

Clean Code

Estilo y nombrado.



Para TrainingIT

Por Alberto Basalo

1 - Estilo y nombrado

Claridad con el menor esfuerzo.

"Solo hay dos cosas difíciles en Informática: invalidar la caché y nombrar las cosas"

--  **Phil Karlton**

Esta cita parece una broma, pero no lo es. Primero respecto a la caché; todo el mundo sabe crear una. Pero invalidarla es casi un arte. ¿Con qué criterio? ¿uso reciente, peso, reutilización tiempo de cálculo, tasa de refresco? **Cada caso requiere atención especial.**

Y poner nombre a las cosas (variables, funciones, clases, módulos y hasta programas) es igual de fácil. **Lo realmente difícil es poner nombre significativos.** De nuevo requiere de especial atención.

Pero recordemos que escribimos para otros humanos en un idioma sujeto a reglas sintácticas precisas. Entonces, ¿por qué no definir **reglas de estilo y semánticas**?

Al final cuando abres un libro, un periódico o una revista lo que esperas es poder entenderlo con el menor esfuerzo. Para ello los editores usan reglas para los tamaños de páginas, letras, márgenes y demás. Las aplican para todas las hojas y nuestros cerebros agradecen dicha homogeneidad.

El cerebro da un *like* a las repeticiones porque no le obligan a pensar. Vale para tamaños pero también para conceptos.

Usar nombres claros para expresar los mismos conceptos es el otro mantra para conseguir nuestro objetivo: **claridad con el menor esfuerzo.**

En los próximos temas estudiaremos detenidamente conceptos que permitan crear:

- Código **agradable** de leer, incluso bonito y elegante.
- Homogéneo, **sin sobresaltos.**
- **Que expresa** claramente una intención.
- **Creando un idioma** para nuestro negocio.

Te dejo con una reflexión que nos motiva a realizar este esfuerzo. La recompensa es clara porque **dedicamos más tiempo a leer que a escribir.** Así que ¡prestemos mayor atención a la escritura!

"El tiempo dedicado a la lectura es muy superior al dedicado a la escritura. Leemos código antiguo para escribir código nuevo. Facilitar la lectura es facilita la escritura."

--  **Robert C. Martin**

1.1 Estilo y orden.

Código agradable, bonito, elegante.

"Cada línea de código debe parecer escrita por la misma persona, sin importar el número de participantes."

--  **Quien lo vaya a leer**

Esta cita la firmaría cualquier lector de código. Seguro que conoces **esa sensación de abrir un fichero e inmediatamente saber quien es su autor**. Es como un rastro, un olorcillo que dejó detrás de sí. Y casi nunca es bueno.

Resulta muy desagradable cuando cada fichero huele distinto al anterior. Así que lo primero será **hacer la experiencia de lectura un poco más placentera**. Sensación íntimamente relacionada con la belleza.

Belleza

Algo que agrada a tus sentidos.

Si un programador escribe sobre belleza, mejor que cierres inmediatamente el navegador. Y menos este programador que ahora estás leyendo.

Respecto a la cualidad subjetiva de la belleza no tengo nada que aportar. Sé que está en los ojos del que la mira, y sé que hay gente capaz de generarla... y otros que a duras penas podemos admirarla.

Pero aquí no hablamos de la belleza como una expresión artística. **No valoramos el tema de color de tu editor...** aunque el mío es más chulo. 😊 Sería una discusión entretenida pero poco concluyente.

En cambio, hay consenso científico en decir que **los cerebros humanos valoran positivamente todo aquello que comprenden sin esfuerzo**. Y resulta que hay ciertas características objetivas que aportan esa dosis de placer neuronal:

Sencillez.

Las formas geométricas sencillas agradan a todo el mundo.

Armonía.

Facilidad para interpretar el siguiente objeto una vez conocido el anterior.

Repetición.

Caso extremo de la armonía, como por ejemplo la simetría y las series.

Pues esto mismo lo puedes aplicar al texto que escribes en tu editor. Que sea **sencillo de interpretar**, que haya una coherencia en su apariencia, y desde luego que sea todo igual en lo formal.

Bajando al detalle de qué hacer con el texto para facilitar su lectura, nos dicen que **lo primero es prestar atención a sus dimensiones.**

Belleza **horizontal**

Se trata de determina como se colocan y acumulan las instrucciones en las líneas físicas del editor.

