7.3. Good Routine Names

Code Complete

* A good name for a routine clearly describes everything the routine does.
* In the routine's name, describe all the outputs and side effects.
* ¿Qué sucede si existen muchos efectos laterales?

The cure is not to use less-descriptive routine names; the cure is to program so that you cause things to happen directly rather than with side effects.

* Avoid meaningless, vague, or wishy-washy verbs.

Some verbs are elastic, stretched to cover just about any meaning. Routine names like HandleCalculation(), PerformServices(), OutputUser(), are OK.

* In other cases, the verb is vague because the operations performed by the routine are vague. The routine suffers from a weakness of purpose, and the weak name is a symptom.
* Don't differentiate routine names solely by number.

One developer wrote all his code in one big function. Then he took every 15 lines and created functions named Part1, Part2, and so on. The numerals at the ends of these names provide no indication of the different abstractions the routines represent.

* Make names of routines as long as necessary.

Research shows that the optimum average length for a variable name is 9 to 15 characters. Routines tend to be more complicated than variables, and good names for them tend to be longer. On the other hand, routine names are often attached to object names, which essentially provides part of the name for free.

* To name a function, use a description of the return value.

A function returns a value, and the function should be named for the value it returns. For example: cos(), customerId.Next(), printer.IsReady().

* To name a procedure, use a strong verb followed by an object.

A procedure with functional cohesion usually performs an operation on an object. The name should reflect what the procedure does, and an operation on an object implies a verb-plus-object name. Example: PrintDocument().

* In object-oriented languages, you don't need to include the name of the object in the procedure name because the object itself is included in the call. You invoke routines with statements like document.Print(), orderInfo.Check(), and monthlyRevenues.Calc(). Names like document.PrintDocument() are redundant and can become inaccurate when they're carried through to derived classes.
* Use opposites precisely. Using naming conventions for opposites helps consistency, which helps readability. Opposite-pairs like first/last are commonly understood.
* Establish conventions for common operations. In some systems, it's important to distinguish among different kinds of operations.

Example:

In one of our projects, we neglected to establish a convention for naming the routines that would return the object identifier, so we had routine names like these: employee.id.Get()

dependent.GetId()

supervisor()

candidate.id()

for the same operation.

## CHECKLIST: High-Quality Routines

### Big-Picture Issues

* Is the reason for creating the routine sufficient?
* Have all parts of the routine that would benefit from being put into routines of their own been put into routines of their own?
* Is the routine’s name a strong, clear verb-plus-object name for a procedure or a description of the return value for a function?
* Does the routine’s name describe everything the routine does?
* Have you established naming conventions for common operations?
* Does the routine have strong, functional cohesion—doing one and only one thing and doing it well?
* Do the routines have loose coupling—are the routine’s connections to other routines small, intimate, visible, and flexible?
* Is the length of the routine determined naturally by its function and logic, rather than by an artificial coding standard?

### Parameter-Passing Issues

* Does the routine’s parameter list, taken as a whole, present a consistent interface abstraction?
* Are the routine’s parameters in a sensible order, including matching the order of parameters in similar routines?
* Are interface assumptions documented?
* Does the routine have seven or fewer parameters?
* Is each input parameter used?
* Does the routine avoid using input parameters as working variables?
* If the routine is a function, does it return a valid value under all possible circumstances?

7.3. Buenos nombres de rutina

Código completo

Un buen nombre para una rutina describe claramente todo lo que hace la rutina.

En el nombre de la rutina, describa todos los resultados y efectos secundarios.

¿Qué sucede si existen muchos efectos laterales?

La cura no es utilizar nombres de rutina menos descriptivos; la cura es programar de manera que usted haga que las cosas sucedan directamente y no con efectos secundarios.

Evite verbos sin sentido, vagos o insípidos.

Algunos verbos son elásticos, estirados para cubrir casi cualquier significado. Los nombres de rutina como HandleCalculation (), PerformServices (), OutputUser (), están bien.

