Programación Orientada a Objetos

Profr. Pedro Pablo Mayorga

Actividad 2

Unidad 1

Ciclo de vida del software y Diseño Orientado a Objetos

Ciclo de Vida del Software

Un modelo de ciclo de vida define el estado de las fases a través de las cuales se mueve un proyecto de desarrollo de software.

El primer ciclo de vida del software, "Cascada", fue definido por Winston Royce a fines del 70. Desde entonces muchos equipos de desarrollo han seguido este modelo. Sin embargo, ya desde 10 a 15 años atrás, el modelo cascada ha sido sujeto a numerosas críticas, debido a que es restrictivo y rígido, lo cual dificulta el desarrollo de proyectos de software moderno. En su lugar, muchos modelos nuevos de ciclo de vida han sido propuestos, incluyendo modelos que pretenden desarrollar software más rápidamente, o más incrementalmente o de una forma más evolutiva, o precediendo el desarrollo a escala total con algún conjunto de prototipos rápidos.

Definición de un Modelo de Ciclo de Vida.

Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software, intenta determinar el orden de las etapas involucradas y los criterios de transición asociadas entre estas etapas.

Un modelo de ciclo de vida del software:

- Describe las fases principales de desarrollo de software.
- Define las fases primarias esperadas de ser ejecutadas durante esas fases.
- Ayuda a administrar el progreso del desarrollo, y
- Provee un espacio de trabajo para la definición de un detallado proceso de desarrollo de software.

Así, los modelos por una parte suministran una guía para los ingenieros de software con el fin de ordenar las diversas actividades técnicas en el proyecto, por otra parte suministran un marco para la administración del desarrollo y el mantenimiento, en el sentido en que permiten estimar recursos, definir puntos de control intermedios, monitorear el avance, etc.

Dos de los modelos de ciclo de vida del software más usados son el modelo de cascada y el de espiral.

Modelo Cascada.

Este es el más básico de todos los modelos, y sirve como bloque de construcción para los demás modelos de ciclo de vida. La visión del modelo cascada del desarrollo de software es muy simple; dice que el desarrollo de software puede ser a través de una secuencia simple de fases. Cada fase tiene un conjunto de metas bien definidas, y las actividades dentro de una fase contribuyen a la satisfacción de metas de esa fase o quizás a una sub secuencia de metas de la fase. Las flechas muestran el flujo de información entre las fases. La flecha de avance muestra el flujo normal. Las flechas hacia atrás representan la retroalimentación.

El modelo de ciclo de vida cascada, captura algunos principios básicos:

- **Requerimientos**: Saber que información necesita el software que se va a desarrollar y de qué manera interactúan. Aquí es muy importante visualizar las distintas clases que intervienen en estos procesos planteadas en forma general.
- **Diseño**: es diseñar los modelos de clases y la relación que estas tienen con otras clases del sistema, así como la especificación de los resultados que se desean.
    - Este punto se hace con la información de requerimientos del punto anterior.
    - En este paso se tienen que definir formalmente los diagramas de clases (relación de dependencia entre las clases en forma de herencia) y especificar las propiedades y métodos de cada una de las clases.
    - Es muy importante definir las relaciones que necesita tener una clases con otras (no es el diagrama de herencia). Este tipo de diagramas se llama Diagrama de Flujo de Datos o DFD. Un DFD permite saber de qué forma los métodos y propiedades de una clase necesitan comunicarse con los métodos y atributos de otras clases.
    - El diseño debe de ser independiente del lenguaje de programación que se vaya a usar en la etapa de codificación.
    - Si el diseño no es capaz de representar el comportamiento del sistema, se regresa al paso anterior para corregir posibles errores al plantear los requerimientos.
- **Codificación:** Es el proceso en el que el los diagramas de clases y los DFD se programan en algún lenguaje de programación para producir un archivo ejecutable que realice las tareas requeridas.
    - En este punto se tiene que adaptar el Diagrama de clases y el DFD al lenguaje elegido en esta etapa
    - Se realiza una prueba del sistema, si no se obtienen los pasos adecuados se retorna al paso anterior para corregir defectos errores de diseño.
- **Integración del sistema**: En esta parte se integra el sistema para que el usuario final pueda darle uso. Se puede volver al paso anterior para corregir errores de programación.
- **Operación y Mantenimiento**: En este punto el sistema ya funciona adecuadamente y se le da el mantenimiento adecuado según los requerimientos para que siga operando adecuadamente. En esta etapa se realizan las siguientes adaptaciones:
    - Cuando han surgido actualizaciones del entorno del equipo donde opera el sistema. Por ejemplo instalación de nuevos dispositivos tales como impresoras, escáner, lectores de huella digital entre otros.

