¿Es verdad que la programación orientada a objetos solo complica las cosas innecesariamente y hace que el software sea más lento?

Jesús Gómez, trabajé en Universidad Central de Venezuela
Actualizado hace 4 Oct. 2018

Teóricamente añade una capa de abstracción lo que implica más complejidad y más costos operativos. Pero la ganancia es lo contrario de complicar: simplificar la solución al brindar herramientas cognitivas a los programadores.

Pero … sí que puede llegar a ser un estorbo para los programadores, causando diseños complicados, si se insiste en utilizar ese paradigma en dominios donde no es adecuado.

Los lenguajes son sólo herramientas, y no debes ver todo problema como un clavo.

Nada mejor que la POO para el desarrollo de aplicaciones interactivas con metáforas de la vida real (escritorio, ventana, icono, etc.) ~y/o en tiempo real (Entornos de Escritorio, video juegos, etc.) ~.

En otros dominios, la POO empieza a perder su pertinencia. Un buen ejemplo es el procesamiento y análisis de datos, en el que la programación funcional es más adecuada e insistir en un diseño bajo el paradigma POO entorpecería el proceso de desarrollo.

_____

Actualización:

Inicialmente mencioné que la POO era el mejor estilo de programación para, entre otras cosas, programar video juegos.

Sin embargo, he aprendido que, a pesar de ser el paradigma preferido en la industria de los videojuegos, la POO es un problema gratis que los diseñadores tienen que resolver luego con técnicas de agrupación de datos, sobre todo en el caso de videojuegos en tiempo real que requieren un alto desempeño de la computadora.

Si el diseño del juego se centra en los datos y sus transformaciones, estos problemas de desempeño que introduce la POO se evitarían. Y por supuesto, en este tipo de diseños centrados en los datos, es más intuitivo utilizar la programación funcional.

Comparto un artículo (en inglés) que explica el concepto del Diseño Orientado a los Datos[1].

Alexandro Ramos Rodríguez
Actualizado 23/7/2018 · El autor tiene **143** respuestas y **33,8k** vistas de respuestas

Me parece que antes que preguntar esto, habría que entender qué es la programación orientada a objetos.

Posiblemente, has aprendido a usar ya sentencias de control en algún lenguaje de programación: If, For, Switch /Case, While, Do, While, etc… Estas se denominan estructuras de control.

Puede que también hayas aprendido estructuras de datos: Colas, Pilas, arreglos, diccionarios, tablas, árboles, etc..

Lo antes mencionado, se denomina "programación estructurada" por ser constituida por estructuras de datos, y estructuras de control. Esta programación estuvo en boga desde los 50's.

Conforme el tiempo pasó, hubo mayores capacidad de memoria y procesamiento, y los programadores se hartaron de hacer cosas similares, en el sentido de que, si ya has programado una rutina para algo, y tienes que volver a implementarlo, no andas copiando el código, sino empezaron a haber librerías con las rutinas (como las cabeceras de C) Además los equipos de trabajo crecieron, por lo que había necesidad de establecer reglas sobre cómo se debería programar, para que lo hecho por un miembro del equipo no interfiera con lo hecho por otros.

Además, las estructuras de datos llegaron a ser realmente complejas para aplicaciones de diseño (CAD)

Entonces vino la idea de juntar las estructuras de datos, junto con las estructuras de control que aplicaban. Esto fue para que el código, empezara a tener lógica por sí mismo. Por ejemplo, si dibujabas un círculo, bien podrías tener las operaciones que aplicaban para los círculos aparte de las que aplicaban a otros objetos (como rectángulos). La idea de llevar la separación de intereses (separation of concerns Separación de intereses - Wikipedia, la enciclopedia libre) y llevarlo a la práctica con código reutilizable conlleva al concepto de "objeto" naturalmente. El código en si mismo ya no es un conjunto de instrucciones, sino empieza a tener reglas sobre cómo pueden interactuar los objetos, o qué pueden hacer. Este también es punto de inicio de los lenguajes fuertemente tipados (Tipado fuerte - Wikipedia, la enciclopedia libre).