En otros casos, el verbo es vago porque las operaciones realizadas por la rutina son vagas. La rutina sufre de una debilidad de propósito, y el nombre débil es un síntoma.

No distinga los nombres de las rutinas únicamente por el número.

Un desarrollador escribió todo su código en una gran función. Luego tomó cada 15 líneas y creó funciones llamadas Part1, Part2, y así sucesivamente. Los números al final de estos nombres no proporcionan ninguna indicación de las diferentes abstracciones que representan las rutinas.

Haga nombres de rutinas siempre que sea necesario.

La investigación muestra que la longitud promedio óptima para un nombre de variable es de 9 a 15 caracteres. Las rutinas tienden a ser más complicadas que las variables, y los buenos nombres para ellas tienden a ser más largos. Por otro lado, los nombres de rutina a menudo se adjuntan a los nombres de los objetos, lo que esencialmente proporciona parte del nombre de forma gratuita.

Para nombrar una función, use una descripción del valor de retorno.

Una función devuelve un valor y la función debe tener el nombre del valor que devuelve. Por ejemplo: cos (), customerId.Next (), print.IsReady ().

Para nombrar un procedimiento, use un verbo fuerte seguido de un objeto.

Un procedimiento con cohesión funcional suele realizar una operación sobre un objeto. El nombre debe reflejar lo que hace el procedimiento, y una operación en un objeto implica un nombre de verbo más objeto. Ejemplo: PrintDocument ().

En los lenguajes orientados a objetos, no es necesario incluir el nombre del objeto en el nombre del procedimiento porque el objeto en sí está incluido en la llamada. Invoca rutinas con declaraciones como document.Print (), orderInfo.Check () y MonthlyRevenues.Calc (). Nombres como document.PrintDocument () son redundantes y pueden volverse inexactos cuando se transfieren a clases derivadas.

Usa los opuestos con precisión. El uso de convenciones de nomenclatura para los opuestos ayuda a la coherencia, lo que ayuda a la legibilidad. Los pares opuestos como primero / último se entienden comúnmente.

Establecer convenciones para operaciones comunes. En algunos sistemas, es importante distinguir entre diferentes tipos de operaciones.

Ejemplo:

En uno de nuestros proyectos, nos olvidamos de establecer una convención para nombrar las rutinas que devolverían el identificador de objeto, por lo que teníamos nombres de rutina como estos: working.id.Get ()

dependiente.GetId ()

supervisor()

Identificación del candidato()

para la misma operación.

LISTA DE VERIFICACIÓN: Rutinas de alta calidad

Problemas de panorama general

¿Es suficiente el motivo para crear la rutina?

¿Todas las partes de la rutina que se beneficiarían de ser puestas en rutinas propias, se han puesto en rutinas propias?

¿Es el nombre de la rutina un nombre de verbo más objeto claro y fuerte para un procedimiento o una descripción del valor de retorno de una función?

¿El nombre de la rutina describe todo lo que hace la rutina?

¿Ha establecido convenciones de nomenclatura para operaciones comunes?

¿Tiene la rutina una cohesión funcional fuerte: hacer una sola cosa y hacerlo bien?

¿Las rutinas tienen un acoplamiento flexible? ¿Son las conexiones de la rutina con otras rutinas pequeñas, íntimas, visibles y flexibles?

¿La duración de la rutina está determinada naturalmente por su función y lógica, más que por un estándar de codificación artificial?

Problemas de paso de parámetros

¿La lista de parámetros de la rutina, tomada como un todo, presenta una abstracción de interfaz consistente?

¿Están los parámetros de la rutina en un orden sensato, incluida la coincidencia del orden de los parámetros en rutinas similares?

¿Están documentadas las suposiciones de la interfaz?

¿Tiene la rutina siete parámetros o menos?

¿Se utiliza cada parámetro de entrada?

¿La rutina evita el uso de parámetros de entrada como variables de trabajo?

Si la rutina es una función, ¿devuelve un valor válido en todas las circunstancias posibles?