- Cuando el sistema operativo necesita actualizarse y se debe de adaptar el programa para que siga siendo compatible con el SO.
- Cuando surge un nuevo requisito de funcionamiento del software, por ejemplo identificación de usuarios con huella digital en lugar de {usuario, contraseña} o cuando se requiere agregar un nuevo reporte no previsto en los requisitos, por ejemplo agregar una gráfica de barras a los reporte de ventas.

Un diagrama que muestra estas fases se muestra en la figura 1.
Este diagrama muestra de manera clara el proceso y nos da una clara interpretación gráfica del nombre "Cascada".

Para realizar un avance en la serie de procesos es necesario haber concluido el proceso anterior de manera exitosa. Si se detecta un fallo en alguna fase se retorna a la fase previa para corregir o agregar los procesos que puedan dar solución a dicha problemática.

Este modelo es poco usado en la actualidad porque impide realizar procesos en paralelo, sin embargo para proyectos pequeños que involucren no más de 5 personas, como los que se realizan en empresas pequeñas y medianas, es posible llevar a cabo este modelo en cascada sin problemas.

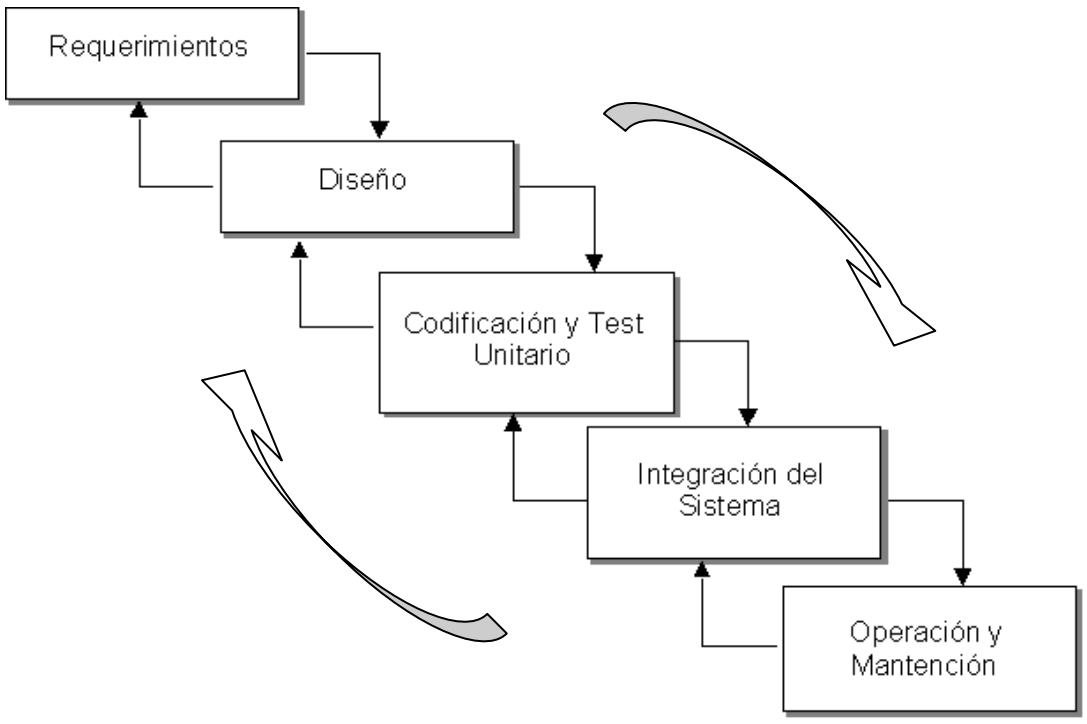


Fig. 1. Diagrama del modelo de Ciclo de Vida en Cascada. Para realizar un proceso es necesario haber concluido el proceso anterior, excepto por la fase de requerimientos. Si se detecta un fallo en alguna fase se retorna a la fase previa para corregir o agregar los procesos que puedan dar solución a la problemática.

Modelo en Espiral.
El modelo espiral de los procesos software es un modelo del ciclo de meta- vida. En este modelo, el esfuerzo de desarrollo es iterativo. Tan pronto como uno completa un esfuerzo de desarrollo, otro comienza. Además, en cada desarrollo ejecutado, puedes seguir estos cuatros pasos:

1. Determinar qué quieres lograr.
2. Determinar las rutas alternativas que puedes tomar para lograr estas metas. Por cada una, analizar los riesgos y resultados finales, y seleccionar la mejor.
3. Seguir la alternativa seleccionada en el paso 2.
4. Establecer qué tienes terminado.

La dimensión radial en la figura 2, refleja costos acumulativos incurridos en el proyecto. Observemos un escenario particular. Digamos que en este proyecto, nosotros viajaremos a resolver un conjunto particular de problemas del cliente. Durante el primer viaje alrededor de la espiral, analizamos la situación y determinamos que los mayores riesgos son la interfaz del usuario. Después de un cuidadoso análisis de las formas alternativas de direccionar esto (por ejemplo, construir un sistema y esperar lo mejor, escribir una especificación de requerimientos y esperar que el cliente lo entienda, y construir un prototipo), determinamos que el mejor curso de acción es construir un prototipo.

