Designing software for ease of extension and contraction
David L. Parnas. Proceedings on the 3rd International Conference in Software Engineering, 1978.

David L. Parnas, nacido en Canadá (1941), pionero en la Ingeniería del Software, PhD en Carnegie Mellon University, con el título de Ingeniero Electrónico.

Problemas comunes con el desarrollo de software

- Concepción atomica del software. Un problema significa un "programa" que lo solucione

Problemas comunes con el desarrollo de software

- Concepción atomica del software. Un problema significa un "programa" que lo solucione
  - "Queremos preentergar una realese con solo un subconjunto de requerimientos, pero no funciona hasta que todo el resto funcione"

Problemas comunes con el desarrollo de software

- Concepción atomica del software. Un problema significa un "programa" que lo solucione
  - "Queremos preentergar una realese con solo un subconjunto de requerimientos, pero no funciona hasta que todo el resto funcione"
- No se toman en cuenta posibles cambios de requerimientos durante el desarrollo.

Problemas comunes con el desarrollo de software

- Concepción atomica del software. Un problema significa un "programa" que lo solucione
  - "Queremos preentergar una realese con solo un subconjunto de requerimientos, pero no funciona hasta que todo el resto funcione"
- No se toman en cuenta posibles cambios de requerimientos durante el desarrollo.
  - "Queremos agregar o quitar una pequeña funcionalidad pero hacerlo implica reescribir la mayoría del código"

Problemas comunes con el desarrollo de software

- Concepción atomica del software. Un problema significa un "programa" que lo solucione
  - "Queremos preentergar una realese con solo un subconjunto de requerimientos, pero no funciona hasta que todo el resto funcione"
- No se toman en cuenta posibles cambios de requerimientos durante el desarrollo.
  - "Queremos agregar o quitar una pequeña funcionalidad pero hacerlo implica reescribir la mayoría del código"
  - "Queremos simplificar o mejorar el código pero implica reescribir la mayoría del código."

Manifestaciones de estos problemas

- Excesiva distribución de información. Modulos dependen de información irrelevante a su naturaleza para poder operar.

Manifestaciones de estos problemas

- Excesiva distribución de información. Modulos dependen de información irrelevante a su naturaleza para poder operar.
- Arquitectura de pipeline sin flexibilidad. Se arman cadenas de transformaciones que no puede modificarse debido a incompatibilidad de datos de entrada y salida.

Manifestaciones de estos problemas

- Excesiva distribución de información. Modulos dependen de información irrelevante a su naturaleza para poder operar.
- Arquitectura de pipeline sin flexibilidad. Se arman cadenas de transformaciones que no puede modificarse debido a incompatibilidad de datos de entrada y salida.
- Componentes con mas de una función. Se crean modulos con multiples usos porque los requerimientos que solucionan son muy simples o porque usualmente estan muy relacionadas.

Manifestaciones de estos problemas

- Excesiva distribución de información. Modulos dependen de información irrelevante a su naturaleza para poder operar.
- Arquitectura de pipeline sin flexibilidad. Se arman cadenas de transformaciones que no puede modificarse debido a incompatibilidad de datos de entrada y salida.
- Componentes con mas de una función. Se crean modulos con multiples usos porque los requerimientos que solucionan son muy simples o porque usualmente estan muy relacionadas.
- Ciclos en la relación “usa”. Modulo A usa modulo B que a su vez usa A. Nada funciona hasta que todo funciona.

Familias de Software

Hoy el diseñador de software debería saber que **no** se está diseñando **un solo** programa sino una familia de éstos.

Algunas maneras en la que los miembros de una familia de programas puede diferir:

- Correr en distinto hardware

Familias de Software

Hoy el diseñador de software debería saber que **no** se está diseñando **un solo** programa sino una familia de éstos.

