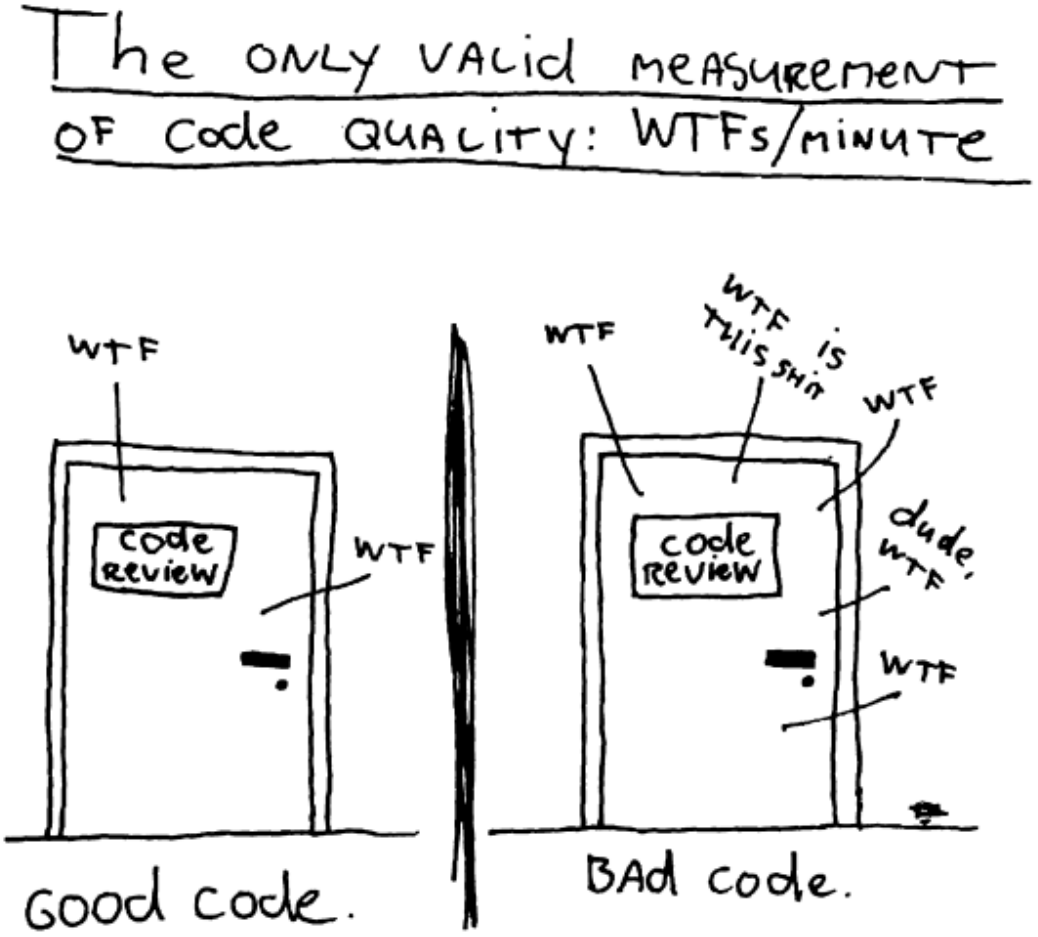
Tecnicatura Universitaria en Programación
Programación I

Apuntes para un código limpio

Nombres con sentido



Este breve resumen está construido con conceptos extraídos del libro **Clean Code o Código Limpio** de Robert C. Martin. Se recomienda tomar este apunte como un puntapié para leer dicho libro.

Usar nombres que revelen las intenciones

Debemos elegir los nombres correctos para nuestras variables, funciones, argumentos, archivos, etc.
Elegir nombres correctos lleva tiempo pero también ahorra trabajo. Por ello, hay que prestar atención a los nombres y cambiarlos cuando se encuentren mejores.

El nombre de una variable o función debe responder una serie de cuestiones básicas. Debe indicar por qué existe, qué hace y cómo se usa.

Si un nombre requiere un comentario, significa que no revela su contenido.

