



¿Qué es el **Principio FIRST** ?
por Raúl Expósito

Imagen: <http://www.flickr.com/photos/34127470@N00/92967452/>



Índice

¿Qué contiene este documento?.....	03
Sobre el autor.....	03
Sobre la licencia.....	03
¿Qué cualidades deben tener nuestros tests?.....	04
Primera cualidad: Fast (rápido).....	06
Segunda cualidad: Independent (independiente).....	09
Tercera cualidad: Repeteable (repetible).....	12
Cuarta cualidad: Self-validating (auto evaluable).....	14
Quinta cualidad: Timely (oportuno).....	16

¿Qué contiene este documento?

El texto que vas a poder leer a continuación explica en qué consisten las cualidades que forman parte del **principio FIRST**, las cuales tienen como objetivo mejorar la calidad de los tests que prueben nuestro software.

Te encuentras ante un documento de carácter técnico que no entra en demasiado detalle para conseguir que cualquier persona pueda leerlo, aunque inevitablemente aparecerán algunos términos y vocablos propios de la jerga informática que se explicarán de una manera sencilla según aparezcan.

La maquetación de este documento está inspirada en la fantástica maquetación del documento "*Eres Productivo. Vol 1*" de Berto Pena:

<http://albertopena.com/descargas/>

Sobre el autor

Raúl Expósito es un ingeniero en informática residente en Getafe (Madrid) a quien le gusta materializar ideas, la creatividad, el diseño, el cuidado de los detalles y la simplicidad, definiéndose por tanto como una persona con vocación técnica pero a la vez creativa y con inquietudes.

Podéis encontrarle en su página web:

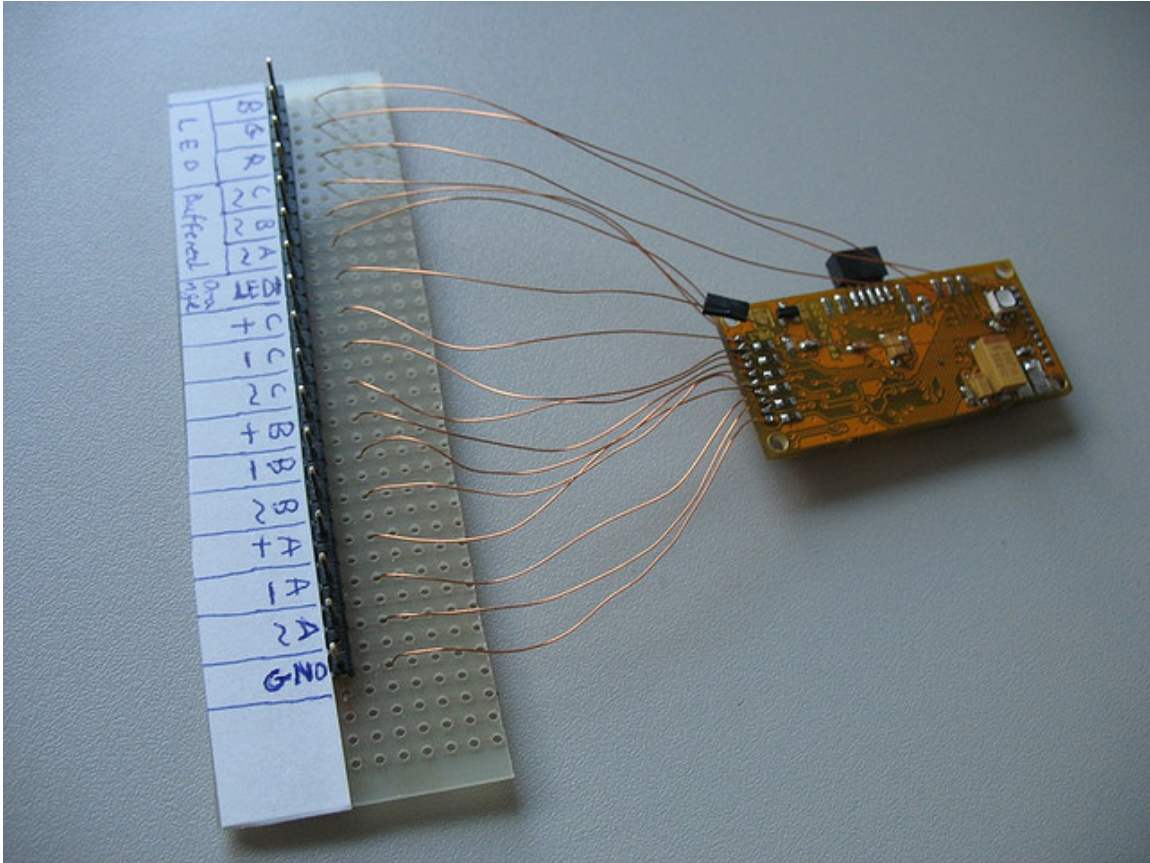
<http://raulexposito.com/>

Sobre la licencia

Este documento se publica bajo la licencia [Creative Commons 3.0 de Reconocimiento-No comercial-Sin obras derivadas](#), lo que significa que puedes realizar y distribuir copias siempre que:

- reconozcas la autoría de estos textos,
- no haya un beneficio comercial en tu distribución de las copias y
- no modifiques el contenido.

