No Silver Bullet. Essence and Accidents in Software Engineering. - Brooks

Gonzalo Castillo, Victoria Elizalde, Segio Gonzalez, Martín Page

FCEyN

Ingeniería del Software II

Introducción

- El paper No Silver Bullet. Essence and Accidents in Software Engineering. IEEE Computer, Abril de 1987.

Introducción

- El paper No Silver Bullet. Essence and Accidents in Software Engineering. IEEE Computer, Abril de 1987.
- Quienes Somos? Gonzalo Castillo, Victoria Elizalde, Sergio Gonzalez y Martín Page

Introducción

- El paper No Silver Bullet. Essence and Accidents in Software Engineering. IEEE Computer, Abril de 1987.
- Quienes Somos? Gonzalo Castillo, Victoria Elizalde, Sergio Gonzalez y Martín Page
- Y Brooks? Fred Brooks es Científico de la Computación e Ingeniero de Sofware. Recibió el Turing Award en 1999 y es conocido por haber escrito el libro The Mythical Man-Month, además de No silver bullet.

La metáfora

- Brooks compara un proyecto de software con un hombre lobo: algo inocente se transforma en un monstruo.

La metáfora

- Brooks compara un proyecto de software con un hombre lobo: algo inocente se transforma en un monstruo.
- Necesidad de una "bala de plata", algo que haga bajar costos y aumente productividad, confiabilidad y simplicidad.

La metáfora

- Brooks compara un proyecto de software con un hombre lobo: algo inocente se transforma en un monstruo.
- Necesidad de una "bala de plata", algo que haga bajar costos y aumente productividad, confiabilidad y simplicidad.
- Mayor dificultad del sofware: la especificación, diseño y testing de la estructura conceptual.

Las dificultades

Las dificultades

- *Dificultades Esenciales*: Inherentes a la naturaleza misma del software.

Las dificultades

- *Dificultades Esenciales*: Inherentes a la naturaleza misma del software.
  - **Complejidad**: Intrínseca del software(escalabilidad, numeración de estados, comunicación).

Las dificultades

- *Dificultades Esenciales*: Inherentes a la naturaleza misma del software.
  - **Complejidad**: Intrínseca del software(escalabilidad, numeración de estados, comunicación).
  - **Conformidad**: El software debe cumplir con limitaciones arbitrarias impuestas por personas y reglas de negocio.

Las dificultades

- *Dificultades Esenciales*: Inherentes a la naturaleza misma del software.
  - **Complejidad**: Intrínseca del software(escalabilidad, numeración de estados, comunicación).
  - **Conformidad**: El software debe cumplir con limitaciones arbitrarias impuestas por personas y reglas de negocio.
  - **Modificabilidad**: El software siempre va a estar sujeto a cambios.

Las dificultades

- *Dificultades Esenciales*: Inherentes a la naturaleza misma del software.
  - **Complejidad**: Intrínseca del software(escalabilidad, numeración de estados, comunicación).
  - **Conformidad**: El software debe cumplir con limitaciones arbitrarias impuestas por personas y reglas de negocio.
  - **Modificabilidad**: El software siempre va a estar sujeto a cambios.
  - **Invisibilidad**:El software es invisible e individualizable en el espacio. El software se intuye, pero no se ve.

Las dificultades

- *Dificultades Esenciales*: Inherentes a la naturaleza misma del software.
  - **Complejidad**: Intrínseca del software(escalabilidad, numeración de estados, comunicación).
  - **Conformidad**: El software debe cumplir con limitaciones arbitrarias impuestas por personas y reglas de negocio.
  - **Modificabilidad**: El software siempre va a estar sujeto a cambios.
  - **Invisibilidad**:El software es invisible e individualizable en el espacio. El software se intuye, pero no se ve.
- *Dificultades accidentales*: Dificultades no inherentes al software sino a su producción(Ej.Tipo de lenguaje de programación).

Avances que resolvieron dificultades accidentales

Avances que resolvieron dificultades accidentales

- Lenguajes de Alto Nivel: Abstracciones conceptuales. Esconden complejidad accidental del programa compilado.

Avances que resolvieron dificultades accidentales

- Lenguajes de Alto Nivel: Abstracciones conceptuales. Esconden complejidad accidental del programa compilado.
- Time-Sharing: La posibilidad de compartir el tiempo de ejecución entre procesos combate el accidente de los programas batch.

Avances que resolvieron dificultades accidentales

- Lenguajes de Alto Nivel: Abstracciones conceptuales. Esconden complejidad accidental del programa compilado.
- Time-Sharing: La posibilidad de compartir el tiempo de ejecución entre procesos combate el accidente de los programas batch.
- Ambientes de desarrollo unificado: Combaten el accidente de tener aplicaciones que resuelven en forma individual las problemáticas comunes (bibliotecas integradas, formatos de archivos unificados, tuberías y filtros).

Esperanzas y potenciales balas de plata

Esperanzas y potenciales balas de plata

- **Lenguajes de alto nivel y POO**: Ventajas de subclasificación e information hiding.