Algunas maneras en la que los miembros de una familia de programas puede diferir:

- Correr en distinto hardware
- Realizar la misma tarea pero diferir en el input/output

Familias de Software

Hoy el diseñador de software debería saber que **no** se está diseñando **un solo** programa sino una familia de éstos.

Algunas maneras en la que los miembros de una familia de programas puede diferir:

- Correr en distinto hardware
- Realizar la misma tarea pero diferir en el input/output
- En las estructuras o algoritmos de acuerdo a los recursos disponibles

Familias de Software

Hoy el diseñador de software debería saber que **no** se está diseñando **un solo** programa sino una familia de éstos.

Algunas maneras en la que los miembros de una familia de programas puede diferir:

- Correr en distinto hardware
- Realizar la misma tarea pero diferir en el input/output
- En las estructuras o algoritmos de acuerdo a los recursos disponibles
- En las estructuras o algoritmos de acuerdo al tamaño del input o la frecuencia de ciertos eventos

Familias de Software

Hoy el diseñador de software debería saber que **no** se está diseñando **un solo** programa sino una familia de éstos.

Algunas maneras en la que los miembros de una familia de programas puede diferir:

- Correr en distinto hardware
- Realizar la misma tarea pero diferir en el input/output
- En las estructuras o algoritmos de acuerdo a los recursos disponibles
- En las estructuras o algoritmos de acuerdo al tamaño del input o la frecuencia de ciertos eventos
- Algunos usuarios pueden requerir menos funcionalidad que otros y no quieren pagar por el resto

Familias de Software

Hoy el diseñador de software debería saber que **no** se está diseñando **un solo** programa sino una familia de éstos.

Solución

Reconciliar la metodologia ingenieril con la visión matematica de abstracción. Generar diseños más generales, para así poder anticiparse a los potenciales cambios que puedan surgir.

Hacia una mejor estructura

Se presentan 4 puntos que Parnas cree ayudarán a no tener los problemas mencionados.

A. Identificación de Requerimientos:
Identificar requerimientos y derivar de ellos los distintos subconjuntos de funcionalidades.

- Buscar primero el subconjunto **mínimo** que sea útil.
- Armar un conjunto de incrementos **mínimos** al sistema.

B. Ocultamiento de la información: Se pretende separar el comportamiento (modulo) de la semantica (interfaz) para que esta se mantenga válida a traves de las versiones

- Identificar los items que probablemente cambien. Estos son los "secretos".
- Ubicación de los componentes especializados en módulos separados.
- Diseñar interfaces entre módulos que no se vean afectadas por los cambios anticipados. Las partes que cambian se demoninan los "secretos" del módulo.

Esta es precisamente la intención de conceptos como encapsulado, abstracción y ocultamiento de la información

C. Concepto de ~~Virtual~~ Abstract Machine

- Intenta romper el concepto de la programación como una serie de transformaciones de un input (“pipeline mentality”)
- Diseñar cada modulo para que resuelve un problema general, es decir, tenga un alto de nivel de cohesión.

D. Diseñar la estructura de la relación "usa":

- Definir criterios estrictos para permitir al responsable del diseño aplicar una relación de usa.
- Evitar ciclos creando una jerarquia donde cada nivel solo usa al nivel inferior.

Un buen diseño jerarquico de relaciones crea una serie de propiedades deseables en un diseño.

- Soluciona el problema de "nada funciona hasta que todo funciona"
- Cada nivel define un subconjunto implementable iterativamente
- Se pueden usar como milestones.

Conclusiones

- Todas las soluciones que dió se utilizan hoy en día.

Conclusiones

- Todas las soluciones que dió se utilizan hoy en día.
- Lamentablemente seguimos sufriendo los mismos problemas que el encontró hace 30 años.

Conclusiones

- Todas las soluciones que dió se utilizan hoy en día.
- Lamentablemente seguimos sufriendo los mismos problemas que el encontró hace 30 años.
- Las metologias de desarrollo que estan vigentes hoy en dia se basan en los conceptos que el expuso.