aprenderaprogramar.com

Abstracción y aprehensión. Diseño top-down de algoritmos. Parte 1. (CU00225A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel II

Fecha revisión: 2024

Autor: Mario R. Rancel

Resumen: Entrega nº 24 del Curso Bases de la programación Nivel II

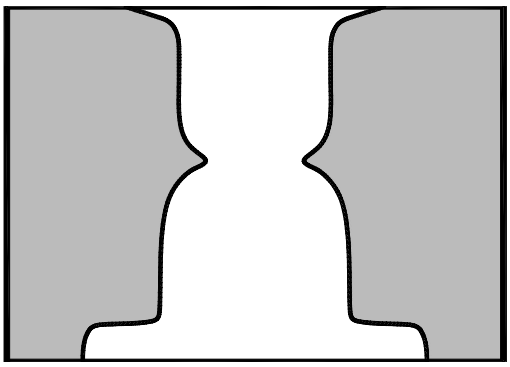
24

ABSTRACCIÓN Y APREHENSIÓN. EL DISEÑO TOP-DOWN DE ALGORITMOS

Utilizaremos abstracción en un sentido de "obtener la esencia" al identificar o percibir el problema. De elevar nuestro punto de vista para ir englobando las partes componentes. Veo a una pequeña hormiga circulando por la corteza de una rama. Si logro abstraerme puedo pasar a considerar el árbol en su totalidad, el bosque, la cuenca del río, la unidad bioclimática, el continente, el planeta Tierra... Es un camino hacia arriba, desde lo sencillo a lo complejo, desde las partes componentes hasta las unidades englobadoras.

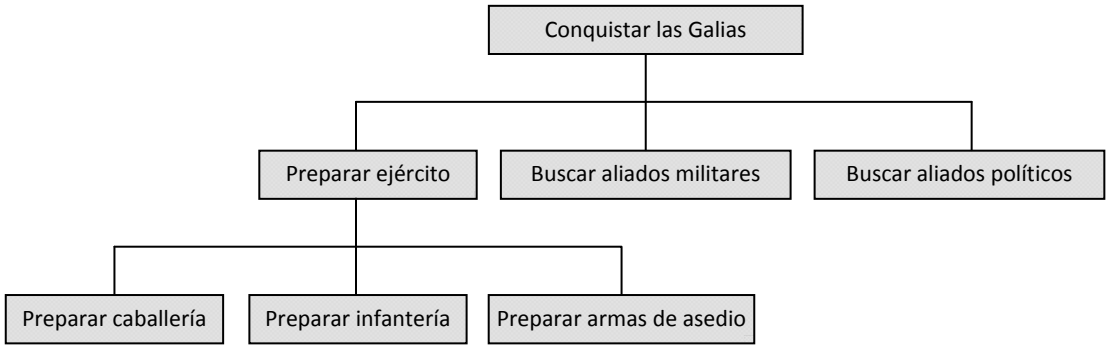
Utilizaremos aprehender en un doble sentido de coger, asir, abarcar y de llegar a conocer. De acercar nuestro punto de vista para ir descifrando, conociendo, desentrañando las partes componentes. Veo el planeta Tierra, el continente, la unidad bioclimática, la cuenca del río, el bosque, un árbol, una pequeña hormiga circulando por la rama del árbol... Es un camino hacia abajo, desde lo complejo a lo sencillo, desde las unidades englobadoras hasta las partes componentes.

La abstracción del problema. La aprehensión del problema. ¿Por qué unas personas ven cosas que otras no ven teniendo delante lo mismo?



En la figura tenemos el conocido efecto óptico de la copa y las dos caras. Según consideremos como fondo la parte blanca o la parte gris veremos una cosa u otra. Los procesos de abstracción y aprehensión están en buena medida condicionados por nuestras características personales, nuestras capacidades... Pero también por el ejercicio y esfuerzo, el empeño y el uso de métodos o enfoques adecuados.