Esperanzas y potenciales balas de plata

- **Lenguajes de alto nivel y POO**: Ventajas de subclasificación e information hiding.
- **Inteligencia Artificial y Sistemas Expertos**: Conjunto de reglas de base y motor de inferencia para facilitar el desarrollo a principiantes.

Esperanzas y potenciales balas de plata

- **Lenguajes de alto nivel y POO**: Ventajas de subclasificación e information hiding.
- **Inteligencia Artificial y Sistemas Expertos**: Conjunto de reglas de base y motor de inferencia para facilitar el desarrollo a principiantes.
- **Programación Automática**: A partir de especificaciones generar código. Inviable y poco generalizable.

Esperanzas y potenciales balas de plata

- **Lenguajes de alto nivel y POO**: Ventajas de subclasificación e information hiding.
- **Inteligencia Artificial y Sistemas Expertos**: Conjunto de reglas de base y motor de inferencia para facilitar el desarrollo a principiantes.
- **Programación Automática**: A partir de especificaciones generar código. Inviable y poco generalizable.
- **Programación Visual**: Inviable por la invisibilidad del software.

Esperanzas y potenciales balas de plata

- **Lenguajes de alto nivel y POO**: Ventajas de subclasificación e information hiding.
- **Inteligencia Artificial y Sistemas Expertos**: Conjunto de reglas de base y motor de inferencia para facilitar el desarrollo a principiantes.
- **Programación Automática**: A partir de especificaciones generar código. Inviable y poco generalizable.
- **Programación Visual**: Inviable por la invisibilidad del software.
- **Verificación de Programas**: La verificación es costosa, no siempre aplica. Es suceptible a errores. Importancia de la validación.

Esperanzas y potenciales balas de plata

- **Lenguajes de alto nivel y POO**: Ventajas de subclasificación e information hiding.
- **Inteligencia Artificial y Sistemas Expertos**: Conjunto de reglas de base y motor de inferencia para facilitar el desarrollo a principiantes.
- **Programación Automática**: A partir de especificaciones generar código. Inviable y poco generalizable.
- **Programación Visual**: Inviable por la invisibilidad del software.
- **Verificación de Programas**: La verificación es costosa, no siempre aplica. Es suceptible a errores. Importancia de la validación.
- **Entornos y herramientas de desarollo**: Facilitan el trabajo de los desarrolladores(Ej. reducen errores sintácticos). Postivo, pero contribución marginal.

Ataques a dificultades escenciales

- **Comprar vs Contruir**: Si la construcción de software es tan difícil, entonces compremoslo!!.

Ataques a dificultades escenciales

- **Comprar vs Contruir**: Si la construcción de software es tan difícil, entonces compremoslo!!.
- **Prototipos y refinación de requerimientos**: El cliente no sabe lo que quiere. Es muy importante el feedback.

Ataques a dificultades escenciales

- **Comprar vs Contruir**: Si la construcción de software es tan difícil, entonces compremoslo!!.
- **Prototipos y refinación de requerimientos**: El cliente no sabe lo que quiere. Es muy importante el feedback.
- **Desarrollo iterativo incremental**: Es imposible contruir el producto en su totalidad de manera inmediata.

Ataques a dificultades escenciales

- **Comprar vs Contruir**: Si la construcción de software es tan difícil, entonces compremoslo!!.
- **Prototipos y refinación de requerimientos**: El cliente no sabe lo que quiere. Es muy importante el feedback.
- **Desarrollo iterativo incremental**: Es imposible contruir el producto en su totalidad de manera inmediata.
- **Buenos diseñadores**: Buen diseñador = buen diseño. Fomentar el crecimiento de buenos diseñadores.

Conclusiones

- Brooks fue un adelantado a su época, hace 20 años atrás tuvo visiones que incluso hoy en día tienen gran relevancia en el desarrollo del software.

Conclusiones

- Brooks fue un adelantado a su época, hace 20 años atrás tuvo visiones que incluso hoy en día tienen gran relevancia en el desarrollo del software.
  - *Validación usando prototipos.*

Conclusiones

- Brooks fue un adelantado a su época, hace 20 años atrás tuvo visiones que incluso hoy en día tienen gran relevancia en el desarrollo del software.
  - *Validación usando prototipos.*
  - *Procesos iterativos incrementales(PU, Scrum).*

Conclusiones

- Brooks fue un adelantado a su época, hace 20 años atrás tuvo visiones que incluso hoy en día tienen gran relevancia en el desarrollo del software.
  - *Validación usando prototipos.*
  - *Procesos iterativos incrementales(PU, Scrum).*
  - *Importancia de los buenos diseños.*

Conclusiones

- Brooks fue un adelantado a su época, hace 20 años atrás tuvo visiones que incluso hoy en día tienen gran relevancia en el desarrollo del software.
  - *Validación usando prototipos.*
  - *Procesos iterativos incrementales(PU, Scrum).*
  - *Importancia de los buenos diseños.*
- Reflexión del grupo: Comprar vs Construir ¿El futuro?