Ensayo

"No Silver Bullet" de *Frederick P. Brooks, Jr.*.

José Alejandro Gómez Castro

José Andrés Ramírez Ocampo

BIOSOFT

Patrones de Diseño

II Cuatrimestre

- 2016 -

Tabla de Contenidos

1. Introducción

Para muchas personas el software son solo programas de computadora, sin embargo nos comenta que son todos aquellos documentos asociados a la configuración de datos que se necesitan para hacer que estos programas operen de manera adecuada.

Para el diseño, análisis e implementación de proyectos de software se aplican metodologías, modelos y técnicas que permiten controlar las posibles variaciones que un sistema eventualmente podría generar.

En la actualidad se está haciendo un esfuerzo en el ámbito del diseño del software para reforzar la capacidad de representación en un modelo. Logrando atrapar con mayor profundidad el comportamiento e interrelación que debe tener las entidades que conformen un sistema.

Desde los años 50 existían metodologías de desarrollo, hecho que da lugar a la curiosidad y capacidad que los humanos han prevalecido y potenciado a lo largo de los años, inclinándose siempre a buscar la mejor manera posible de llevar a cabo una tarea.

“Un esfuerzo consistente y disciplinado para desarrollar, difundir y explotar estas innovaciones debería conducir a una mejora de un orden de magnitud. No hay un camino dorado, pero hay un camino”.

> No hay un camino dorado, pero hay un camino. Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering"

La anterior frase hace referencia a la indudable mejoría que los sistemas pueden alcanzar siguiendo un estricto código de esfuerzo.

2. Dificultades esenciales

“No sólo no hay balas de plata a la vista, sino que la misma naturaleza del software impide que las haya.” Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering"

El proceso de desarrollo de software está plagado de dificultades desde una conceptualización del problema errónea, la gran cantidad con los que cuenta un sistema, así también los constantes cambios que sufre un sistema de software es por eso que en la actualidad se han estado realizando grandes avances en el área del desarrollo de software un ejemplo muy claro de esto son los patrones de diseño cuyo propósito es dar solución a problemas mediante una estructura definida.

Brooks menciona cuatro propiedades inherentes a la esencia irreductible de los modernos sistemas de software: complejidad, conformidad, variabilidad e invisibilidad.

2. 1 Complejidad

Brooks menciona que las entidades de software son más complejas debido a su tamaño que, posiblemente, cualquier otra construcción humana porque no hay dos partes iguales es decir, el software difiere enormemente de los ordenadores, automóviles o edificios donde estos cuentan con muchas partes repetidas en el caso del software esto no es así ya que el software debe ser adaptado a las necesidades, forma de trabajo y presupuesto de un cliente específico. Esto no quiere decir que desarrollar software signifique formular una solución a la ligera ya que se cuentan con patrones de diseño así como también conceptos fundamentales según el paradigma en el que se desarrolle que ayudan a guiar la organización de los conceptos del dominio del problema.

Una de las causas de mayor complejidad en el desarrollo de software es la magnitud de los sistemas actuales donde la falta de comunicación y errores conceptuales pueden terminar en un producto deficiente. Para contrarrestar esta problemática es necesario contar con una buena base conceptual además de un conocimiento adecuado del problema.

La complejidad es algo inherente de los sistemas de software pero si contamos con

las armas adecuadas es posible crear diseños adecuados ese uno de los principales objetivos de los patrones de diseño.

2. 2 Conformidad

“La mayoría de la complejidad que él [ingeniero de software] debe controlar es arbitraria, forzada sin ritmo ni razón por las muchas instituciones humanas y sistemas a los que debe ajustarse. Estos difieren de interfaz a interfaz, y de un momento a otro, no por necesidad sino sólo porque fueron diseñados por gente distinta, en lugar de por Dios.” Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering"

En el párrafo anterior se habla sobre la gran dificultad con la que el desarrollador tiene que lidiar debido a los diferentes entornos a los que debe ajustarse en este ámbito los patrones de diseño ofrecen una gran ventaja ya que facilita en gran manera la compresión del diseño porque este está basado en una especificación estándar.