PseInt: Definir d Como Entero; //Tiempo transcurrido en días
 	
C: Int d; //Tiempo transcurrido en días

El nombre d no revela nada, no evoca una sensación de tiempo transcurrido, ni de días. Se debe elegir un nombre que especifique lo que se mide y la unidad de dicha medida.

```
Definir tiempoTranscurridoEnDias como Entero;
Definir diasDesdeCreacion como Entero;
Definir diasDesdeModificacion como Entero;
Definir diasDesdeNacimiento como Entero;
```

La elección de nombres que revelen intenciones facilita considerablemente la comprensión y modificación del código. ¿Para qué sirve el siguiente código?

```
1  Algoritmo malEjemplo
2      Definir da, ma, aa, dn, mn, an, resutado Como Entero
3      da← 12
4      ma← 4
5      aa ←2023
6
7      Leer dn, mn, an
8      Si (ma>mn o ( ma = mn y da≥dn)) Entonces
9          resutado ← aa - an
10     SiNo
11         resutado ← aa - an - 1
12     FinSi
13     Mostrar "Anios: ", resutado
14
15 FinAlgoritmo
```

¿Por qué es complicado saber que realiza? No hay expresiones complejas, sólo hay unas pocas variables y constantes todas enteras, sólo hay una estructura de control, etc.

El problema no es la simplicidad del código sino que el contexto del problema que resuelve no es explícito en el propio código. Veamos una segunda opción.

```
1  Algoritmo ejemploMejor
2      Definir dia_actual, mes_actual, anio_actual Como Entero
3      Definir dia_nacimiento, mes_nacimiento, anio_nacimiento, aniosHastaLaFecha Como Entero
4      dia_actual<- 27
5      mes_actual<- 4
6      anio_actual <-2021
7      Leer dia_nacimiento, mes_nacimiento, anio_nacimiento
8
9      Si ( (mes_actual > mes_nacimiento) o ( ( mes_actual == mes_nacimiento ) y (dia_actual ≥ dia_nacimiento) ) )
10         aniosHastaLaFecha ← anio_actual - anio_nacimiento
11     SiNo
12         aniosHastaLaFecha ← anio_actual - anio_nacimiento - 1
13     FinSi
14
15     Mostrar "Anios: ", aniosHastaLaFecha
16
17 FinAlgoritmo
```

La simplicidad del código no ha cambiado. Sigue teniendo los mismos operadores, constantes y variables, pero ahora es mucho más explícito. **Ahora es fácil saber qué sucede y es la ventaja de seleccionar nombres adecuados.**

Evitar la desinformación

Los programadores deben evitar dejar pistas falsas que dificulten el significado del código.

Debemos evitar hacer referencias a un grupo de cuentas como ***accountList*** a menos que realmente sea una lista (List). La palabra lista tiene un significado concreto para los programadores. Si se trata de un arreglo de cuentas un nombre mejor sería ***accountArray, accountGroup*** o simplemente ***accounts.***

Evitar usar nombre con variaciones mínimas. ¿Cuánto se tarda en apreciar la sutil diferencia entre ***cantidadCajasAmarillasEnDeposito*** contra ***cantidadCajasAmarillasParaDeposito***?. Ambos identificadores son muy similares.

La ortografía incorrecta también es desinformación, ***amount_rainfall_millimetres_per_month*** es muy distinto a ***ammount_rainfall_millimetres_per_moth.***

Realizar distinciones con sentido

Los nombres de series cómo ***a1, a2, …, an*** si bien son válidos como identificadores son los contrario a nombres intencionados. No desinforman pero no ofrecen información alguna.

```
Definir nota1 Como Real;
Definir nota2 Como Real;
Definir nota3 Como Real;
```

Notar la diferencia con los siguientes nombres escogidos cuidadosamente. Un pequeño cambio puede ofrecernos mucho más contexto.

```
Definir notaPrimerParcial Como Real;
Definir notaSegundoParcial Como Real;
Definir notaTercerParcial Como Real;
```

Las palabras adicionales son otra distinción sin sentido. Imagínese dos variables con los nombres ***aProduct*** y ***theProduct,*** son nombres distintos pero con el mismo significado. ***a, an, the*** son palabras adiciones que no deben utilizarse a menos que la distinción tenga sentido. Las palabras adicionales son redundantes.