Desde una perspectiva de bajo nivel, lo que hace un objeto, es copiarse (instanciarse) desde la clase, a un espacio de memoria de trabajo.

Entonces, habiendo establecido el por qué surgió, se puede decir que la POO consume más recursos que una programación estructurada. No necesariamente más lenta. La diferencia entre programación estructurada y POO, podría ser análoga a la diferencia que hay entre una célula procarionte y una célula eucarionte. La primera es más simple, lo que le da agilidad (viven en más ambientes) pero son incapaces de crear seres complejos, o aprovechar mejor el alimento (ciclo de krebs completo).

Por cierto, buscando quién había inventado la POO, me encontré con un muy interesante video en el cual Alan Kay, inventor de la POO, está mostrando un CAD realizado por él, en los 70's: Alan Kay: Doing with Images Makes Symbols Pt 1 : Free Download & Streaming : Internet Archive

*encontrado en la respuesta de Quora: Who invented Object Oriented Programming(OOP) and what was the motivation and inspiration?

Como colofón la programación orientada a objetos no ha desplazado a la programación estructurada o procedural, simplemente ha sido una opción, una solución a cómo crear software.

Marco Alvarado, Más de 25 años de experiencia en muchos lenguajes.
Contestado 20/5/2018

La respuesta más simple es NO. No complica nada ni hace el software más lento.

Lo más importante es modelar una solución a un problema y la orientación a objetos es una forma muy efectiva de crear modelos elegantes, correctos y funcionales.

Si lo que quiero es programar de forma procedimental con facilidades de orientación a objetos, entonces pues si, será más lento … pero NO será programación orientada a objetos, será una simple traducción de primitivas de programación, lo cual de todas formas es una forma incorrecta de utilizar el paradigma. Esto ocurre muchas veces cuando se escribe un programa en C utilizando C++. Si quiero utilizar C++, pues es mejor diseñar primero correctamente y luego codificar correctamente con el paradigma que corresponda.

También está el error que dice que entre menos código las cosas son más rápidas. La verdad es que la velocidad viene de haber diseñado correctamente y de utilizar los algoritmos adecuados. Puedo trabajar un acceso secuencial de un archivo en ensamblador que sea más lento que uno indizado sobre una biblioteca orientada a objetos que abstraiga cierto concepto.

Al final, la complejidad innecesaria y la lentitud vienen de un mal trabajo previo a la codificación, sin importar el paradigma ni la herramienta utilizada para desarrollar el software.

Alfredo Pinto, Front End Developer (2017 - presente)
Contestado 23/5/2018 · El autor tiene **115** respuestas y **94,9k** vistas de respuestas

Siempre en este tipo de preguntas hay que investigar un poco mas para poder comprender ya que todas las cosas no son blanco y negro. ¿Por que nació la POO? ¿como fue que a alguien se le hizo buena idea usarla?

La POO nació como una forma de modelar el clima del planeta. Como todos sabemos las computadoras solo entienden de ceros y unos. La función del programador es encontrar como decirle a la computadora que solucione el problema que tenemos y los patrones de diseño aplicados a los lenguajes de programación nos ayudan a reducir esa complejidad de procesos y estructuras de datos.

La POO modela el mundo real, y fue tan bueno en eso que sin problemas fue fácilmente aceptada a la hora de crear software para empresas (enterprise software) donde los complejos procesos de trabajo y estructuras de datos podían ser mucho mas fácil modelados y manejados.

No es el único patron que existe y hay que aprender a saber cuando si usarlo y cuando no. En este negocio no hay balas de plata, o sea, una solución para todos los problemas. Para aplicaciones

que necesitan respuesta en tiempo real no es recomendable, tampoco es la mejor opción para juegos y microcontroladores. No se usa para programar sistemas operativos ni software embebido. Su fuerte es el software para empresas o enterprise.

Frecuentemente cuando alguien no le gusta algo o lo esta usando mal, lo esta usando en donde otros patrones son mejores para resolver ese tipo de problema o no sabe usarlo. En mi experiencia casi siempre es la última (y lo digo por experiencia propia).

