Código Sólido

Reflexiones sobre el desarrollo de código y principios SOLID

Gonzalo Ayuso

Código Sólido

Reflexiones sobre el desarrollo de software y principios SOLID

Gonzalo Ayuso

Este libro está a la venta en http://leanpub.com/codigosolido

Esta versión se publicó en 2018-03-16

Miro a mi alrededor y solo veo caras nuevas.

Índice general

¡Funciona!

Cuando eres novato, si tu código funciona estás más que satisfecho. Cosas como: el que sea legible, la duplicidad de código y en resumen la calidad, te parecen superfluas.

Si funciona está bien, ¿no?

Invertir en calidad de código parece sinónimo de *"tardar más"*. Tienes que hacer código que funcione. Parece lo único importante. Esas cosas de la calidad las dejas para más adelante y solo te preocupas de lo *"importante"*: que funcione. Es la típica mentalidad del programador novato. Nos pagan para que el código funcione y para que esté listo para ayer. No para mostrar nuestro código en un museo y que nos digan lo bonito que es.

Cuando te empiezas a curtir en batallas te das cuenta lo equivocado que estabas. Primer error: Lo importante no es que el código funcione. Por supuesto que tiene que funcionar. Eso se da por sentado. Pero lo realmente importante es que nuestro código sea **mantenible**.

¿A que dedica el tiempo el programador?

Parece una pregunta para el capitán obvio. El programador dedicará la mayor parte de su tiempo a programar, ¿no? En cierta medida esa afirmación es cierta. El programador dedica tiempo a programar. De hecho, si no dedicara tiempo a programar no sería un programador. Sería otra cosa. Pero la mayor parte del tiempo, al contrario de lo que muchos piensan, el programador no lo pasa programando. La mayor parte del tiempo los programadores lo pasamos **leyendo código**.

¿A que nos referimos cuando decimos que dedicamos tiempo a leer código? Pues cuando decimos esto no estamos diciendo abriendo una librería de código y leyendo las líneas como si fueran una novela. Estamos diciendo leyendo código para: Ver de dónde viene ese bug, donde pongo yo esta funcionalidad, etc. Esto se traduce en que el código que escribimos va a ser leído. Leído por personas como nosotros que necesitarán entender de qué va y no solo por el intérprete o por el compilador.

Nuestro código va ser leído.

Si nuestro código va a ser leído, entonces tenemos que hacer que sea *legible*. Puede parecer otra obviedad digna del capitán obvio, pero no lo es tanto. Cuando leemos código, este puede haber sido escrito por nosotros o por algún compañero. Si el código ha sido escrito por nosotros mismos, puede haber sido escrito hace unas horas o incluso años, por lo que la probabilidad de que no nos acordemos de lo que hemos escrito o, peor aún, el porqué de lo que hemos escrito, es muy alta.

La verdad está en el código.

Hace poco leí por ahí:

Todo equipo de programadores tiene al menos dos miembros: Tú y tú tres meses después.

No podemos depender en "recordar" el por qué hicimos algo. Nuestra memoria es limitada y a nada que pase el tiempo los detalles simplemente se nos olvidan. Hay quien piensa que el trabajo de un programador es mantener en su cabeza cuanta más información de manera que si tiene que tocar algo, corregir un bug o implementar una funcionalidad, solo tiene que tirar de su experiencia y tocar el código donde sea necesario porque *"sabe"* donde hay que hacerlo. Esto es un gran error. Puede ser que esa persona sea un genio y su memoria lo retenga todo, pero ¿Que pasa si esa persona cambia de trabajo, se pone enfermo o simplemente se toma unos días de vacaciones? Pero, aun suponiendo que lo tiene todo en su cabeza (que es ya de por si mucho suponer), ¿como transmitimos ese conocimiento al resto del equipo? No nos engañemos. Las cosas no son así. Si tenemos que recordar muchas cosas para que el sistema funcione, seguro que alguna se nos olvida y dado que seremos conscientes de ello trataremos de evitar tocar el código siempre que podamos, pasándole el marrón a otros con el fin de diluir nuestra responsabilidad. Esa mentalidad deriva en que cada programador solo se responsabiliza de "su código", porque es el que controla y sabe (o cree saber) como funciona.

He estado en reuniones junto a varios programadores en las que el objetivo de la reunión parecía que, más que como solucionar un problema y decidir quién lo tenía que implementar, era tener claro quien NO lo tenía que tocar. Cada uno (incluyéndome a mí, ojo) tratando que lo implementara otro con el fin de no tener que tocar "su código" y que los problemas fueran de otros. El código es del equipo. Da igual quien lo haya implementado. Todos tienen que entenderlo. Para conseguir esto tenemos dos opciones. Una es crear una maravillosa documentación técnica que todo el equipo lee y entiende. Todos los que nos dedicamos a esto sabemos que esto es una quimera. Esa maravillosa documentación técnica es como los unicornios. Suena muy bien, es muy bonita, pero no existe. Y encima cuando existe, queda obsoleta enseguida. Nadie puede asegurar si lo que pone ahí se corresponde con la última versión o está incompleta. En resumen, que si se da el caso que existiera, nadie se la lee. O si alguien se la lee, no se fía de ella (lo que es peor todavía). La otra opción es hacer que el código sea "legible" y que este (ósea el propio código) sea la documentación técnica.

