables y extensibles, capturando la atención de los sectores académicos e industriales. A fin de asegurar un software eficiente de gran escala, los ingenieros de software han concentrado esfuerzos en la definición de técnicas y metodologías para producir software de alta calidad y en el desarrollo de herramientas automatizadas de soporte.

Varios métodos planteados con base en la orientación a objetos, difieren en su notación, terminología, pero son similares con respecto a sus nociones subyacentes básicas, pero hasta la actualidad no se han definido completamente.

La importancia de trabajar sobre la base de un estándar para el desarrollo de software orientado a objetos, se refleja en los numerosos empeños que han realizado diferentes autores.

El primer propósito del presente trabajo es proponer una terminología estándar orientada a objetos, analizando los principales métodos para el análisis y diseño de software en esta línea como son los métodos de Booch, Rumbaugh (OMT) y Jacobson,

El segundo propósito es proponer como estándar para el modelamiento de software orientado a objetos, el lenguaje de modelamiento unificado (UML).

El tercer propósito es definir una metodología estándar para el desarrollo de software orientado a objetos que cubra todas las fases del ciclo de vida, esto es, las fases de análisis, diseño, implementación, pruebas, mantenimiento y que incluya las administración de proyectos orientado a objetos.

Métodos

Se ha utilizado el método de investigación deductivo para el desarrollo de la presente tesis, analizando las diferentes concepciones de los tres más importantes autores sobre la tecnología orientada a objetos.

Resultados

El resultado de la presente tesis es plantear un estándar para las fases de planificación, análisis, diseño, implementación, pruebas, y mantenimiento de software orientado a objetos.

1.- Planificación

Para la fase de planificación se plantea realizar las siguientes tareas para el plan de proyecto de software[10]:

1.- Introducción
- 1.1.- Propósito del plan
- 1.2.- Ámbito del proyecto y objetivos
  - 1.2.1.- Declaración del ámbito
  - 1.2.2.- Funciones principales
  - 1.2.3.- Aspectos de rendimiento
  - 1.2.4.- Restricciones técnicas y de gestión

2.- Estimación del proyecto
- 2.1.- Datos históricos usados para las estimaciones
- 2.2.- Técnicas de estimación
- 2.3.- Estimaciones de esfuerzo, coste, duración

3.- Estrategia de gestión del riesgo
- 3.1.- Tabla de riesgo
- 3.2.- Estudio de los riesgos a tratar
- 3.3.- Plan RSGR para cada riesgo
  - 3.3.1.- Reducción del riesgo
  - 3.3.2.- Supervisión del riesgo
  - 3.3.3.- Gestión del riesgo

4.- Planificación temporal
- 4.1.- Estructura de descomposición del trabajo del proyecto
- 4.2.- Red de tareas
- 4.3.- Gráfico de tiempo (Gráfico Grantt)
- 4.4.- Tabla de recursos

5.- Recursos del proyecto
- 5.1.- Personal
- 5.2.- Hardware y software
- 5.3.- Tabla de recursos

6.- Organización del personal
- 6.1.- Estructura de equipos
- 6.2.- Informes de gestión

7.- Mecanismos de seguimiento y control
- 7.1.- Garantía de calidad y control
- 7.2.- Gestión y control de cambios

8.- Apéndices

2.- Análisis

El análisis es un proceso secuencial, y como el proceso global de desarrollo de software se caracteriza por ser un proceso iterativo, permite incrementar y depurar la información de los diferentes modelos conforme aumenta el conocimiento del dominio del problema[4]; para realizar esto se propone usar como estándar el lenguaje de modelamiento unificado (UML).
En esta etapa se especifica el sistema modelándolo en tres perspectivas[11]:

- Especificación de Datos o estática, que comprende los datos del dominio del problema
- Especificación Funcional, identifica los procesos

- Especificación Dinámica, analiza como el sistema o los objetos del sistema van variando conforme pasa el tiempo

➢ **Especificación de datos o estática**

El producto obtenido durante esta fase es el Modelo de Objetos, este modelo nos permite identificar una vista estática del sistema[6], siguiendo los siguientes pasos:

- **Identificar objetos y clases:** Es necesario seguir los siguientes pasos:
  - Seleccionar los nombres o sustantivos en el documento de requerimientos que el cliente entrega inicialmente
  - Añadir clases adicionales procedentes de nuestro conocimiento del tema
  - Eliminar redundancias
  - Eliminar clases irrelevantes
  - Eliminar clases vagas
  - Separar atributos
  - Separar métodos
  - Eliminar objetos de diseño

El diccionario de datos es uno de los elementos mas importantes dentro del desarrollo de software, y no solamente en el análisis, de allí que es necesario definir un diccionario de clases describiéndolas en lenguaje natural [11].