Para hacer software para empresas considero en mi muy particular opinión que es el mejor patron ya que de manera más fácil podemos estructurar nuestro código fuente de manera que modele el mundo real y así nosotros podamos crear y mantener mucho más fácil el software. Sin embargo, en estos últimos años an surgido nuevos patrones (como la programación funcional) que parecen prometer muy buenos resultados también.

Ignacio Nicolás Rodríguez, Propietario en Fast Microservices (2015-presente)
Actualizado 10/6/2018 · El autor tiene **177** respuestas y **56,1k** vistas de respuestas

Si leés los libros de Grady Booch (más traductores e imitadores) de 1990–2000, vas a salir convencido de que la programación no es otra cosa que modelar el universo del problema y sus interacciones, y que conviene usar OOP para que ese modelaje sea lo más directo y por tanto fácilmente verificable.

Todo parece fantástico, pero es lo que se conoce científicamente como #pajamental (también “masturbación cerebral”).

En particular, si bien la mayor parte de lo que OOP ofrece es bueno y no tiene dificultades intrínsecas, hay algo que sí y es la herencia.

La herencia (reutilización de la implementación en clases derivadas), es un elemento que**rompe el aislamiento, genera compartición de estado, impide la fluidez, dificulta el refactoring**, y finalmente actúa como una sentencia de muerte por aburrimiento.

Si escuchás los podcast actuales de Grady Booch Grady Booch On Architecture - IEEECS, vas a ver muchos de estos problemas corroborados.

Claudio Monsalves, estudié en la Inacap
Contestado 16/4/2018 · El autor tiene **676** respuestas y **243,1k** vistas de respuestas

Buena pregunta.
Lo que pasa que cada paradigma y lenguaje de programacion fue diseñado para resolver un conjunto de problemas y no todos los problemas , ademas de la evolucion de la industria y las complejidades q esto acarrea en cuanto a tiempo y costos monetarios .

Cada paradigma esta hecho para resolver algunos problemas en espesifico, por ejemplo si quiero hacer un sistema operativo se decanta mas en lenguajes como c y ensamblador usando el paradigma imperativo ya que es mas rapida para tratar con electronica , lo mismo pasa para programar micro controladores o circuitos electronicos .
La POO es una solucion para hacer los sistemas grandes mas modulares logicos y reutilizables ,

ya que hacer por ejemplo un sistema por ejemplo de punto de venta seria un disparate hacerlo con orientacion imperativa porque seria largo y costoso en el tiempo y dificil de seguir , en cambio con poo tienes los modulos (clases) bien definidos , ademas cada clase puede ser reutilizada si se te presenta un problema similar .

Ahora si lo vemos por el lado de la industria , cada dia se necesita mas codigo o mas programadores q reparen los problemas de los programas y la forma mas facil de hacer esto es con poo , si a esto le sumamos que es mas barato ya que es mas legible para humanos comprender la poo tenemos como resultado un paradigma que siempre se va a ocupar si el dominio del problema es compatible con poo

Alejandro Gamboa, Estudié Ingeniería en Sistemas Computacionales en la Universidad autonoma de chihuahua
Contestado 19/5/2018 · El autor tiene **206** respuestas y **104,7k** vistas de respuestas

Lo lento o rápido no creo que dependa del paradigma ya que al final todos los lenguajes compilados son traducidos a lenguaje ensamblador , si hablamos de Java o C# la lentitud se debe a la maquina virtual que utilizan de intermediario.

