No Silver Bullet

Esscence and Accident in Software Engineering

by Frederick P. Brooks, Jr.
(1986)

La leyenda del "Hombre-Lobo"

De todos los monstruos que tenemos como parte de las pesadillas de nuestro folklore, ninguno es más terrorífico que el **Hombre-Lobo**, porque se transforma –inesperadamente- de lo **familiar** a lo **horrorífico** ...

La leyenda del "Hombre-lobo"

Para el **Hombre-Lobo** buscamos una **Bala de Plata**, que pueda **mágicamente** "ponerlo a descansar"

Idea principal

El desarrollo de software tiene una complejidad única, que impide una solución mágica para las principales dificultades

El desarrollo de hardware ha mejorado en varios ordenes de magnitud en costo y performance, algo que no se aprecia con el software

Dificultad en el desarrollo de software

<u>Esencial</u>
Definir estructuras conceptuales y su funcionamiento

<u>Accidental</u>
Codificar esas estructuras

Esencia del Software

“La esencia de una entidad software es el resultado de una construcción de conceptos entrelazados: conjuntos de datos, relaciones entre los elementos de datos, algoritmos y de invocaciones de funciones.

Esta esencia es abstracta, en el sentido de que la construcción conceptual es la misma bajo muchas representaciones diferentes. Sin embargo es altamente precisa y ricamente detallada.”

Esencia del Software

“La parte difícil en la creación de un software es la especificación, diseño y prueba de la construcción de esta base conceptual, no el trabajo de representar y probar la calidad de la representación.”

“Si esto es cierto, construir software siempre será difícil. De forma inherente ***la bala de plata no existe****.”*

Propiedades inherentes de esta esencia irreducible de los sistemas de software:

- *Complejidad*
- *Conformidad*
- *Modificabilidad*
- *Invisibilidad*

Dificultades esenciales

Complejidad

En el software no se repiten cosas.
Todo es diferente

Incrementar software no es agregar cosas ya existentes, sino agregar cosas nuevas e integrarlas

Por su tamaño, el software puede tener muchos estados => es difícil de especificar (problemas de robustez, fallas de producto, deadlines, problemas de comunicación,…)

Conformidad (arbitrariedad)

Otras áreas de estudio como la física deben enfrentar complejidad, pero esta proviene de leyes naturales, que no son caóticas.

“Deben existir explicaciones sencillas de la naturaleza, porque Dios no es caprichoso o arbitrario ...”

En el software la complejidad proviene de personas con distintas ideas y necesidades.

También deben respetarse interfaces ya existentes, cosas que no pueden simplificarse cambiando solo el software

Modificabilidad

El software es fácil de modificar a diferencia de objetos físicos

El software requiere ser modificado pues su fin puede cambiar

Invisibilidad

El software es invisible y por lo tanto difícil de representar visualmente

Representaciones geométricas no capturan en forma efectiva la naturaleza del software y no permiten razonar correctamente acerca de el

Dificultades Accidentales

-Problemas de los métodos de producción actuales.

- Inicialmente las preocupaciones estaban en el hardware : que no se queme una válvula, conectar cables, etc

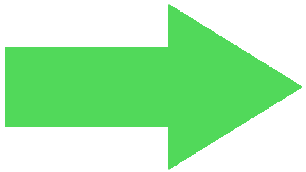
- Programación de bajo nivel.

No son problemas inherentes del desarrollo de software

Soluciones para dificultades accidentales

Lenguajes programación de alto nivel

```
MOV DX, Vec
MOV CX, 200
Inicio:
MOV [DX], 1
ADD DX, 2
LOOP Inicio
```



```
int i = 0;
int vec[200];
while (i < 200) {
  vec[i] = 1;
  i++;
}
```

Permiten concentrase en las entidades abstractas del problema y olvidarse de los detalles de la máquina

Multitarea Evita retrasos en el trabajo del programador

Entornos de desarrollo Equipamiento de herramientas y facilidades: librerías, formatos de archivos, etc

Esperanzas - Potenciales balas de plata

ADA y otros avances en lenguajes de alto nivel.

- Modularización.
- Tipos Abstractos de Datos
- Jerarquización.

“Es, después de todo, solo otro lenguaje de alto nivel”
Elimina las complejidades accidentales de la maquina

Programación orientada a objetos

Tal vez lo mejor que se ha logrado, elimina las dificultades accidentales de la expresión de diseño. Pero no puede eliminar la complejidad del diseño en si.

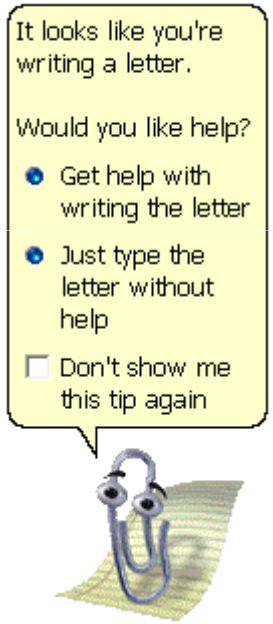
Inteligencia artificial

La IA que disponemos solo permite resolver problemas muy específicos, que no sirven para el desarrollo de software.