2. 3 Variabilidad

“El software está constantemente sometido a presiones para cambiarlo.” Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering"

El entorno cambiante que hay en las empresas hace que se debe modificar con regularidad los sistemas de software. Brooks menciona que todo software que triunfa cambia es decir tiene la posibilidad de adaptarse fácilmente a nuevas funcionalidades.

El software debe ser débilmente acoplado y fuertemente cohesivo y debe estar cerrado para ser modificado.

2. 4 Invisibilidad

“El software es invisible e invisualizable.” Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering"

Es por esto que es necesario contar con una buena especiación del diseño, la utilización de los principios del paradigma y el uso de patrones de diseño para ayudar a la comprensión de las diversas estructuras que forman parte del sistema de software.

3. Solución de dificultades accidentales

Las dificultades accidentales o errores son accidentales son aquellas no inherentes del dominio del problema, Brooks define tres soluciones a tres dificultades accidentales diferentes.

4. Lenguajes de alto nivel

Esto dio paso a una mayor productividad a la hora de escribir el código ya que libró de las dificultades accidentales que venían del lenguaje de la dificultad de entender el código y dio paso a una escritura más natural, más cercana al mundo real.

5. Tiempo compartido

También aumentó significativamente la productividad de los desarrolladores gracias a la rápida respuesta que este brinda.

6. Entornos de desarrollo unificados

Los IDES significaron un avance muy grande hacia una mayor productividad ya que contaban con herramientas realmente útiles.
La dificultad accidental puede ser relacionada con el hecho de centrarse en lo esencial

Esperanzas de descubrir la plata

7. Ada y otros avances en lenguajes de alto nivel

Brooks comenta que Ada abarca características para animar a utilizar diseño moderno y modularidad.
Además también habla de que la gran contribución de Ada ha sido el cambio ocasionado en los programadores adoptando técnicas modernas de diseño de software.

8. Programación Orientada a objetos.

La orientación a objetos es un paradigma de programación enfocado principalmente en representar casos, tareas o eventos del mundo real, utilizando recursos que permiten manejar los elementos, que conforman la POO, que pueden verse como los pilares del paradigma, el marco de referencia de la orientación a objetos, según Booch [Booch 1996]: Abstracción, Encapsulamiento, Modularidad, Jerarquía… etc.

Es importante mencionar algunos de los beneficios adquiridos con este paradigma.
POO permite al programador escribir y entender código de una manera más natural para los humanos, basándose en simular o representar eventos o acciones propios del mundo real, pero esto implica conocer bien los conceptos utilizados dentro de un sistema orientado a objetos.
Mark Sherman de Dartmouth nos dice en CSnet News que debemos ser cuidadosos distinguiendo dos ideas separadas que aparecen bajo el mismo nombre
POO permite el modelado integrado de propiedades estáticas y dinámicas del ámbito del problema, facilitando la implementación, el mantenimiento, y la reutilización.

Con el paso del tiempo los programadores han sufrido varios problemas que resultan ser muy comunes cuando se desarrolla un software verdaderamente robusto y eficaz.
Este hecho de nuevo ha abierto nuevos horizontes para que los humanos expresen su creatividad, creando una solución que requiere una estructura adecuada, la comunicación entre estas y el resultado del comportamiento que se busca para solucionar el problema.
A esto se le llama patrones de diseño, que podría verse como la integración de pensamientos que otros programadores nos han brindado, consolidando soluciones a problemas que pueden aparecer en el entorno del software.

Como al interpretar un músico famoso y hábil, tratando de atrapar el sonido del compositor dentro de su canción, los programadores atrapan las mejores ideas consolidando buenas prácticas de software, aumentando su nivel de eficacia y precisión para resolver problemas.

9. Inteligencia artificial

La mayoría de la gente espera que avances en inteligencia artificial nos proporcionen el avance revolucionario que nos dará ganancias de varios órdenes de magnitud en la productividad del software y su calidad. Aunque Frederick P. Brooks, Jr. se refiere a la inteligencia artificial como si al momento de no volver a ver la arquitectura del software, perdiera su capacidad de expresar o generar un ente tecnológico con su propia inteligencia.