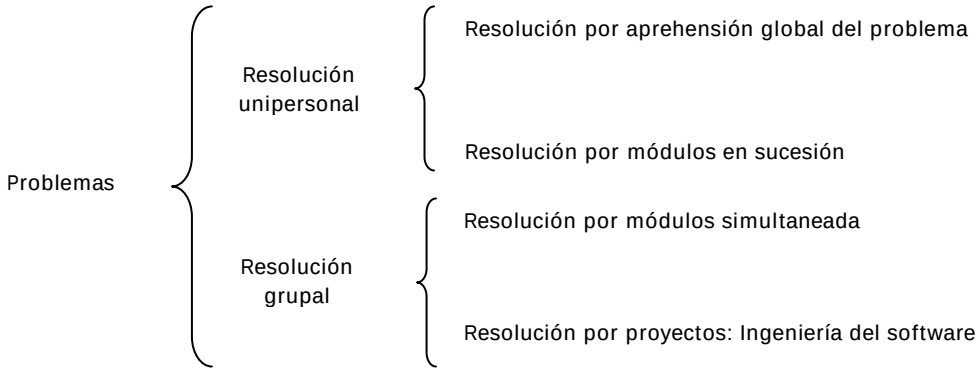
Analizaremos ahora lo que los programadores llaman diseño *top – down* de algoritmos desde una perspectiva un poco libre. Existen desarrollos teóricos más o menos detallados, pero nos conformaremos con "aprehender el espíritu". El llamado diseño *top – down* (arriba – abajo) de algoritmos o programas, no es otra cosa que trasladar al ámbito de la programación una estrategia de resolución de problemas muy antigua, basada en la capacidad humana de abstracción. Quizás algún emperador romano llegó a plantearse un esquema parecido al siguiente:



Partió de un objetivo. Su condición humana no le permitía aprehender el mismo: era demasiado abstracto, demasiado largo o complejo quizás para conocer todo lo que implicaba. Dividió el problema en partes componentes más fáciles de aprehender. Algunas ya le resultaban fácilmente comprensibles y ejecutables, pero otras seguían manteniendo abstracción y complejidad. Estas las volvió a dividir y así sucesivamente hasta que su objetivo complejo podía ser alcanzado a través de muchas acciones sencillas. Construyó el esquema desde lo general a lo particular, desde lo abstracto a lo aprehensible. Y lo ejecutó desde lo particular a lo general, construyendo abstracciones complejas a partir de elementos simples. Y conquistó Las Galias.

Cuando hablábamos de "Conocer el problema" planteamos la necesidad de dividirlo en extensión y complejidad para hacerlo programable. Y proponíamos una regla orientativa: "Sólo trataremos de programar aquello que mentalmente somos capaces de abarcar en método, extensión y condicionantes". Por otro lado clasificamos los problemas en de resolución directa, documentada, iterativa, a resolver con tabla de decisión o de resolución intuida pero método paso a paso a determinar.

Llega el momento de matizar planteamientos y de reestructurar nuestra clasificación de los problemas. Posiblemente ya somos capaces de resolver los problemas que mentalmente somos capaces de abarcar. Pero si sólo pudiéramos resolver los problemas que somos capaces de "aprehender", no habríamos llegado a desarrollar la energía atómica, los trenes de alta velocidad o los sistemas bancarios modernos... Entonces, ¿Cómo enfrentarnos a problemas que superan nuestra capacidad de aprehensión? Pues, de forma muy sencilla, aplicando los criterios de abstracción y concreción que estamos viendo y vamos a seguir desarrollando. La clasificación de problemas podemos definirla de una nueva manera:



Llamamos problemas de resolución unipersonal a aquellos que se ajustan a la capacidad de una persona. Pueden ser breves y directos, resolubles en unos pocos minutos, o largos y complejos, implicando varios años de trabajo. Pero están limitados terriblemente por el factor humano: una cabeza y dos manos. Por otro lado, la resolución grupal abarcaría desde dos hasta miles de personas trabajando en un mismo objetivo.

Aprehensión global del problema significa que, aunque lo resolvamos por partes o paso a paso, el proceso en su totalidad lo tenemos presente y conocemos o somos conscientes de todas las partes.