- **Identificar y depurar relaciones:** Consta de los siguientes pasos[12]:
  - Seleccionar verbos relacionales en los requisitos
  - Añadir relaciones adicionales procedentes de nuestro conocimiento del tema
  - Eliminar relaciones de diseño o entre clases eliminadas
  - Eliminar eventos transitorios
  - Reducir relaciones ternarias,
  - Eliminar relaciones redundantes o derivadas
  - Añadir relaciones olvidadas
  - Definir la multiplicidad de cada relación

El resultado de estas operaciones constituye la base del modelo de clases sin herencia.

- **Identificar atributos de objetos y relaciones**

Se deben seguir los siguientes pasos:

- Distinguir los objetos de los atributos
- Distinguir entre los atributos de objetos y de relaciones
- Eliminar atributos privados (de diseño
- Eliminar atributos no relevantes durante la fase de análisis.
- Localizar atributos discordantes
- Una vez identificado los atributos de los objetos, se define los atributos de las relaciones

- **Añadir herencia :** La relación de herencia permite introducir clases nuevas o virtuales, que contienen información común a dos o más clases preexistentes. Es preferible evitar la herencia múltiple, a menos que sea estrictamente necesaria.

- **Comprobar los casos de uso (iterar)**

Con el fin de identificar posibles errores a corregirse, es necesario observar en el modelo lo siguiente:

- Asimetrías en las relaciones: añadir clases nuevas para equilibrarlas.
- Atributos muy dispares entre ellos: descomponer una clase en dos.
- Dificultades en la formación de superclases: descomponer una clase en dos.
- Operaciones sin objetivo: añadir clase.
- Relaciones duplicadas: crear superclase.
- Conversión de relaciones en clases: por ejemplo, clase *Empleado*.
- Operaciones que no encuentran una manera para ejecutarse: añadir relaciones.
- Relaciones redundantes: eliminarlas.
- Relaciones demasiado detalladas o demasiado vagas: incorporarlas (ascenderlas)a una superclase o (descenderlas) a una subclase.

- Clases sin atributos, sin métodos o sin relaciones: eliminarlas.
- Relaciones no necesarios: eliminarlas.

Atributos de clase necesarios en un acceso: pasarlos a atributos de relación.

- **Modularizar**

Una vez identificadas las clases con sus atributos y relaciones, es necesario organizar el modelo en módulos o tópicos que permitan organizar de mejor manera el trabajo de análisis, para lo cual agruparemos las clases en módulos, dependiendo el dominio del problema.

- **Añadir y simplificar métodos**

Veamos a continuación un ejemplo para el modelamiento de una máquina de café Fig.2-1. Un diagrama de estructura estática inicial podría ser:

➢ **Especificación dinámica**

Diagramas De Secuencia

Indican la secuencia explícita de estímulos, siendo muy útiles para describir escenarios de alta complejidad y especificaciones de tiempo real[3]. El diagrama presenta dos dimensiones:

- Dimensión vertical, que representa el tiempo
- Dimensión horizontal, representando los diferentes objetos.

Veamos un ejemplo de diagrama de secuencia, para una cafetera. Fig2-2.

Diagramas De Colaboración

Indican las relaciones entre las instancias, siendo útiles para comprender los efectos sobre una determinada instancia[3].

Este diagrama permite identificar colaboraciones, que contienen los diferentes roles desempeñados por los objetos y las relaciones necesarias en un contexto particular.

Además, este tipo de diagrama es utilizado para diseñar los procedimientos[18], vemos un ejemplo de este diagrama en la Fig. 2-3.

Diagramas De Gráficas De Estado
Representa el comportamiento dinámico de entidades capaces de emitir una respuesta al recibir eventos de sucesos. Se utiliza para describir el comportamiento de las clases, pero puede utilizarse también para indicar el comportamiento de casos de uso, actores, subsistemas, operaciones o métodos[18].
Veamos a continuación los estados posibles de la clase MaquinaCafe Fig. 2-4.

Diagramas De Actividad
Este diagrama es un tipo específico de diagrama de estado, donde todos los estados son acciones o subactividades, donde las transiciones son originadas al terminar las acciones o subactividades en los estados fuente[5] (ver Fig. 2-5).