Es posible pensar y de hecho la forma de seguir el camino dorado que se menciona antes, el buscar la manera de crear especificaciones, algoritmos y soluciones que busquen su propia autonomía, además de ser incrementales y darle lugar cada vez a un sistema más grande y poderoso.

Actualmente podemos ver como se busca la manera de automatizar el funcionamiento de automóviles siguiendo un conjunto de especificaciones y validaciones que el software debe ser capaz de controlar, creando una entidad capaz de llevar a cabo una tarea que debía ser consumida por una persona. Si abordamos el tema desde las entidades de software que han venido a tomar cargos de seres humanos, podría ser la inteligencia de una persona plasmada en un sistema, pero gracias a esto y al beneficio que además produce, nos abre paso a muchos otros conocimientos que llevan al ente tecnológico como un compañero de trabajo, una nueva forma de vida, la inteligencia artificial.

10. Programación gráfica.

“A veces los teóricos, justifican la aproximación considerando los flujogramas como el medio ideal para el diseño de programas y proporcionando poderosos medios para construirlos.

Nada convincente, y mucho menos excitante, ha surgido de esos esfuerzos. Estoy convencido de nunca lo hará.” Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering"

Los modelos gráficos de los sistemas, diagramas de flujo y todas las diferentes herramientas gráficas que son utilizadas para representar un sistema y su comportamiento son la base fundamental del desarrollo de software ya que forman parte del diseño de la solución la cual da paso a la implementación. Es por esto que no comparto la idea que plante Brooks ya que como se mencionó anteriormente todas las herramientas gráficas que son usadas en el desarrollo de un sistema son de gran utilidad para la comprensión del problema, así como también son muy útiles en el momento en que se necesite realizar cambios en el sistema.

11. Verificación del programa

La verificación de un programa es esencial para comprobar que el mismo cumple con todas sus especificaciones de manera correcta y con esto ayudar a una mejor implementación.

Una herramienta muy útil para el manejo adecuado de las especificaciones son los patrones de diseño los cuales ayudan en la delegación de responsabilidades, encapsulamiento, entre otros conceptos importantes.

12. Grandes diseñadores

“La cuestión central sobre cómo mejorar el arte del software se centra, como siempre, en la gente.”Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering"
Para el refinamiento potencial de los sistemas de software es recomendable tomar en cuenta ciertos aspectos conceptuales que nos llevan a una visión más profunda y funcional de la implementación.

Crear aplicaciones débilmente acopladas y fuertemente cohesivas (Modularidad).

Un módulo debe estar cerrado para ser modificado, pero abierto para ser extendido (Reutilización).

Diseñar soluciones heurísticas que conforman una fuerte resistencia contra los enemigos del software y que además conducen a las buenas prácticas.

El uso de patrones para la conceptualización del software como elemento fundamental para determinar mayor eficacia y excelencia (Patrones).

13. Referencias

- https://d5127e38-a-62cb3a1a-s-sites.googlegroups.com/site/edarioit/file-cabinet/NoSilverBullet.pdf?attachauth=ANoY7cq5IoAec_524mrSKsJgn5Nf7lRHrRswlgXziLk9sWSytvFMuaK1DjqtnjAL8gaJqQ7iXyhK4SSR2CSnDlVxm8-nJQy3DOVEAAsDnOv-G4ewYxG_E0sI6u5BdUhPM5G4UMl7Q3P773jn7DLf_ctibHNx9f317Eyed256ipeUVzliObMP5CZLeIc1axS2fhwJCAGATlYOQKi8BP7WLMzFq3es1Q7u2iBb-Nrwa0R7sSy8LjFXQoI%3D&attredirects=1
- http://codigolinea.com/2015/04/01/no-hay-balas-de-plata-lo-esencial-y-lo-accidental-en-la-ingenieria-del-software/
- http://www.eumed.net/tesis-doctorales/2014/jlcv/software.htm
- Brooks. (1987). "No Silver Bullet—Essence and Accidents of Software Engineering”