Recordemos además el punto dos del manifiesto ágil:

Software funcionando sobre documentación extensiva

El código fuente nos cuenta como está implementado en programa. Las presentaciones, diagramas de flujo y similares nos cuentan una historia. A veces nos cuentan como pretendíamos que fuera nuestro software, pero el código fuente nos da la cruda realidad. Es por esto que pasamos tanto tiempo

leyendo código, para entender lo que pone y poder añadir/corregir funcionalidad y en definitiva, realizar nuestro trabajo. Es por esto también por lo que tenemos que tratar que sea legible.

Te pongo un ejemplo para intentar explicar todo esto. Imagina el siguiente código:

```
createUser(true);
```

¿Que hace esto? Podemos asumir que estamos creando un nuevo usuario, pero ¿que es eso del *true*? Esta duda nos obliga a entrar dentro de la función *createUser* .

```
function createUser($type) {
    …
}
```

Viendo la descripción de la función vemos que el *true* es para una variable que se llama *$type*. La verdad es que sigue sin aportarnos mucho. Nos obliga a inspeccionar con más detalle el código, hasta que nos damos cuenta que *true* significa que el usuario que estamos creando es de tipo administrador. Bien. El código *"funcionaba"*, pero hemos usado demasiado tiempo en entenderlo. Esto se traduce en una baja productividad. Recuerda eso de que el tiempo es dinero.

Quizás la persona que programó la función en una primera instancia no tardó mucho en desarrollarla. Esta persona tenía muy claro que era eso del true, la variable $type y demás, pero a ese tiempo de desarrollo hay que sumarle el tiempo que hemos tardado nosotros en entender esta función después. Lo peor de todo no es solo esto, sino que es que luego vendrá otro compañero y tendrá que hacer lo mismo, una y otra vez, aumentando así el tiempo de desarrollo. Lo malo es que medir estos tiempos totales no es nada sencillo. Medir el tiempo que tardamos en desarrollar la funcionalidad la primera vez sí que es sencillo: La fecha en que lo ponemos en producción menos la fecha en que lo hemos empezado a desarrollar. Sin embargo, el incremento de tiempo que le tenemos que añadir para entenderlo o recordarlo cada vez que tengamos que trabajar con él es muy difícil de calcular, pero esto no quiere decir que no exista. Existe y es real, pero que muy real.

Nos podríamos ahorrar todo esto si tuviéramos dos funciones:

```
createStandardUser();
createAdminUser();
```

Vale, escribimos más código en el momento inicial, pero ¿qué más da? Los futuros programadores que lean nuestro código (o nuestro yo futuro) no tendrán que preguntarse qué es lo que hace nuestro código e investigar por su cuenta. Lo tienen delante. Solo tienen que leer. Esto no implica que el código no pueda tener fallos, pero al menos le estamos haciendo la vida más fácil a nuestros compañeros (y a nosotros mismos), lo que se traduce en una mejor productividad y rapidez. Dicen que:

Tenemos que programar como si nuestro código fuera a ser leído por un asesino en serie que sabe dónde vivimos.

Quizás no hay que llegar a esos extremos (o quizás sí :).

Comentarios.

Una tentación del lado oscuro de la fuerza que nos puede surgir es la de usar comentarios.

```
1  createUser(true); // create admin user
```

Yo soy enemigo de los comentarios en el código, así en plan general. En bruto. Sé que es un tema espinoso, controvertido y que da para largas discusiones. Hay quien piensa que son necesarios y que un buen programador es aquel que pone buenos comentarios en su código. Hace poco discutiendo sobre esto con un camarada en un foro, me dijo:

El código *necesita* comentarios. Si no los necesitase es que es demasiado obvio.

En realidad, creo que dio en el clavo.

Los buenos programadores hacen que el código parezca obvio. Tan legible y descriptivo que los comentarios sobran.

Lo malo es que hacer esto es complicado. Bueno. Más que complicado, requiere experiencia, estudio, y mejorar día a día. Pero bueno. En eso estamos, ¿no? Yo, como digo, intento evitar los comentarios. Hay veces que los uso, pero soy consciente que el mero hecho de usarlos es un claro indicativo que estoy haciendo algo mal. O dicho de otro modo, quizás tendría que haber desarrollado en código de otra manera. Vamos que cuando me veo obligado a usarlos, en realidad me estoy avergonzando un poco de lo que acabo de realizar.

Esto lo dejo para después ...

> Yo primero hago que funcione y *luego*, con calma lo dejo bonito.

Es otro error. Más que un error es una ilusión. Otro unicornio. Ese *"luego"* nunca llegará. Nunca vamos a tener ese tiempo para dejar bien algo que ya funcionaba. Asumámoslo. Así que lo mejor para nosotros será intentar hacerlo bien desde el principio. Qué fácil es decir esto, ¿no? Fácil de decir

y muy complicado de llevarlo a cabo. Requiere años de experiencia y aprendizaje, pero esto no nos tiene que asustar. Cuanto antes comencemos a mejorar nuestras habilidades y conocimientos, antes empezaremos a ser productivos de verdad.