- **Especificación funcional**

Durante el análisis la funcionalidad del sistema se identifica mediante los diagramas de casos de uso.
Estos diagramas se utilizan para mostrar la relación entre actores y casos de uso dentro de un sistema. Los componentes de este diagrama son:

- Conjunto de casos de uso
- Actores
- Relaciones de caso de uso
- Relaciones entre actores

Veamos el diagrama de caso se uso para la máquina de café Fig. 2.6.

La funcionalidad del sistema se ve identificada al definir las operaciones de cada uno de los objetos, las cuales permitirán accesar o navegar de un objeto a otro.
Con la finalidad de describir claramente la funcionalidad del sistema se utilizará el siguiente esquema (tabla 2-a), por cada operación identificada durante el análisis.

Operación:	Nombre
Descripción:	Texto
Lee:	Items
Modifica:	Items
Envía:	Agente y Eventos
Asume:	Precondición
Resulta	Postcondición

Tabla 2-a

3.- Diseño

En el diseño se utilizarán los modelos obtenidos durante el análisis, los cuales serán refinados y adaptados al ambiente de implementación; además se tomarán decisiones de cómo resolver el problema
En el diseño se realiza la clasificación de las personas que usarán el sistema, basándose en algunos criterios como: nivel de destreza, nivel de organización, miembros en diferentes grupos[8].
El propósito es realizar un diseño que soporte los conceptos de abstracción, encapsulamiento, independecia funcional y modularidad. Para lo cual se plantea el Diseño de la arquitectura física, y Diseño de la arquitectura lógica[19]

- **Diseño de la arquitectura física del sistema**

Diseño Del Entorno De Implementación
Las tareas a realizar son:
- Identificar las restricciones técnicas en el que el sistema será incluido
- Incluir lenguajes de programación, sistemas de administración de datos, ambientes de red, sistemas operativos, administrador de transacciones a utilizar.
- Agregar clases que manejen cambios de entorno, para que estos cambios sean solo locales y no tengan efectos sobre el resto de los objetos.
- Considerar requerimientos para la ejecución o limitación de memoria.
- Estimación de los requerimientos de hardware y software para su implementación

Diseño de la arquitectura lógica del sistema
Se realiza un refinamiento del modelo de análisis, las tareas a realizar son:
- Verificar y si es necesario aumentar clases para mejorar la eficiencia
- Diseñar objetos
- Diseñar interfaces de usuario
- Identificar tareas para el sistema
- Diseño de objeto
- Escoger los algoritmos para implementar funciones
- Escoger las estructuras de datos
- Definir clases internas y operaciones
- Diseñar operaciones
- Reorganizar las clases y operaciones
- Optimizar rutas de acceso
- Identificar tareas del sistema
- Diseñar Interfases de usuario

4.- Implementación

La implementación es un proceso de traducción[13], ya que se traduce el diseño detallado a un lenguaje de programación que, por último es transformado en instrucciones ejecutables por la máquina[10].
Para la implementación de software orientado a objetos, podemos emplear herramientas orientadas a objetos y no orientadas a objetos.
El poder implementar software orientado a objetos con herramientas no orientadas a objetos es una de las ventajas que tiene esta tecnología, cosa que no sucede con la tecnología estructurada. Dentro de las herramientas no orientadas a objetos tenemos lenguajes de programación estructurada, y bases de datos relacionales, donde por mencionar las clases son implementadas como tablas y los métodos como store procedures[1]

Dentro de las herramientas orientadas a objetos tenemos: Herramientas CASE como Object Team, Rational Rose, Lenguajes de programación como C++, Eiffel, Smaltalk, Java, Bases de datos como Versant ODBMS, Object Store, todas ellas orientadas a objetos.

Dentro de las orientadas a objetos se deben definir estándares de acuerdo a la herramienta que se vaya a utilizar, por ejemplo en PowerBuilder el estándar para ventana es W_ventana, el estándar para menú: es m_menú, el estándar para un botón de comando es: cb_botón

Diagrama de Implementación
Muestra la estructura del código (diagrama de componentes) y la estructura del sistema en ejecución (diagrama de ejecución)

Diagrama de componentes
Muestra las dependencias lógicas entre componentes de software, sean estos componentes fuentes, binarios o ejecutables. Los componentes de software tienen tipo, que indica si son útiles en tiempo de compilación, enlace o ejecución. Se representan como un grafo de componentes unidos por medio de relaciones de dependencia(generalmente de compilación). Puede mostrarse también contenencia de entre componentes software e interfaces soportadas.
Veamos un ejemplo Fig. 2-7 en el que se tiene tres componentes, GUI dependiendo de la interfaz *undate* provista por *Planner*, *Planner* dependiendo de la interfaz *reservations provista por Sheduler.*

