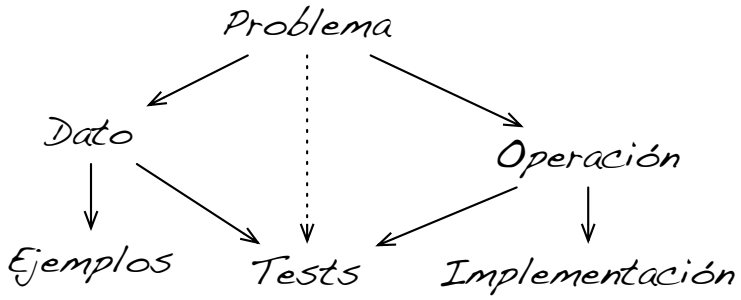
Como Diseñar Programas

Un Breve Resumen

Éric Tanter
http://www.dcc.uchile.cl/~etanter

29 de julio de 2008

Este apunte se basa en el enfoque detallado por Felleisen et al. [2], y resumido por Chris Dutchyn [1]. Se propone un enfoque para diseñar programas que, con gran confianza, hacen lo que se espera. El proceso consiste de cuatro pasos, centrados en la idea que programar implica manipular datos. Para asegurar que los datos se manipulan correctamente, se adopta un enfoque agíl, basado en tests.



Notese que los tests son independientes de la implementación. Esto asegura que los tests verifican las condiciones del problema, en vez de ser ciegamente determinados por el código.

Los cuatros pasos del proceso son:

1. *Diseñar los Datos* – antes de poder manipular datos, debemos decidir cuales datos son relevantes y como representarlos.
   a) Definición del tipo de datos, en términos de su *interfaz* (constructores y accesores)
   b) Ejemplos de construcción de datos
   c) Elección de una representación para el tipo de datos (*implementación* de la interfaz)
2. *Diseñar las Operaciones* – luego debemos seleccionar una operación requerida para completar el problema de programación y hacer un esqueleto — incluyendo casos de test.
   a) Propósito – descripción corta de la operación
   b) Contrato – una firma del tipo de la operación
   c) Esqueleto – un cuerpo válido en tipo, que no hace nada
   d) Tests – un conjunto de bloques de código que especifican operacionalmente la operación

   A esta altura, el programa debe compilar correctamente, aunque no haga nada correctamente.

3. *Implementar las Operaciones* – seleccionamos una de las operaciones boceteadas e implementamos su cuerpo. Para eso, nos basamos directamente en el diseño de datos para estructurar los casos del cuerpo. Luego implementamos — incluyendo compilar y verificar cada uno de sus casos de test.

4. *Testear* – nos aseguramos que el programa en su enteridad satisface la especificación completa — se pueden proveer casos de test adicionales:

   a) tests de cubrimiento – cada linea de código (que no sea de debugging) tiene que ser necesaria, y testeada

   b) evaluación de eficiencia – donde y cuando sea necesario, se puede optimizar las representaciones de los tipos de datos y las distintas operaciones. Pero ¡ojo!: *primero correcto, luego rápido*

Para progresar, nunca nos saltamos pasos:

- Nunca debemos postergar la escritura de los tests de una operación al despues de haberla implementada: esto nos llevaría a escribir tests basados en nuestra implementación particular, no en el problema inicial.
- Es posible que tengamos que volver atrás en el proceso para agregar o modificar un diseño de dato o una operación, o para agregar o reparar una implementación. En cada caso, nunca evitamos los pasos siguientes.

Despues de cada paso, nuestro código debe *funcionar* – es decir, debe compilar correctamente. En general, cada paso se tiene que repetir varias veces — hay muchas formas de diseñar datos, y muchas operaciones por diseñar e implementar para la construcción de un programa largo.

Es interesante notar que el trabajo intelectual de escribir programas se divide en tres partes distintas, estrictamente ordenadas en terminos de dependencia. La mayor parte de la escritura misma del código se vuelve automatica, requiriendo poco esfuerzo mental.

- *El Diseño de Datos* nos lleva a analizar el problema para determinar que información es relevante y sera modelada en nuestro programa, y que detalles pueden ser ignorados.
- *El Diseño de Operaciones* nos centra en el conjunto de operaciones (que manipulan los datos) que resolverán el problema; pero *no en como las tenemos que implementar*. Insistiendo nuevamente, la validación de nuestras operaciones se basa en el problema, no en el código. Escribir los tests *antes* del código impone rigurosamente esta disciplina.
- *La Implementación de Operaciones* parte con código prototípico que no requiere pensar: se basa directamente en el diseño de datos. Luego, se completa cada caso. Seleccionamos el algoritmo adecuado, y lo aplicamos.

Referencias

[1] Christopher Dutchyn. How to design SML programs, February 2008.

[2] Matthias Felleisen, Robby Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs*. MIT Press, 2001. http://www.htdp.org.