

Introducción al diseño de TADs 1

Fernando Schapachnik¹

¹Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Algoritmos y Estructuras de Datos II,
primer cuatrimestre de 2015

(2) Hasta ahora...

- Nos preocupábamos por el *qué*, por ser *claros*.
- Hoy vamos a ver qué cosas hay que tener en cuenta cuando queremos pasar al *cómo*.
- Aprendamos de mi experiencia personal...

(3) Yo quería...



(4) Pero me dieron...



¡¿Qué falló?!

(6) Cambio de mundos

- Hubo un cambio de mundos.
- Nuevos elementos.
- Lo mismo pasa en el software.

(7) Más en concreto...

- Consideremos el TAD Conjunto.
- Veamos dos implementaciones posibles:
 - Un arreglo redimensionable.
 - Inserción (sin repetidos): $O(n)$
 - Búsqueda: $O(\log(n))$
 - Una secuencia.
 - Inserción (sin repetidos): $O(1)$
 - Búsqueda: $O(n)$
- ¿Cuál me conviene?
- Depende de qué necesite...


Lo que está claro es que no podemos pasar de la especificación al código directamente, necesitamos una *etapa intermedia*:

La etapa de **diseño**.

(9) Diseño de tipos abstractos

Qué significa diseñar un tipo:

- A nivel conceptual:
 - Preocuparnos no ya del *qué* sino del *cómo*.
 - Cambiar de paradigma (del funcional al imperativo).
 - Resolver los problemas que surgen como consecuencia de eso.

 En un plano un poco más concreto...

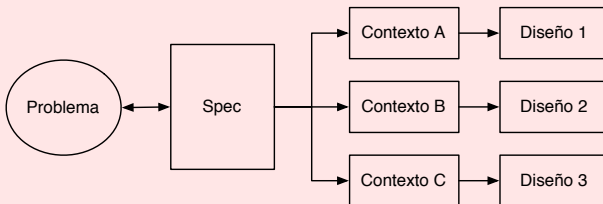
- Proveer una representación para los valores.
- Definir las funciones del tipo.
- Demostrar que eso es correcto.

(10) Volvamos al conjunto

¿Cómo discriminamos entre las dos soluciones?

Contexto de uso y
requerimientos de eficiencia. ⚠

Ejemplo



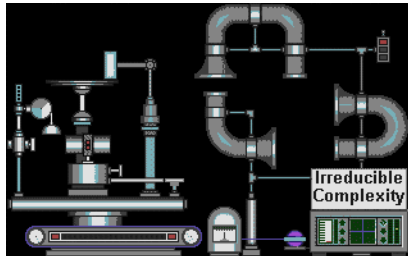
(11) Más en concreto...

- Consideremos el TAD Conjunto.
- Veamos dos implementaciones posibles:
 - Un arreglo redimensionable.
 - Inserción (sin repetidos): $O(n)$
 - Búsqueda: $O(\log(n))$
 - Una secuencia.
 - Inserción (sin repetidos): $O(1)$
 - Búsqueda: $O(n)$
- ¿Cuál me conviene?
- Depende de qué necesite...

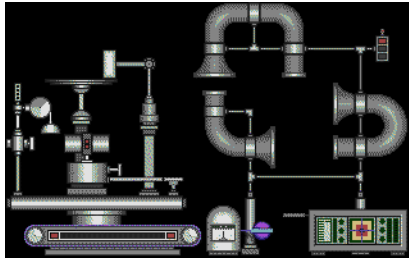
Analicemos conjunto sobre secuencia:
No terminamos acá, de hecho tenemos
un nuevo “problema” por resolver.
¿Esto es realmente un problema?

(13) Un poco de historia...

En 1960, una implementación se parecía más o menos a esto...

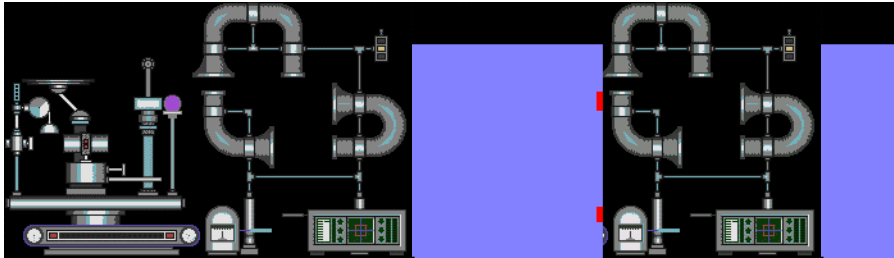


(14) Varios problemas...

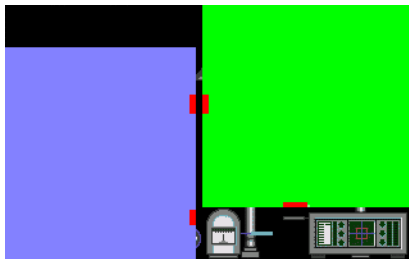


- Es complejo, difícil de entender.
- Las partes están muy interrelacionadas.
- Hay demasiada interacción entre las partes, lo que dificulta la comprensión.