Por otro lado, la palabra VARIABLE no debe incluirse nunca en el nombre de una variable, existen muy pocas excepciones cómo softwares matemáticos o estadísticos.

Las distinciones deben tener sentido. ¿Cómo saben los programadores qué función deben invocar?

```
getInformacionAlumno( );
getInformacionAlumnos( );
getInformacionAlumnoActivo( );
```

Por otro lado la variable ***cliente*** no se distingue de ***clienteInfo, clienteData*** o ***unCliente***.
Debe diferenciar los nombres de manera que otro programador/lector aprecie las diferencias por sutiles que estas fuesen.

Usar nombres que se puedan pronunciar

A los humanos se nos dan bien las palabras y por definición las palabras son pronunciables. Comunicar a un compañero sobre un error encontrado con respecto a una variable, constante o función que no se pueda pronunciar es un dolor de cabeza. Por lo tanto cree nombres pronunciables. La variable ***genymdhms*** para hacer referencia a la generación año, mes, día, hora, minuto y segundo. De más está decir que resulta impronunciable.

¿Cuál opción le resulta más fácil de pronunciar?

genymdhms o generacionTimeStamp menymdhms o modificacionTimeStamp

Evitar variables de una sola letra

Un contador para un bucle podría ser ***i, j, k*** ya que tradicionalmente las variables de una letra se utilizan para bucles. En otro contexto serían nombres muy pobres.

Es muy difícil recordar que la variable ***n*** se refiere a la cadena almacenada en la variable ***name*** pero todo en minúscula.

Evitar formas coloquiales o códigos internos del equipo

No recurra a bromas culturales o formas coloquiales de nombrar las cosas. No recurra a ***killThisS( )*** si lo que quiere decir es ***abortAction( )***.

Una palabra por concepto

Elija una palabra por cada concepto abstracto y manténgala. Por ejemplo resultaría confuso que en su código se utilice ***conseguir, recuperar*** o ***obtener*** indistintamente. ¿Cómo recordará cuando utiliza cada uno?

Misma palabra para uso distintos

Evite usar la misma palabra con dos fines distintos, ya que no sabrá qué esperar.

Utilizar nombres del dominio de la solución

Ya que los lectores del código son programadores se puede utilizar términos informáticos, matemáticos, algoritmos, físicos, etc. Son términos conocidos y comunican rápidamente el contexto, por ejemplo ***resolvente, pila, bucle***, etc.

Utilizar nombres del dominio del problema

Cuando no exista un término comúnmente aceptado por los programadores para lo que está haciendo, utilice nombres del dominio del problema. Ya que el programador que mantiene el código tiene el contexto de negocio y puede entender el significado.
Por ejemplo para un programa para una empresa de logística el nombre de la función ***crearHojaRuta( )*** tiene un significado reconocible rápidamente.

Separar conceptos del dominio de soluciones y del problema es trabajo de un buen programador.

Añadir contexto

Imagine los identificadores de variables ***nombre, calle, numeroCalle, ciudad, codigoPostal y Provincia***. Si se tienen todas en secciones cercanas del código está claro que son datos relativos a una dirección.
Pero si se utiliza la variable ***ciudad*** en alguna parte aislada del código ¿Sabría que se refiere a la ciudad, domicilio de la persona?. Para evitar esta situación se puede añadir contexto.

```
Definir dir_nombre, dir_calle, dir_nor_calle, dir_cod_postal, dir_ciudad, dir_provincia como Texto;
```

No añadir contexto innecesario

En un programa para la utn sería realmente redundante agregar el prefijo utn en cada nombre que seleccionemos. ***utnCarrera, utnCarreraMaterias, utnCarreraAnioComisio***, etc no aporta ningún significado distinto de ***carrera, carreraMaterias, carreraAnioComisio,*** etc

Conclusión

Si es posible aplique estas reglas y compruebe si mejora o no la legibilidad de su código. Se verán recompensas a corto y largo plazo.

Bibliografía

Martin, Robert C. (2009). Clean Code: A Handbook of Agile Software Craftsmanship

Versiones

Fecha	Versión
25/03/2023	1.0

Autores

María Mercedes Valoni