Hay veces que la falta de experiencia nos puede acomplejar. Está claro que un programador con experiencia es más productivo que uno sin experiencia.

Si hubieras empezado hace un año, hoy llevarías un año ganado

Nunca es tarde para empezar algo. Si empiezas hoy, dentro de un año ya llevaras 365 días. Sin embargo, si hoy no lo haces, dentro de un año estarás en el mismo punto que estas ahora. Además, este mundo de las tecnologías de la información es tan cambiante que estas cosas pueden incluso jugar a nuestro favor. Pensemos por ejemplo en JavaScript. A día de hoy este lenguaje es el rey de la programación de FrontEnd. Incluso se programa BackEnd usando JavaScript con nodeJS. Frameworks como AngularJs están tomando mucha fuerza. ¿Sabes cuantos años de experiencia tienen los gurús de AngularJS? Dudo que lleven más de 3 años, más que nada porque el framework no tiene mucho más tiempo de vida. Puedes decir: Vale, pero seguro que llevan muchos años con JavaScript. Lo dudo. Hace 7 años casi nadie programaba JavaScript (hablo de JavaScript en serio como se está programando hoy). Si que nos podemos encontrar a gente que lleve más de diez años programando en Java, PHP o Python, pero nunca es mal momento para empezar. Pero nadie nos asegura que el lenguaje X va a seguir vivo 10 años. ¿Que le pasará al JavaScript con la llegada de ES6? Si tienes una bola de cristal que te diga el futuro, entonces úsala. Si no, no te vuelvas loco con estas cosas. Mejora cada día con las herramientas del presente y así te adaptaras mejor a los cambios del futuro.

Productividad y eficiencia.

Estamos hablando de productividad y eficiencia. Esto es algo clave. Es lo que diferencia a un programador novato de uno senior. Muchas veces asociamos los conceptos programador novato a gente joven y programador senior a gente mayor. Hay que tener mucho cuidado con esto ya que no es siempre así. Las canas llegan solas. No hay que preocuparse por ello. Simplemente nos sentamos en el sofá y un buen día aparecen en nuestra cabeza. Con la experiencia no pasa lo mismo. Es obvio que para adquirir experiencia necesitamos que pasen los años, pero esta no nos va a llegar por si sola (los años y las canas si). Tenemos que invertir tiempo en asimilar conceptos, aprender cosas nuevas y ser capaces de dar soluciones a los problemas que nos van surgiendo con el paso de los años. En definitiva, aportar más valor lo que hacemos. El proceso de adquirir experiencia no es un proceso pasivo. Es muy, pero que muy activo. Si nos dormimos en los laureles simplemente conseguiremos que pasen los años sin adquirir experiencia. Con esto no conseguiremos más que seguir siendo novatos. Con canas en la cabeza eso sí, pero novatos.

Nadie dijo que fuera fácil

Si estás empezando en este mundo, te voy a hacer un pequeño spoiler:

Esta es una profesión muy dura.

Parece muy cool, pero te vas a pasar muchas horas delante de una pantalla sin ver la luz del sol. Es por esto que muchos acaban quemados abandonan la programación en cuanto pueden. Requiere mucho aprendizaje. No vale con aprender algo un día y vivir de eso durante años. Puedes aprender algo y vivir de eso unos años, pero como no estés aprendiendo cosas nuevas, te puedes plantar en un futuro cercano totalmente fuera del mercado. Hace falta una gran pro-actividad para adaptarse a los cambios y adquirir esa "experiencia" de la que hemos hablado antes. Esa experiencia que no es gratis y que la necesitamos si o si, si queremos seguir en esto del mundo del código. Si no seremos eternos novatos (con o sin canas en la cabeza) y tendremos que competir con todos los novatos que se van incorporando (y con sueldos de novatos por supuesto), por lo que nos quemaremos más rápido aún. Recordemos que las universidades y demás centros de formación son unas máquinas que ponen en el mercado programadores novatos año tras año.

Además, la capacidad de aprendizaje puede verse mermada con los años. No es lo mismo aprender con 20 años que con 40 o con 60. No es que no se pueda, es que la vida nos va metiendo en nuestra zona de confort y salir de ella nos puede dar pereza. También estarán nuestras obligaciones, compromisos, familia, etc. En definitiva, que con el paso de los años la cantidad de buenas excusas para no aprender y formarte aumentarán. Si queremos marcar la diferencia y aportar más valor hay que leer mucho y estudiar tecnologías que puede que queden obsoletas en poco tiempo. Todo esto puede llegar a quemarnos rápidamente. Avisado quedas. Pero por otro lado programar puede llegar a ser divertido. Es un ejercicio mental muy satisfactorio que engancha y nos depara un reto tras otro. Yo ya llevo casi dos décadas con esto y sigo con ganas. No sé qué pasará en el futuro, pero a día de hoy me sigo divirtiendo.