Resolución por módulos en sucesión significaría que hemos dividido el problema en partes y nos centramos en una de ellas, desconociendo o ignorando a las demás. El concepto de módulo aplicado en la clasificación es libre: no ha de coincidir exactamente con lo que hemos llamado módulo como elemento definido de programación. Podría sustituirse por "partes" o "componentes".

La resolución grupal sería la aplicable a problemas que se dividen en partes pero no para abordar una detrás de la otra, como estaremos obligados a hacer si trabajamos solos, sino para abordar varias partes simultáneamente o lo que se llama “atacar el problema por varios frentes”.

Entre resolución por módulos simultaneada y resolución por proyectos la diferencia estaría en la envergadura del programa a realizar y los medios que se utilizan. El simultaneo de módulos comprendería organizaciones de escasa entidad, desde dos amigos trabajando juntos hasta un pequeño departamento de una empresa con 7 u 8 personas. La ingeniería del software comprendería programas que requieren largos tiempos de desarrollo, con un ataque simultáneo de una estructura de definición muy ramificada y compleja. Estaríamos hablando de programas con centenares de módulos en los que pueden estar trabajando cientos de programadores. Se trataría de desarrollos propios de grandes empresas y marcas comerciales tipo multinacional como puede ser la española Indra o las norteamericanas IBM o Microsoft. También, con otra perspectiva, algunos desarrollos por parte de comunidades de programadores y programadores individuales organizados de forma relativamente espontánea, como es el caso del sistema operativo Linux[1].

Este curso se centra en problemas de resolución unipersonal, y puede también ser útil para resolución por módulos simultaneada a pequeña escala (en un gabinete de proyectos de ingeniería, en un negocio familiar o pequeña empresa). Tratar de resoluciones a gran escala o ingeniería del software requiere de mayor número de conocimientos en estructuras de datos, archivos y otros aspectos que los aquí tratados.

Podemos modificar ahora el enunciado “Sólo trataremos de programar aquello que mentalmente somos capaces de abarcar” por este otro, sin duda más ambicioso:

“Podemos desarrollar programas que exceden la capacidad de comprensión humana. Para ello descompondremos los problemas abstractos sucesivamente hasta llegar a partes componentes que mentalmente sea posible abarcar. El programa surgirá al integrar dichas partes formando estructuras cada vez más complejas”.

La actividad de un programador se centrará en una parte del programa que mentalmente es capaz de abarcar en método, extensión y condicionantes. Dicha parte es habitual que coincida con lo que hemos denominado módulo como elemento de programación, pero también es habitual que sea sólo una parte del módulo.

Obviamente la persona que coordina un equipo de programadores asume una gran responsabilidad materializada en el momento de realizar la integración entre las partes. Por poner un ejemplo, tres personas están trabajando en un módulo: una en aspectos estéticos relacionados con el diseño gráfico, otra en el algoritmo en sí y otra en la exportación de resultados desde el módulo hacia otros puntos o dispositivos. El trabajo puede ser independiente, pero bajo una dirección o coordinación que permitirán que la integración sea posible.

Veamos ahora el método para abordar un programa a partir de un diseño descendente. Si estamos habituados a la confección de índices para trabajos académicos o profesionales nos resultará muy sencillo. Partiremos de un objetivo o problema abstracto y lo dividiremos en partes componentes. Cada parte componente se dividirá a su vez en sus partes componentes y así sucesivamente hasta llegar a un nivel de no descomposición. El nivel en el cual se para la división depende del programador: de sus hábitos, gustos personales, conocimientos, experiencia, etc. Supongamos que después de subdividir varias veces un problema hemos llegado a que uno de los enunciados que tenemos es: Llamar a los

[1] En sus orígenes, Linux fue un desarrollo unipersonal de Linus Torvalds en Finlandia. Con el tiempo se fue incrementando el número de personas involucradas en el desarrollo del sistema pasando a ser un desarrollo simultáneo y finalmente, se convirtió en fenómeno global con una estructura de desarrollo tipo ingeniería del software.

bomberos. ¿Debemos seguir descomponiendo el enunciado? La decisión depende del grado de detalle que se quiera lograr y de lo que subjetivamente considere cada uno como fácilmente ejecutable. Porque en principio no nos hace falta descomponer este enunciado, pero si nos estuviéramos dirigiendo a niños que no saben manejar un teléfono del todo bien quizás deberíamos descomponer "Llamar a los bomberos" en "Descolgar el teléfono y esperar tono + pulsar 080" (suponiendo que sea este el número).