Lo realizamos. Luego proveemos el prototipo al cliente quien nos provee con retroalimentación útil. Ahora, comenzamos el segundo viaje alrededor de la espiral. Este tiempo decidimos que el mayor riesgo es ese miedo a que muchos nuevos requerimientos comiencen a aparecer sólo después de que el sistema sea desplegado. Analicemos las rutas alternativas, y decidimos que la mejor aproximación es construir un incremento del sistema que satisfaga sólo los requerimientos mejor entendidos. Hagámoslo ya. Después del despliegue, el cliente nos provee de retroalimentación que dirá si estamos correctos con esos requerimientos, pero 50 nuevos requerimientos ahora se originarán en las cabezas de los clientes. Y el tercer viaje alrededor de la espiral comienza.

El modelo espiral captura algunos principios básicos:

- Decidir qué problema se quiere resolver antes de empezar a resolverlo.
- Examinar tus múltiples alternativas de acción y elegir una de las más convenientes.
- Evaluar qué tienes hecho y qué tienes que haber aprendido después de hacer algo.
- El sistema que estás construyendo NO será el sistema que el cliente necesita, es decir lo será al final, pero durante el proceso será susceptible de mejoras.
- Conocer (comprender) los niveles de riesgo, que tendrás que tolerar. El modelo espiral no es una alternativa del modelo cascada, ellos son completamente compatible.

La ventaja del modelo en espiral es que el software es que es modular, se comienza con los objetivos (Que necesidad debe cubrir el producto) se postulan las alternativas de

conseguir los objetivos de forma exitosa y finalmente Desarrollar y Verificar un prototipo.

La espiral tiene una forma de caracola y se dice que mantiene dos dimensiones, la radial y la angular. La Angular indica el avance del proyecto software dentro de un ciclo. La dimensión Radial, indica el aumento del coste del proyecto, ya que con cada nueva iteración se pasa más tiempo desarrollando.

Este sistema es muy utilizado en proyectos grandes y complejos como puede ser, por ejemplo, la creación de un Sistema Operativo. Al ser un modelo de Ciclo de Vida orientado a la gestión de riesgo se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente los riesgos.