Sangría y llaves en bloques

Por ejemplo. ¿Alguien leería cómodamente un código dónde todas sus instrucciones empiezan en la posición cero? Claro que no, porque nuestras instrucciones no tienen todas el mismo peso. Desde hace décadas lo resolvemos usando **márgenes específicos, llamados sangrías o indentaciones.**

Y hasta ahí el consenso. Hay lenguajes que obligan a un determinado tipo de sangrado, como Python o yaml. En otros lo dejan a criterio del programador. Originando *las guerras del tab*.

Tamaño de las líneas

Obvio que los libros y revistas físicos tienen un ancho. También lo tienen los blogs y demás medios online. Cierto que en unos dispositivos la lectura es más cómoda que en otros. Pero nadie, repito nadie, crea una aplicación de lectura cuyo contenido no se ajuste a los márgenes horizontales de la ventana.

Entonces ¿por qué el código escrito en un navegador tienen que ser distinto? ¿Es que acaso **nos gusta la barra de scroll horizontal?** ¿O es que queremos hacer ejercicios con el cuello mientras programamos?

Diréis que no es tan sencillo, que depende mucho del tamaño de letra, y del tamaño y resolución de la pantalla. Por supuesto, pero aún así podríamos **poner límites**, ¿no?

Belleza **vertical**

La preocupación en horizontal, debes rotarla y trasponerla al eje vertical. En este caso puedes preguntarte por el orden en el que escribes ciertas instrucciones.

Orden de las variables o propiedades.


- **estilo funcional:** Todas al inicio de su bloque o función.
- **estilo optimizador:** O quizás decidas que lo mejor es lo más cerca de su uso.

Orden de las funciones o métodos:

En este caso algunos lenguajes no te dan opción y te obligan a declarar las funciones antes de poder invocarlas. Pero en otros, especialmente en los orientados a objetos, puedes colocar tus funciones en el orden que estimes. Y, cómo no, esto ha dado lugar a otro par de estilos.

- **estilo revelador:** Primero todos los métodos públicos para revelar lo antes posible lo que hace el módulo; y más abajo, sólo para quien le interese, los métodos privados.
- **estilo newsletter:** En este caso cada método público tiene lo más cerca posible a los métodos privados en los que se apoya. Se intenta agrupar funcionalidades usando criterios semánticos en lugar de usar criterios sintácticos.

Belleza **interna**

Seguro que has oído eso de que la belleza está en el interior, es lo que decimos los feos . Pero aquí no hablaremos de esa belleza. La idea es prestar atención a **todos esos caracteres no leíbles** tan habituales en el desarrollo. Paréntesis, llaves, corchetes, comas, comillas de todo tipo...

Separadores de listas e instrucciones

Por ejemplo el uso de la coma final en los arrays

Espacios en las expresiones

Piensa en dejar algún espacio en blanco para que respiren las expresiones lógicas o aritméticas

- alrededor de los paréntesis
- alrededor de los operadores

Delimitadores de cadenas

La otra gran batalla de nuestros días. **Comillas dobles o simples** para las cadenas de texto.

Para finalizar este tema dedicado a reflexionar sobre algo tan etéreo como la belleza te dejo con estas preguntas en el aire.

*¿Cuántas líneas en blanco seguidas **realmente** necesitas?*

.

.

¿Sólo entre métodos o también dentro, entre sus instrucciones?

.

.

.

¿Tantas? .

.

.

.

.

¿En serio?

1.2 - Tamaños y límites

Código homogéneo, sin sobresaltos.

"No me gusta usar las barras de desplazamiento para leer tu código."

--  **Quien lo vaya a leer**

Código con Reglas de Estilo

Al hablar de establecer normas, reglas y límites siempre me sale un poco de sarpullido. No soy yo la persona más amante de la reglamentación que hayas visto.

Por eso me apoyo en los que saben de estas cosas, psicólogos y sociólogos, para hablaros en su nombre. Nos dicen, que a la hora de establecer una norma, sea de tráfico o de desarrollo, se pasa por tres fases.

Decisión:

Deliberadamente vamos a establecer reglas sobre ciertos aspectos, y vamos a dejar libres otros. ¿Cuáles regularemos?. Ya lo veremos, pero adelanto que serán los que tengan un mayor impacto y que al mismo tiempo se les pueda controlar su aplicación. Un ejemplo del tráfico sería la velocidad en carretera.

Opción:

Una vez determinado algo importante, viene el momento de ponerle letra, o en nuestro caso cifra. Las cifras límite tienen que ser asumibles pero eficaces. De nada vale limitar la velocidad de un automóvil en carretera a 10 km/hora o a 1000km.