Sistemas expertos

Sistema experto = Motor de inferencia +
base de conocimientos

Dada la entrada de un problema intentan encontrar una solución. Podrían ayudar a sugerir interfaces y casos de test.
Se sigue requiriendo de un experto.



Programación automática

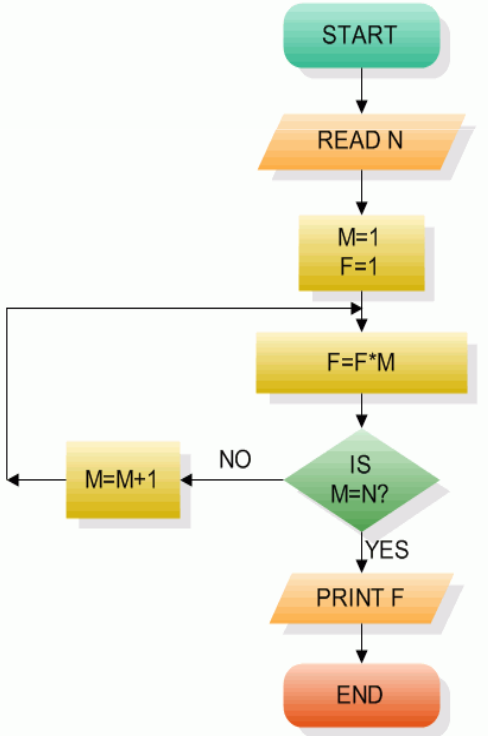
Dada una especificación una herramienta genera el código para resolver el problema.
Solo funciona para problemas caracterizados por pocos parámetros. La especificación debe ser determinada.

Programación gráfica

Se ha intentado muchas veces hacer cosas parecidas al diseño de circuitos integrados

Metáfora del asiento del avión. Solo se puede ver pocas cosas a la vez.

El software es muy difícil de visualizar



Verificación de programas	Es muy difícil verificar programas grandes y aun con un verificador perfecto solo se consigue saber que se esta cumpliendo la especificación. Lo mas complicado es llegar a esa especificación.
Entornos y herramientas	Un buen IDE puede ayudar, pero ya no parece haber mucho por hacer. Como mucho se pueden eliminar los errores sintácticos y algunos semánticos
Workstations	El incremento continuo de MIPS no va a hacer magia

Ataques prometedores para las dificultades esenciales

Buy versus build

La solución más radical de la construcción del software, es no construirlo.

Refinamiento de requerimientos y prototipos tempranos

Iteración y refinamiento de los requerimientos.

“La parte más difícil en la creación de un sistema software es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los detallados requerimientos técnicos, incluyendo todas las interfaces a las personas, maquinas y otros sistemas de software. Ninguna otra parte del trabajo paraliza los resultados del trabajo si se hace mal. Ninguna otra parte es mas difícil de arreglar después.”

“Por lo tanto, la función mas importante que el creador de software desempeña para el cliente es la extracción iterativa y refinamiento de los requerimientos del producto.
La verdad es que el cliente no sabe lo que quiere.”

Desarrollo incremental

Hacer crecer en vez de construir software.

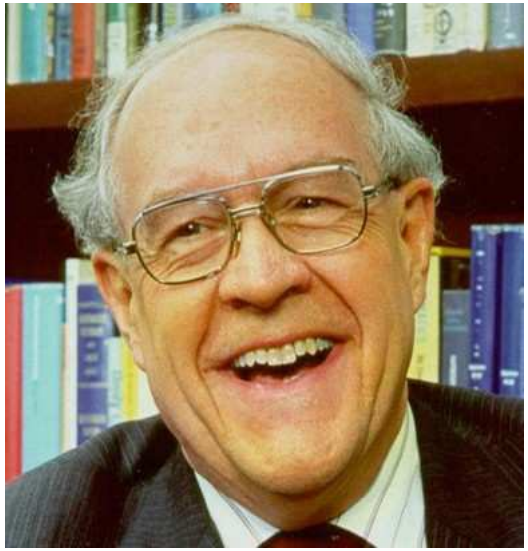
Buenos Diseñadores

“La cuestión central sobre como mejorar el arte del software se centra, como siempre, en la gente.”

Los mejores diseños de software son hechos por los mejores diseñadores.

“La construcción de software es un proceso creativo.”

Sobre el autor



- Nacido en 1931, North Carolina, USA
- Trabajó desde 1956 en IBM, entre otras cosas en el OS/360
- En 1975 publicó el libro *“The Mythical Man-Month : Essays on Software Engineering”*
- Turing award en 1999

Conclusiones.