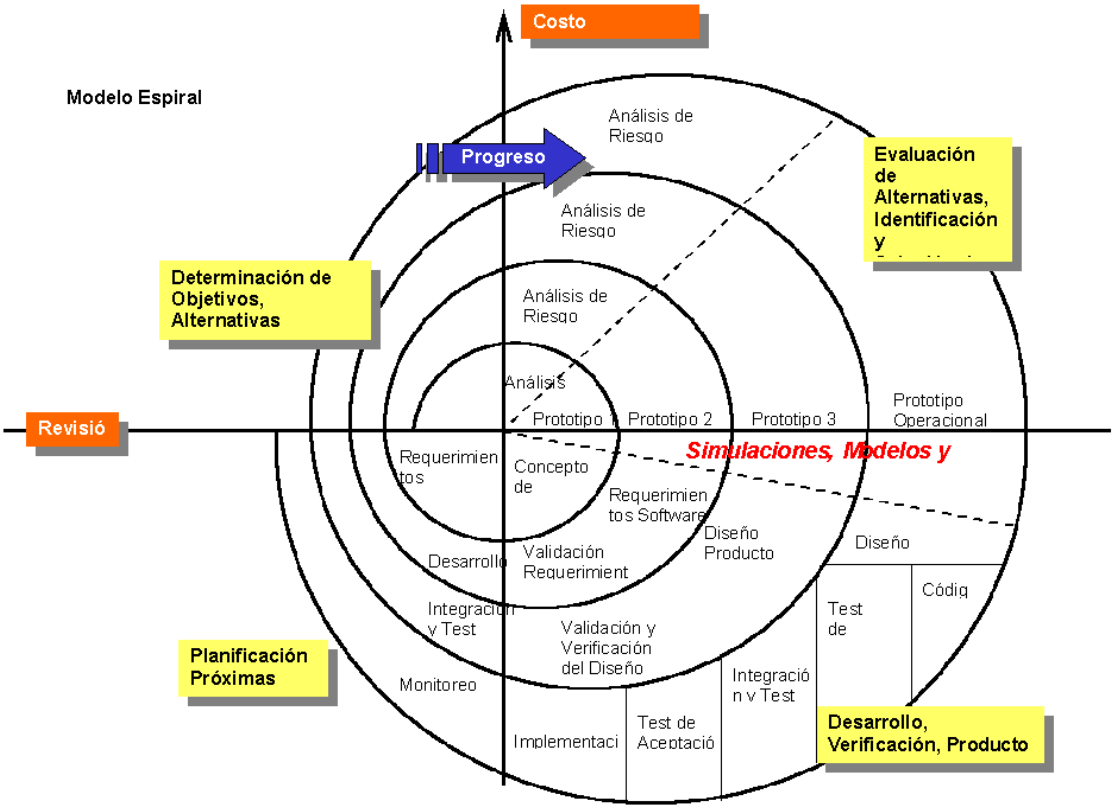


Fig. 2. Modelo de ciclo de vida en Espiral.

Análisis y diseño Orientado a Objetos

El Análisis y diseño orientado a objetos es un enfoque de la ingeniería de software que modela un sistema como un grupo de objetos que interactúan entre sí.

En éste método de análisis y diseño se crea un conjunto de modelos utilizando una notación acordada como, por ejemplo, el lenguaje unificado de modelado (UML).
Esto requiere de la aplicación de técnicas de modelado de objetos para analizar los requerimientos para un contexto
No está restringido al diseño de programas de computadora, sino que cubre sistemas enteros de distinto tipo. Las metodologías de análisis y diseño más modernas son casos de uso guiados a través de requerimientos, diseño, implementación, pruebas, y despliegue. El lenguaje unificado de modelado se ha vuelto el lenguaje de modelado estándar usado en análisis y diseño orientado a objetos.

Diseño Orientado a Objetos

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

Al diseñar una clase se debe pensar en cómo se puede identificar un objeto real, como una persona, un transporte, un documento o un paquete. Estos ejemplos de clases de objetos reales, es sobre lo que un sistema se diseña.