Sentido:









Hay quien dice que las normas están para romperlas. Tampoco os animo aquí a la anarquía pero hay que ser razonables. En ciertas situaciones vamos a incumplirlas... pero a sabiendas. Esto es importante, porque la temeridad se reduce significativamente sabiendo que hay un límite.

Define unas reglas y haz que se cumplan señalando lo que es incorrecto.

Límites

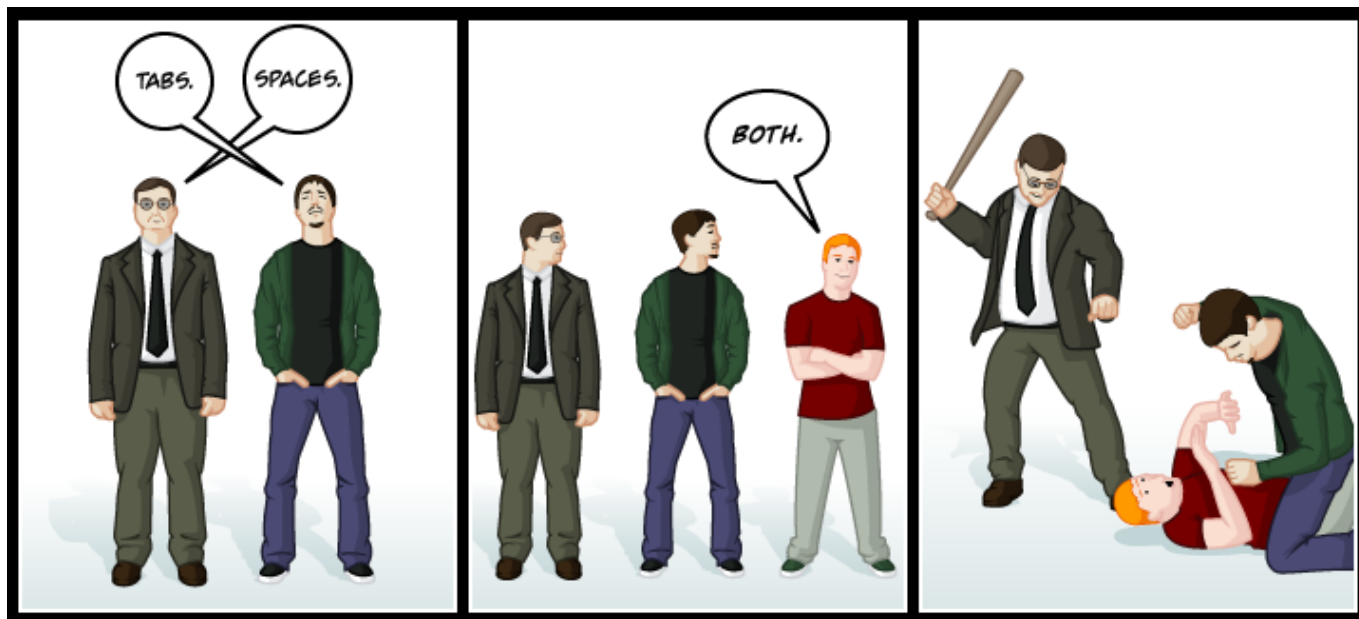
De lo anterior me daría por satisfecho si aceptáis que hay métricas de vuestro código que convendría encauzar dentro de unos límites. Las opciones concretas las tenéis que aportar dentro del equipo.

Te propongo unas horquillas con respecto a los límites aplicables al tema anterior sobre **estilo y orden en el código**

-  80 ↔ 120  caracteres por línea
-  1 ↔ 2  líneas en blanco seguidas
-  100 ↔ 200  líneas por fichero
-  2 ↔ 4  espacios de cada tab

☐ Consistencia

Te recomiendo que empieces por un conjunto reducido de normas. Ya verás que son más fáciles de aplicar que de cumplir. **Es mejor ser consistente con pocas normas... que cambiar de norma consistentemente.**



✂ Evita conflictos trasladando la decisión a otros:

No hay porqué llegar alas manos, como en la viñeta anterior, pero es verdad que establecer límites es fuente de conflictos. Ya hemos mencionada las infames tres guerras mundiales del desarrollo:

- Tabs vs Spaces
- Comillas simples o dobles
- Posición de llaves, paréntesis, operadores...

Lo ideal en estos casos es trasladar la decisión a otros. Un profesor podría valer.

✂ Herramientas de limpieza

Pero lo mejor de lo mejor es usar herramientas para garantizar el cumplimiento. En el mundillo web hay varias que además traen su propio catálogo de prescripciones dogmáticas. Muy útil para en caso de discusión sacar el típico *¿sabes tú más que los de Facebook?*

- [Prettier](#)
- [Beautify](#)

Busca según tu IDE y lenguaje porque las hay aplicables a todo tipo de situaciones.