Diagrama d ejecución
Muestra la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software, procesos y objetos que se ejecutan en ellos. Instancias de los componentes software representan manifestaciones en tiempo de ejecución del código. Este diagrama conecta nodos por asociaciones de comunicación. Un nodo puede contener instancias de componente de software, objetos, procesos. Las de componentes pueden estar unidos por relaciones de dependencia, posiblemente a interfaces.
A continuación se presenta un ejemplo de diagrama de ejecución (Fig. 2-8) en el cual tenemos dos nodos, AdminServer y Joe´sMachine. AdminServer contiene la instancia del componente Scheduler y un objeto activo (proceso) denominado *meetingsDB*. En Joe´sMachine se encuentra la instancia del componente software Planner, que depende de la interfaz *reservations,* definida por Scheduler

5.- Pruebas

Para probar adecuadamente los sistemas orientados a objetos, deben hacerse tres cosas:
1.- La definición de las pruebas deben ampliarse para incluir técnicas de detección de errores aplicados a los modelos de análisis y diseño orientados a objetos
2.- La estrategia para las prueba de unidad e integración deben cambiar significativamente.
3.- El diseño de casos de prueba deben tener en cuenta las características propias del software orientado a objetos.
En cada etapa los modelos pueden probarse en un intento por descubrir errores antes de que se propaguen en la próxima iteración. Si un problema no se detecta durante el análisis y se sigue propagando, pudiera ocurrir los siguientes problemas durante el diseño:
1.- Asignación impropia de clases a subsistemas y/o tareas durante el diseño del sistema
2.- Esfuerzo de trabajo no necesario para crear el diseño procedimental de las operaciones relacionadas con el atributo innecesario.
3.- El modelo de intercambio de mensajes sería incorrecto
Todos los modelos orientados a objetos deben probar su corrección (exactitud), compleción y consistencia dentro del contexto de la sintaxis del modelo, semántica y pragmática.

Se propone el siguiente plan de pruebas:

1.- Ambito de la prueba
2.- Plan de la prueba
    2.1.- Fases de la prueba y construcción
    2.2.- Planificación
    2.3.- Software adicional
    2.4.- Entorno y recursos
3.- Procedimiento de prueba (descripción de la prueba para la construcción)
    3.1.- Orden de integración
        3.1.1.- Propósito
        3.1.2.- Módulos que hay que probar
    3.2.- Realización de la prueba
        3.2.1.- Descripción de la prueba
        3.2.2.- Descripción del software adicional
        3.2.3.- Resultados esperados
    3.3.- Entorno de la prueba
        3.3.1.- Herramientas o técnicas especiales
        3.3.2.- Descripción del software adicional
    3.4.- Datos de los casos de prueba
3.5.- Resultados esperados para la construcción n
4.- Resultados obtenidos de la prueba
5.- Referencias
6.- Apéndices

5.- Mantenimiento

El mantenimiento es la última fase del proceso de ingeniería del software y se lleva la mayor parte del presupuesto destinado al software de computadora.
Es por esto que se debe desarrollar mecanismos para evaluar, controlar y realizar modificaciones. Se puede definir el mantenimiento describiendo cuatro activi dades que son:

- Mantenimiento correctivo: modifica el software para corregir los defectos.
- Mantenimiento adaptivo: produce modificaciones en el software para acomodarlo a los cambios de su entorno.
- Mejoras o mantenimiento de perfeccionamiento: Lleva al software más allá de sus requisitos funcionales originales.
- Mantenimiento preventivo o reingeniería: Hace cambios en los programas a fin de que se pueda corregir, adaptar y mejorar fácilmente.