Durante el proceso del diseño de las clases se toman las propiedades que identifican como único al objeto y otras propiedades adicionales como datos que corresponden al objeto. Con los siguientes ejemplos se definen tres objetos que se incluyen en un diagrama de clases:

Ejemplo 1. Una persona tiene número de documento de identificación, nombres, apellidos, fecha de nacimiento, género, dirección postal, posiblemente también tenga número de teléfono de casa, del móvil, FAX y correo electrónico.

Ejemplo 2. Un sistema informático puede permitir administrar la cuenta bancaria de una persona, por lo que tendrá un número de cuenta, número de identificación del propietario de la cuenta, saldo actual, moneda en la que se maneja la cuenta.

Ejemplo 3. Otro objeto pueden ser "Manejo de Cuenta", dónde las operaciones bancarias de una cuenta (como en el ejemplo 2) se manejarán realizando diferentes operaciones que en el diagrama de clases sólo se representan como operaciones, que pueden ser:

a. Abrir
b. Cerrar
c. Depósito
d. Retiro
e. Intereses

Diagrama de Clases

El diagrama de clases es una forma gráfica de representar a las clases junto con sus propiedades y métodos así como las relaciones de herencia entre ellas.
Recordamos que la herencia es una técnica de la POO que esquematiza una jerarquía en las propiedades y métodos de tal forma que la clase que hereda (padre) a otra clase, le hereda sus propiedades y métodos.

En este punto cabe resaltar que existen tres tipos de ocultamiento dentro de una clase que permite ocultar/mostrar dichas propiedades o métodos.

- **Público**: Esta característica no oculta nada y se usa para que la propiedad y/o método
    a. Se Hereden a las sub clases como tipo **Público**.
    b. Se puedan acceder por otras clases.
- **Protegido**: Esta característica oculta las propiedades y métodos a clases ajenas a ella, pero permite heredarlos a las sub clases.
    a. Se Heredan a las sub clases como tipo **Protegido**.
    b. Se impide el acceso de estas propiedades o métodos por otras clases.
- **Privado**: Esta característica oculta las propiedades y métodos tanto a clases ajenas y a sub clases.
    a. No se heredan las propiedades ni métodos.
    b. Se impide el acceso de estas propiedades o métodos por otras clases.

Se realizan los diagramas de clases para poder realizar la abstracción de un dominio Formalizar el análisis de conceptos así como poder definir una solución de diseño y así poder construir componentes de software.

En la figura 3, se muestra un ejemplo de una codificación de un objeto del mundo real a una clase con el modelo orientado a objetos, observe como se especifican las propiedades con el tipo de dato adecuado a su funcionamiento y los métodos con los parámetros necesarios para poder funcionar e indicando si poseen un valor de retorno que indique el estado de ejecución del método.

En la figura 4, se muestran los distinto tipos de relaciones entre las clases, mismas que se describen a continuación.

Asociación básica entre clases: Define una determinada vinculación entre dos tipos. El conector puede indicar el rol de la asociación fuente y destino, la cardinalidad y el tipo de navegabilidad (bidireccional o unidireccional).

Generalización (Herencia): El elemento destino es una especialización del elemento fuente. Dentro de una escala de abstracción variable, el elemento fuente es el más abstracto. Es común llamar a la case fuente como clase **padre** mintras que a la clase destino se le denomina clase **hija**. Esta relación es una técnica de la POO que se denomina herencia.

Agregación: El elemento destino "forma parte" del elemento fuente. Dicha relación puede romperse sin restricciones. Este tipo de relaciones sucede cuando una clase

pertenece a otra clase o está dentro de otra clase. Por ejemplo una clase "Polígono" tiene como propiedad a la clase "Punto", debido a que la clase polígono necesita de 3 o más puntos para poder existir como objeto.

Dependencia (Colaboración): Se usa cuando una clase depende de otra para poder funcionar adecuadamente.