🔗 Links de referencia

- [eslint-recommended](#)
- [These tools will help you write clean code](#)
- [Poetic](#)
- [Code guide for HTML & CSS](#)

- [12 Principles For Clean HTML Code](#)
- [Clean Code in SQL](#)

En cualquier caso la recomendación final es la siguiente: **haz que tu código sea más agradable de leer e independiente del autor.**

Corolario:

Estas normas pueden parecer triviales. Incluso superficiales o cosméticas. Pero no lo son.

Mantener **ficheros de tamaño reducido** obliga a encapsular el código. Mantener **líneas cortas** obliga no *indentar* demasiado anidando bloques. También dificulta escribir expresiones demasiado complejas en una sola instrucción.

Pero todo esto se verá más adelante.



1.3 - Definiciones con sustantivos

Expresa claramente una intención.

"Da sentido mediante los nombres."

--  **Quien lo vaya a leer**



Objetivo: Claridad

Un programa expresa un proceso con detalle en un lenguaje no ambiguo. Definición perfecta para el código que ejecuta un ordenador. Pero si queremos que alguien, seguramente nosotros mismos, lo podamos modificar en el futuro, entonces **el lenguaje debe además ser comprensible**.

Cuando elegimos un lenguaje (idioma) de alto nivel para escribir un programa es porque **queremos expresar claramente** lo que va a hacer durante la ejecución. Si no fuese así, deberíamos escoger lenguajes optimizados para el rendimiento u otros criterios. Pero, casi todo el desarrollo empresarial moderno escoge la claridad antes que la velocidad. ¿Por qué será?

Los desarrolladores profesionales nos enfrentamos siempre a problemas de negocio complejos. Si fuesen sencillos no gastarían el dinero en nosotros. Y hablando de dinero, el presupuesto que nos asignan cubre a duras penas las necesidades reales. Así que no iremos sobrados de tiempo.

Luego **tenemos problemas complejos y poco tiempo**. ¿De verdad que además queremos código enrevesado? Por supuesto que no, **queremos que el código se lea fácilmente y se entienda con el menor esfuerzo posible**. Y para ello, si podemos, debemos elegir la claridad ante cualquier otra característica.

Mostrar la INTENCIÓN

Al hablar de claridad en el código nos referimos a **mostrar en el lenguaje escrito la intención del programador** del proceso. El lenguaje (idioma) de programación suele ser de propósito general. Es decir sirve para desarrollar un video juego, una tienda online o una aplicación de gestión.

Así que el lenguaje de programación, per se, no nos va a ayudar a expresar la intención que tenemos durante el desarrollo. Tenemos que **enriquecer ese lenguaje, idioma, con un vocabulario propio** del modelo que estamos codificando.

La primera parada para crear este lenguaje (idioma) serán **los sustantivos**. Es posible que en tu juego aparezca el concepto de partida, jugador o puntuación. En una tienda online tendremos pedidos, artículos y precios. La gestión empresarial hablará de clientes, proveedores, facturas, citas o presupuestos.

Los conceptos de negocio que acabo de recitar forman parte de una análisis más o menos formal del dominio del sistema. Suelen ser los nombres de las entidades y sus atributos. En muchos casos aparecen en las bases de datos (estados congelados de uno o varios programas) o se accede a ellos mediante APIS (estados remotos de los programas).

Pues bien, el uso correcto de **ese vocabulario es la base de la creación de un lenguaje** (idioma) específico para tu propósito. Y en ese lenguaje creado por ti, será mucho más sencillo expresar y entender la intención del programador.

Explica lo que vas a almacenar.

Los programas de ordenador manipulan información almacenándola en recursos de hardware. Desde la memoria volátil de trabajo a un disco físico y remoto. Cada vez que almacenamos o recuperamos esa información tenemos que referirnos a su localización, su dirección en términos coloquiales. Pero las direcciones de memoria, de sectores o de servidores no son aptas para el consumo humano. Son la antítesis de la claridad. Así que para facilitar la labor **usamos alias inventados para nuestro favor. Háganse las variables.**

Una responsabilidad fundamental al programar es nombrar extraordinariamente bien las variables, constantes, clases, y propiedades de tus desarrollos. Sin excusas.

¿Y qué significa hacerlo bien? Pues consiste en dar **un nombre que explique claramente lo que se va a almacenar** en cada caso. Para que esto no se quede en una guía de buenas intenciones, tengo al intención de darte unos consejos que te sirvan de guía.