Ahora el tema interesante , la complejidad , todos los paradigmas son en realidad una manera de manejar la complejidad , la programación orientada a objetos fue adoptada por todo el mundo por una razón , esa razón fue que era mas sencillo abstraer los problemas , la POO se adapta o parece mucho a la forma “normal” en que pensamos , nuestro mundo esta hecho de objetos que a su vez están hechos de otros objetos, hasta aquí todo es bonito , la abstracción y el encapsulamiento parecen hacer todo mas fácil pero…. el software y la programación evoluciono mientras la POO se extendía y se convertía en lo mas usado, algunas reglas básicas fueron olvidadas , se hizo mal uso de la POO y esta elimino o permitió manejar algunas complejidades pero introdujo otras como la mutabilidad y el no determinismo derivado del mal uso de los objetos , por otro lado niveles de abstracción mucho mas difíciles de comprender fueron apareciendo como los patrones de diseño y las arquitecturas , aprender programación estructurada era muy sencillo , aprender POO se puede tardar mucho mas y creo que esta es la razón principal por la cual algunos de los antiguos programadores critican la POO , es mas compleja de aprender.

En ocasiones pienso que algunas abstracciones son muy difíciles de entender y me pongo a pensar si vale la pena utilizar estos “patrones” , aprender 24 patrones de diseño , aprender cuando y como utilizarlos , y después aprender una serie de patrones arquitectónicos.

La programación funcional y el modelo basado en actores en mi opinión ofrecen una manera mas sencilla de manejar la complejidad ya que eliminan muchas de las malas practicas de la POO y la necesidad de muchos de los patrones.

Salvador Martínez, estudió Sistemas Informáticos (1991)
Contestado 19/6/2018 · El autor tiene **323** respuestas y **38,5k** vistas de respuestas

Hace años podría haber tenido sentido esa pregunta porque los medios eran menores y los lenguajes , con excepción si acaso del C++, apenas contemplaban esa posibilidad. Ahora sería como decir: ¿Las funciones son necesarias? ¿Realmente sólo complican las cosas?. Serían preguntas sin sentido, puesto que ellas son la base de la programación y en ese mismo aspecto

las clases - objetos, son capaces de agruparlas, junto con las variables (las cuales pueden ser también objetos a su vez, así como estructuras de datos). Ahí ya no hablamos de una tarea repetitiva que se puede programar en una función, sino de funciones y variables que interaccionan entre sí y que ayudan muchísimo a formar librerías, para tareas específicas. Para que te hagas una idea es como “subir un nivel” en la abstración de lo que se quiere realizar. Personalmente el java a veces me resulta incómodo, porque obliga a trabajar con clases en cada módulo que se necesite. Otros lenguajes como el C++, PHP, VB, Perl, Ruby o Python dan a elegir. He programado bastantes años en Clipper y sólo tenía 4 objetos, sin embargo me ayudó a discernir la ventaja de tenerlos y con el C++ aprovechar la ventaja de poder crear mis propios objetos. Una vez que se aprende (lo cual en sus aspectos básicos, no es nada complicado), la ventaja es indiscutible. La lentitud que mencionas hoy en día no tiene sentido tenerla en cuenta salvo casos muy especiales que por ejemplo programando en C++, apenas se notaría.

Juan Antonio Gómez Gutiérrez, La estoy retomando con entusiasmo , me encanta abstraerme .
Contestado 26/7/2018 · El autor tiene **1,5k** respuestas y **86,6k** vistas de respuestas

Lo que complica las cosas es la mala , ineficiente o redundante programación . La potencia abstracta de la OO requiere un control exhaustivo de la gestión de memoria y un uso responsable de los efectos y causas "secundarios". El concepto "elegancia en el código" no es superfluo , es básico si se quiere separar efectivamente interface e implementación . Así tras más de 20 años alejado de la programación , en la actualidad veo que la evolución de la orientación al objeto se ha centrado en el desarrollo de lenguajes derivados de C++ que sin renunciar al paradigma de objetos ,palian los riesgos de mala gestión de memoria y de inseguridad en salidas y entradas ocultas. Es decir , han reducido las plasticidad en beneficio de la seguridad .

Rafael Dellà, Soporte técnico (1998 - presente)
Contestado 13/4/2018 · El autor tiene **184** respuestas y **46,3k** vistas de respuestas

Bueno, hay detalles a analizar:

Sí, las cosas se complican un poco, pero no “innecesariamente”. El la POO aumentan las normas que hacen el código, aparentemente, más complejo, como la encapsulación, pero evita, y mucho, el uso de variables globales.