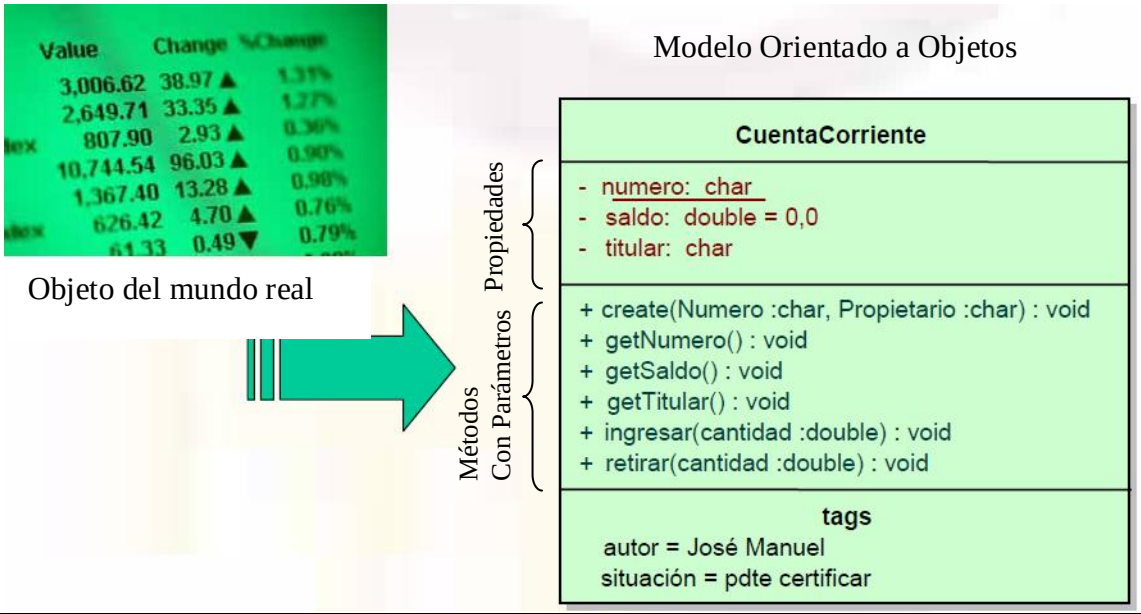
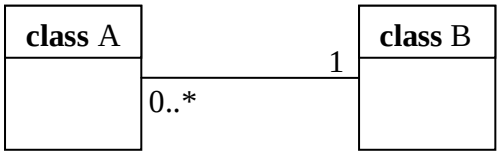
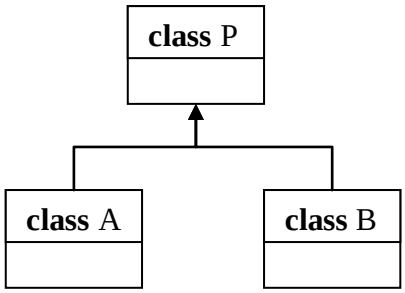


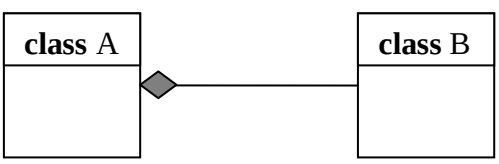
Fig.3. Diagrama de clases de una Cuenta Corriente. El modelo se observa de un objeto del mundo real y se lleva a la abstracción del modelo POO.



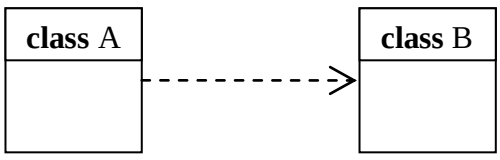
(a) Asociación. Esta relación indica que la clase A se asocia con 0 o más clases B, mientras que en la otra parte indica que la clase B se asocia solamente con 1 clase A.



(b) Herencia. Esta relación indica que la clase P es la clase padre de A y B.



(c) Agregación. La clase B forma parte de A. Nota el rombo indica el inicio de la línea.