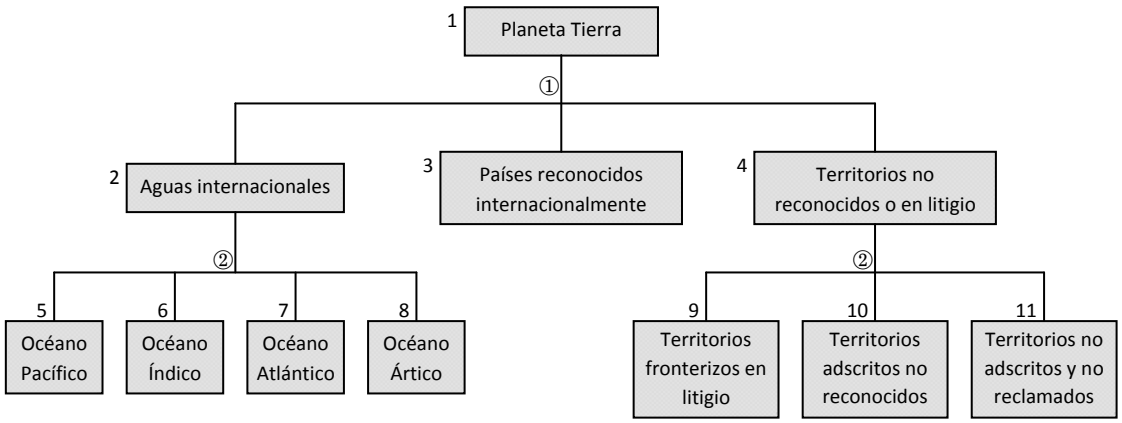
Definido el problema u objetivo, podemos realizar un índice descendente de contenidos, o, gráficamente, un esquema o diagrama descendente de contenidos. Supongamos el objetivo: "Estudiar la tierra desde el punto de vista político".

Índice descendente de contenidos [Ejemplo aprenderaprogramar.com]

1. Planeta Tierra
    - 1.1 Aguas internacionales
        - 1.1.1 Océano Pacífico
        - 1.1.2 Océano Índico
        - 1.1.3 Océano Atlántico
        - 1.1.4 Océano Ártico
    - 1.2 Países reconocidos internacionalmente
    - 1.3 Territorios no reconocidos o en litigio
        - 1.3.1 Territorios fronterizos en litigio
        - 1.3.2 Territorios adscritos no reconocidos internacionalmente
        - 1.3.3 Territorios no adscritos y no reclamados

La numeración es libre pero ha de reflejar cuáles son las partes componentes de cada estructura superior.

Esquema descendente de contenidos:



La numeración es optativa y libre, al predominar el componente gráfico. Su principal función es identificar un punto de partida en un esquema independiente.

Cada elemento del gráfico se denomina nodo. Cada nodo se encuentra en un nivel de concreción o definición. El nivel 0 corresponde al problema primigenio que no ha sufrido subdivisiones, el nivel 1 a

los nodos que se derivan de una única división, el nivel 2 a los nodos que se derivan de la división de nodos del nivel 1, el nivel 3 a los nodos que se derivan de la división de nodos del nivel 2 y así sucesivamente.

Un índice descendente de contenidos puede recordar, muy de lejos, a un algoritmo, y un esquema descendente a un diagrama de flujo. Pero obviamente son cosas muy distintas conceptualmente. En general, nosotros preferimos usar los esquemas descendentes para determinar los módulos del programa, o los módulos y esqueletos de los módulos, sin llegar a un nivel de subdivisión que alcance a bucles, instrucciones, etc. Pero es posible continuar descomponiendo hasta que aparecen nodos coincidentes con elementos del pseudocódigo o el diagrama de flujo. Aún así, la única coincidencia será la aparición de elementos similares, pero su organización y concepto siguen siendo distintos.

Próxima entrega: CU00226A

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=36&Itemid=60