Conclusiones

- Las técnicas orientadas a objetos permiten tener una mejor comprensión de los requerimientos del sistema, logrando proponer eficientes soluciones a los problemas planteados en el análisis, un diseño claro y logrando que el mantenimiento no sea complicado.
- Con la utilización de técnicas orientadas a objetos se obtiene mayor flexibilidad para realizar mantenimiento y modificaciones del software.
- El análisis orientado a objetos se produce a diferentes niveles de abstracción, a nivel empresarial o de negocios; las técnicas asociadas con el análisis orientado a objetos pueden acoplarse con un enfoque de ingeniería de la información.
- El diseño orientado a objetos representa un enfoque único para el ingeniero de software
- Con diseño orientado a objetos se mejora la calidad del software ya que ofrece un medio para romper la partición entre datos y procesos.
- El análisis y diseño orientado a objetos es más complejo que el estructurado.
- Debido a que las fases del ciclo de vida del software orientado a objetos nos son rígidas, se puede realizar un diseño iterativo
- El tiempo empleado en el análisis y diseño del sistema se ve recompensado en la fase de implementación, la cual es más corta que en otras técnicas.
- Con una buena planificación se pueden controlar los procesos de ingeniería de software y tener éxito en el desarrollo del software
- Al desarrollar software se pretende que los productos alcancen un nivel de calidad apropiado para satisfacer las necesidades de los usuarios o para ser competitivos en el entorno donde tienen que convivir; la utilización de estándares puede ayudar a conseguir dichos niveles de calidad.
- No todas las herramientas poseen todas las características de orientación a objetos, y generalmente estas son escogidas por su popularidad en el mercado, soporte local y aspectos económicos.
- Se recomienda que se utilice la tecnología orientada a objetos ya que permite crear software robusto, confiable, y eficiente.
- Se recomiendo que se estudien las herramientas orientadas a objetos que aparecen en el mercado y se divulguen a todo nivel

Referencias

[1] BERTINO E; MARTINO L. ***Sistema de Bases de Datos orientados a Objetos*****,** Addison Wesley, 1993.

[2] BOOCH, G. *Object Oriented Design*, Benjamin/Cummings Publishing Company, 1991

[3] BOOCH,G; RUMBAUGH,J; JACOBSON,I, *The Unified Modeling Language User Guide*, New York, Addison Wesley, 1999.

[4] BOOCH,G. ***Análisis y Diseño Orientados a Objetos con aplicaciones*****,** Addison Wesley, 1996

BOOCH,G; RUMBAUGH,J; JACOBSON,I, *The Unified Modeling Language Reference Manual*, New York, Addison Wesley, 1999

MARTIN FOWLER Y KENDALL SCOTT. ***UML Gota a Gota*****,** Pearson. 1999.

COLEMAN,D.***Object-Oriented Development***, Prentice Hall, 1994

RUMBAUGH,J. ***Object-Oriented Modeling and Design***, Prentice Hall, 1994

KENNETH,R.***Developping Object Oriented Software***, Prentice Hall, 1997

PRESSMAN,R. ***Ingeniería de Software*****,** Mc. Graw Hill, 19997

CORDOVA R.***Caracterización de la Actividad de Análisis en el Desarrollo de software Orientado a Objetos*****,** Sao Paolo, 1995

JEREZ, Y. ***Estudio comparativo de los métodos de Análisis de Sistemas que utilizan la Técnica de Orientación a Objetos***, Quito, 1997

http://www.csc.calpoly.edu/~dbutler/tutorials/ winter96/rose/node7.html

http://www.inf.ufsc.br/poo/ine5383/oose.html

http://www.cool.sterling.com/solutions/hitachi _ telecom.htm

http://www.omg.org/cgi-bin/membersearch.pl

http://meltingpot.fortunecity.com/seymour/83 3/informe.htm

http://www.lsi.us.es/~corba/estudios/corba- idl.ppt

http://agamenon.uniandes.edu.co/~revista/articulos/corba/corba.htm

http://uxmcc1.iimas.unam.mx/foro/javaDC99-1/resumen3.html

Biografía

Tania A. Acosta H, nacida en la ciudad de Guayaquil el 3 de junio de 1972, sus estudios primarios los realizó en la escuela particular "Centro Escolar Ecuador", los secundarios en el colegio "Nacional de Señoritas Ambato", y la ec a realizó en la "Escuela Politécnica Nacional", en la Facultad de Ingeniería de mente se desempeña como ayudante de cátedra a tiempo completo en el In s Básicas, ocupando también el cargo de Jefe del laboratorio de co instituto.

El Ing. Raúl Córdova, Ingeniero en Electrónica y Control Escuela Politécnica Nacional 1989;
Master en Ingeniería Eléctrica especialidad computación, Universidad de Sao Paulo , abril 1995. Profesor de Escuela Politécnica Nacional en la Facultad de Ingeniería de Sistemas desde 1986.
Profesor en la Maestría Informática en el programa conjunto E.P.N. y la Universidad Andina Simón Bolívar.
Profesor de la E.S.P.E. en la Facultad de Ingeniería de Sistemas Informáticos desde 1995
Profesor de la Maestría en Gerencia de Sistemas, E.S.P.E. 1995
Coordinador del área de ingeniería de software en la E.S.P.E. desde 1997
Coordinador del área de ingeniería de software en la
E.P.N. desde 1999