(d) Dependencia. La clase A depende de la clase B.

Fig. 4. Distintas formas de relaciones entre clases. (a) Asociación (b) Herencia y (c) Agregación y (d) Dependencia.

En la figura 5 y 6 se muestran diagramas de clases de herencia y de colaboración de una parte de las librerías VTK localizado en www.vtk.org que se usan para visualizar objetos tridimensionales usando open gl.

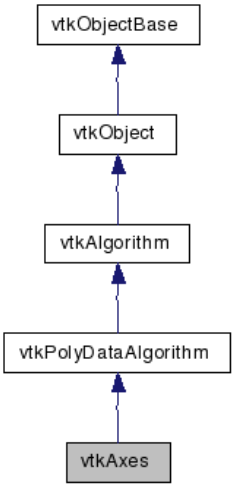


Fig. 5. Diagrama de Herencia de la clase **vtkAxes**

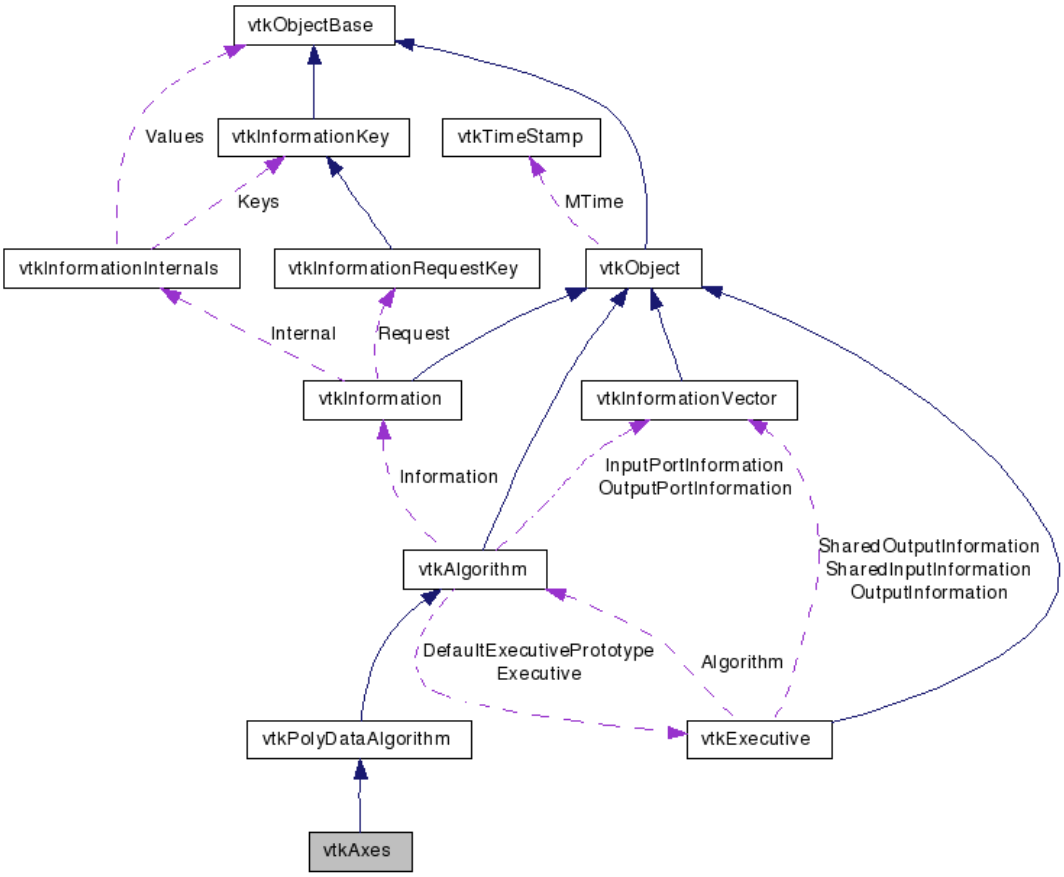


Fig 6. Diagrama de Colaboración de la clase **vtkAxes.**