Guía para nombrar variables / propiedades / constantes / clases

Emplear siempre palabras completas y descriptivas.

```
// 🚫 Evita las abreviaturas:  
cli, numInvs;  
// 🧑 Usa siempre su versión completa  
client, numberOfInvoices;
```

Para que sean pronunciables y corregibles ortográficamente.

```
// 🚫 No hay quien lo pronuncia, ni detecte una mala escritura:  
pndOrdr;  
// 🧑 Usa siempre su versión ortográficamente correcta  
pendingOrders;
```

Definiendo un vocabulario de negocio.

```
// 🚫 No uses distintos nombres para el mismo concepto:  
client, customer;  
// 🧑 establece un vocabulario mínimo  
client;
```

Sin prefijos o sufijos técnicos.

```
// 🚫 No uses distintos nombres para el mismo concepto:  
ClientClass, intAmount;
```


```
// 🤖 establece un vocabulario mínimo  
Client, amount;
```

Agregando valor sin redundancias.

```
// 🤖 no te repitas  
client.clientAddress;  
// 🤖 aprovecha el contexto  
client.address;
```

Lo siento Harry, pero mejor sin magia.

```
// 🤖 No magic numbers:  
db = (h / 8) * 50;  
// 🤖 Define constantes y asígnales el valor.  
hoursByDay = 8;  
amountPerHour = 50;  
dailyBudget = (totalHours / hoursByDay) * amountPerHour;
```



int wtf = 42;
NOT a good name.

Reduce el número de WTF! 🤖

1.4 - Acciones con verbos

Creamos un idioma para nuestro negocio.

"Expresa la lógica con verbos."

--  **Quien lo vaya a leer**

Objetivo: Claridad

Desarrollamos programas para procesar datos, para manipularlos de alguna manera, aunque sólo sea almacenarlos para recuperarlos mas tarde. Esas son las acciones que nos encomiendan a los desarrolladores profesionales: guarda, recupera y manipula esta información.

Mostrar la INTENCIÓN

Así que los programadores profesionales hacemos lo que nos dicen y trasladamos esos deseos humanos a órdenes procesables. Pero de tal forma que la intención de los programadores quede meridianamente clara.

Explica lo que vas a hacer.

Para trabajar con las variables creadas emplearemos instrucciones. Según el lenguaje y paradigma usados las agruparemos en bloques lógicos que pueden llamarse, procedimientos, métodos, rutinas o funciones.



Lo importante es el nombre que le damos esos bloques. Porque será ahí dónde mostremos nuestra intención al escribir instrucciones para el ordenador.

Una responsabilidad fundamental al programar es nombrar extraordinariamente bien las funciones, métodos, o procedimientos de tus desarrollos. Sin excusas.

¿Y qué significa hacerlo bien? Pues consiste en dar **un verbo que explique claramente lo que se va a realizar** en cada caso. Para que esto no se quede en una guía de buenas intenciones, tengo al intención de darte unos consejos que te sirvan de guía.

Guía para nombrar funciones / métodos / rutinas / procedimientos

Obligatorio emplear siempre verbos que indiquen una acción.

```
//  qué hace este método?  
order.client(client);  
//  al comenzar por un verbo queda claro  
order.setClient(client);
```

Para que se lea como una historia.

```
// 🤖 Evita las abreviaturas:  
const client = clients.new(name, taxId);  
order.client(client);  
// 🤖 Usa siempre su versión completa  
const client = clients.create(name, taxId);  
order.setClient(client);
```

▶ Cortos y concretos en flags

is, has, can, must

```
// 🤖 esto no es agradable de leer  
if (client.pendingOrders()) {  
}  
// 🤖 facilita la lectura de las condiciones  
if (client.hasPendingOrders()) {  
}
```

👤 Define listas permitidas para acciones comunes


- Vocabulario para **relaciones y acciones**.
- Define listas permitidas para acciones comunes.
- `get | set - read | write - select | insert`


```
// 🤖 no mezcles  
clients.select();  
clients.post();  
orders.read();  
// 🤖 usa siempre el mismo tipo de verbos  
clients.select();  
clients.insert();  
orders.select();
```

🗨 Clarifica añadiendo sustantivos, adverbios o preposiciones.

```
// 🤖 Evita las sobrecargas:  
const client = clients.select(name);  
const client = clients.select(name, country);  
// 🤖 No te cobran por caracter  
const client = clients.selectByName(name);  
const client = clients.selectByNameAndCountry(name, country);
```

 Piensa en mi

 No me sorprendas

 No me hagas pensar