(15) Una visión alternativa...



(16) Ventajas...



⚠ Cada fragmento presenta *interfaces claras*.

- Cada fragmento tiene una complejidad menor...
- ...y permite concentrarnos en *subproblemas* más fáciles de resolver.

(17) Método de diseño

⚠ Filosofía “top-down”.

⚠ Vamos a descomponer el problema en subproblemas.

- Técnicamente hablando, en módulos, que usarán a otros módulos.

⚠ Cada módulo dirá *qué hace y cuántos recursos necesita*, pero ocultará *cómo lo hace*.

Interfaz de Conjunto (fragmento)

Agregar(inout C: conjunto(nat), in e: nat) $O(1)$

$\{C \equiv C_0 \wedge e \notin C\}$

$\{C \equiv \text{Agregar}(C_0, e)\}$

Pertenece(in C: conjunto(nat), in e: nat) \rightarrow res: bool $O(\#C)$

$\{true\}$

$\{res \equiv e \in C\}$

⚠ Con un enfoque iterativo.

(18) Interfaz

- Lo que cada módulo da a conocer al mundo sobre sí se llama **interfaz**.
- En ella declara a qué tipo abstracto está implementando.
- Y para cada operación, describe:
 - Su signature.
 - Su pre y post condición (más sobre esto en un rato).
 - Su complejidad.
 - Otros aspectos que veremos más adelante.

(19) ¡Momento!

- Si llamo a *Agregar*(C, x), ¿consumo $|x|$ extra bytes? ¿Y si lo borro?
- Otra forma de preguntarse lo mismo: ¿el agregado es por copia o por referencia?
- Respuesta: miremos la especificación...
- ¿Qué sucede?
- Cambio de paradigma.

(20) Paradigmas comparados

- Funcional vs. imperativo.
- Abstracto vs. concreto (¿es tan así?)
- Manchas sobre papel vs. ejecución que toma tiempo.



Parámetros formales vs. parámetros de entrada/salida.

- in: su valor no puede alterarse dentro de la función.
- inout: su valor sí puede alterarse dentro de la función. **Cuidado en la post.**
- out: ídem inout, pero su valor a la entrada no importa, y podría no estar definido. **Cuidado en la pre.**

(21) Paradigmas comparados (cont.)

- Transparencia referencial.

Transparencia referencial



Una expresión E es *referencialmente transparente* si cualquier subexpresión y su correspondiente valor (el resultado de evaluar la subexpresión) pueden ser intercambiados sin cambiar el valor de E .

Ejemplo: si $f(x) := \{\text{return } x + 1\}$, $f(4) + f(3)$ es ref. trans. Si $f(x) := \{y = G(x + 1); G ++; \text{return } y\}$, no.

Transparencia referencial (definición práctica alternativa)

Una función es *referencialmente transparente* si su resultado sólo depende de sus parámetros explícitos.

(22) Aliasing


-  *Aliasing* es la forma en la que se denomina a tener más de un nombre para la misma cosa.
- En concreto, dos punteros o referencias hacia el mismo objeto.
 - Debido a que el paradigma funcional tiene transparencia referencial, no teníamos este “problema”.
 - ¿Es malo?
-  No, pero *debe ser* documentado porque ¡es tan público como la complejidad! (Ver el apunte y la clase práctica.)

(23) Aprovechándonos del cambio de paradigma

- Ejemplo: `Desencolar() + Próximo() → Desencolar()`
- `cambiosDeNombres` a `otros_formatos`
- Manejo de errores.
- Ejemplo: `Encolar(inout C: cola, in e: elem) → Encolar(inout C: cola, in e: elem, out s: status)`
- Tampoco nos abusemos: ¿qué pasa en las pre y las post?

(24) El sombrero como señal de respeto

- Analicemos: $\text{Ag}(\text{inout } C: \text{conjunto}(\text{nat}), \text{in } e: \text{nat})$
 $\{C \equiv C_0\} \{C \equiv \text{Agregar}(C_0, e)\}$
- Si C y e están “hechos de bits”, ¿qué significa \equiv ahí?
- Necesitamos a los términos “equivalentes” a C y e en el mundo funcional...
- Chapeau: \hat{C} y \hat{e} .

 Definición formal de $\hat{\cdot}$: lean el apunte.

(25) Dónde estamos

- Vimos
 - La diferencia de mundos.
 - Cómo los requerimientos de eficiencia deciden la implementación.
 - La idea de diseño top-down.
 - El cambio de paradigma.
 - Aliasing
 - Sombrerito.
- Es decir, por qué es necesaria la etapa de diseño, qué cambios introduce y cuál es el lenguaje que usamos.
- Veremos
 - Cómo escribir un módulo.
 - Qué cosas debemos considerar.
 - Cómo verificar su relación con el TAD.