Otra cosa que aumenta un poco la complejidad es la sobrecarga de funciones y operadores, pero genera un código mucho más sencillo de leer. Aunque, si no eres cuidadoso, se te puede ir de las manos.

Por ejemplo, puedes sobrecargar el + para que sume texto y quedas como Dios. Y también puedes hacer que el + sume triángulos y peras, pero quedas como un cerdo xD

En cuanto a la lentitud del software… según:

Si es un programa compilado tipo el C++ lo que aumenta es el tiempo de compilación, pero no el de ejecución.

Si el programa no es compilado o tiene una gestión de errores muy potente que tiene en cuenta la información del código fuente (como el Java) sí. El software es más lento porque, básicamente,

en tiempo de ejecución debe tener en cuenta las “ventajas” de las sobrecargas de funciones, cuando se escribe el código, para saber qué función invocar.

Siguiendo el ejemplo de antes, el programa debe comprobar que las variables que le has pasado al operador de suma para saber si quieres sumar números, texto o triángulos con peras. Y esta comprobación, aunque trivial y rápida, consume tiempo.

Hay una característica más en el C++ en su implementación de la POO que afecta también al rendimiento del software final: las funciones virtuales. Éstas son complejas de tratar porque has de entender muy bien cuando se juntan con los constructores y destructores de las clases. Además, estas se evalúan siempre en tiempo de ejecución (en realidad, esa es su gracia) así que siempre son más “pesadas” de invocar que cualquier otra función donde la invocación es más básica.

Ahora bien, con las máquinas que tenemos hoy en día y a no ser que debas hacer el motor de búsqueda de google, el tiempo “perdido” por usar estas funcionalidades de la POO sin irrelevantes ante las ventajas.

Raul Islas, Administrador
Contestado 21/5/2018

Por algún lugar leí que el lado negativo de la OOP es su “Bloat” o traducido a algo asi como “código de más, útil pero no para la aplicación en cuestión” y daban un ejemplo algo así como que “si quieres un plátano, no solo obtienes el plátano sino al gorila y a la selva también !!!” Y con esto quiero decir que en términos de empleo de memoria, un simple “Hola Mundo” se lleva un buen de código interno (Las clases, estructuras y todo eso explicado antes”) pero si empleas código puro, será muy eficiente en cuanto a consumo de memoria. Al final, si nos vamos al origen de la OOP, no son mas que un Arreglo multidimensional que almacena, datos y código y que puede pasarse de código en código como parámero o referencia. Saludos.

Dany Garcell, Arquitecto de Software (2017 - presente)
Contestado 24/6/2018

Ya te lo digo yo, no. Actualmente pocas personas saben realmente aplicar la POO y se dedican a escribir código a la antigua. Tiene mucho de filosofía entender la POO. Pero no, no es verdad que la POO complique las cosas porque estás llevando a 0 y 1 el mundo real, lo que tienes en la cabeza. La vida va de objetos, comportamiento, estados y de eso va la POO.

Israel Laureano, estudió Ingeniería en Computación y Programación informática en Universidad Nacional Autónoma de México (1...
Contestado 12/7/2018 · El autor tiene **248** respuestas y **51,8k** vistas de respuestas

No, no es verdad.

La técnica de orientación a objetos abarca el análisis, diseño y desarrollo. Si tu haces un análisis y diseño funcional (basado en funciones) y luego quieres hacer tus desarrollos usando técnicas de programación a objetos entonces sí se cumple lo de complicar las cosas innecesariamente.

Anibal Antonelli, Estudié Ingeniería en Sistemas Computacionales en la Universidad Nacional del Centro de la Provincia de Bue...
Contestado 22/5/2018 · El autor tiene **63** respuestas y **11,9k** vistas de respuestas

Hola. no es verdad. Un buen diseño orientado a objetos simplifica las cosas, haciendo el software más mantenible y reutilizable.

Por otro lado los compiladores tienden a trabajar con cero costo de abstracción para lenguajes como C++ o Rust.

Saludos!