¿Qué cualidades deben tener nuestros tests?



Si estás leyendo esto seguramente estés familiarizado con conceptos tales como TDD, integración continua, tests unitarios, tests de integración, tests de estrés, test funcionales, etc. Sino, no te preocupes, daré una explicación rápida para que más adelante puedas entender fácilmente por qué es tan importante que los tests que desarrollemos cumplan con las cinco cualidades del principio FIRST.

Una parte implícita al desarrollo del software es la de la realización de pruebas que garanticen su correcto funcionamiento. Desarrollar software únicamente no basta, hay que probar que este software funciona y que además funciona correctamente. Para ello o bien realizamos pruebas de manera manual, utilizando nosotros directamente el software, o bien programamos tests automáticos que prueben que el software funciona correcta o incorrectamente por nosotros.

Hay varios tipos de tests. Si lo que queremos probar es que un sistema se comunique con otro crearemos tests de integración. Si queremos probar cuántas peticiones es capaz de soportar antes de encontrarse con problemas, estaremos creando tests de estrés. Si queremos probar el funcionamiento de una unidad de código, diremos que estamos creando un test unitario.

Hay técnicas que centran el desarrollo de nuestras aplicaciones en torno al cumplimiento de tests, de tal manera que lo primero que definiremos es qué queremos que haga el software, programaremos unos tests automáticos en base a ello, y finalmente desarrollaremos nuestro software con el objetivo de conseguir pasar esos tests. A esta manera de trabajar se la conoce con el nombre de TDD (Test Driven Development, o desarrollo dirigido por tests)

http://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas

En las páginas siguientes se presentarán las cualidades que, según Robert Martin en su libro *Clean Code*, todo buen test debería cumplir. Estas cualidades se presentan mediante el acrónimo FIRST, cuyo significado es:

Fast (rápido)

Independant (independiente)

Repeteable (repetible)

Self-validating (auto evaluable)

Timely (oportuno)

Imagen: <http://www.flickr.com/photos/69679892@N00/3905326226/>

Primera cualidad: **F**ast (rápido)



Una de las principales incomodidades con la que nos encontramos los desarrolladores a la hora de construir software aparece cuando hay que realizar pruebas de manera manual, hasta tal punto que éstas, ya sea por dejadez, por falta de disciplina, por aburrimiento o porque se acaban los plazos de entrega no se realizan con el celo profesional que les corresponde.

Realizar pruebas de manera manual tiene muchas "virtudes", entre las cuales se encuentra la pérdida de tiempo que supone realizarlas ya que, por lo general y por cada prueba:

- Hay que poner en funcionamiento el software que estamos creando al completo y, para ello, necesitaremos arrancar aplicaciones. Si son ligeras perderemos algunos segundos en ello, si son pesadas estamos

hablando de perder minutos. Ojo, sólo en arrancarlas. No hemos probado nada todavía.

- Tras arrancarlas posiblemente haya que acceder a algún sistema mediante el uso de usuario y contraseña u otro tipo de autenticación.
- Seguramente haya que navegar a través de menús y pantallas hasta llegar a algún sitio concreto donde podamos realizar alguna acción.
- Una vez allí quizá tengamos que poner algunos datos, y si es así es posible que tengamos que navegar un poco más hasta llegar al punto que queremos probar.

Tras seguir estos pasos podremos determinar si efectivamente los cambios que hemos hecho funcionan como esperamos o no. Muchas veces hay que montar todo esto para probar algo tan sencillo como que el total de una factura coincide con la suma de los conceptos que la forman.

Como veis, una comprobación tan simple como que una suma se realiza correctamente supone gastar más tiempo del que se debería. Es importante probar que la suma es correcta, pero es deseable tardar lo menos posible en realizar dicha comprobación.

En cualquier caso no sólo hablamos del tiempo que se gasta: enfrenarnos siempre a la misma pantalla de login, acceder con el mismo usuario y contraseña, poner los mismos datos, acceder a las mismas pantallas, una y otra vez, un día tras otro, semana tras semana, aburre bastante, desconcentra y es bastante poco productivo. Pero eso es otra historia.

Si conseguimos automatizar las pruebas que tenemos que realizar habremos ganado mucho terreno, pero esto no basta. Una cualidad básica y fundamental que tienen que tener los tests es la de que deben ejecutarse **rápidamente**, haciéndonos perder el menor tiempo posible.

Nos estamos ahorrando el tener que arrancar aplicaciones, el acceder a la aplicación, el navegar por las opciones, el introducir cantidades a mano, el comprobar visualmente los resultados, etc. Sin duda, estamos ganando tiempo y, sobre todo, aumentando nuestra productividad.

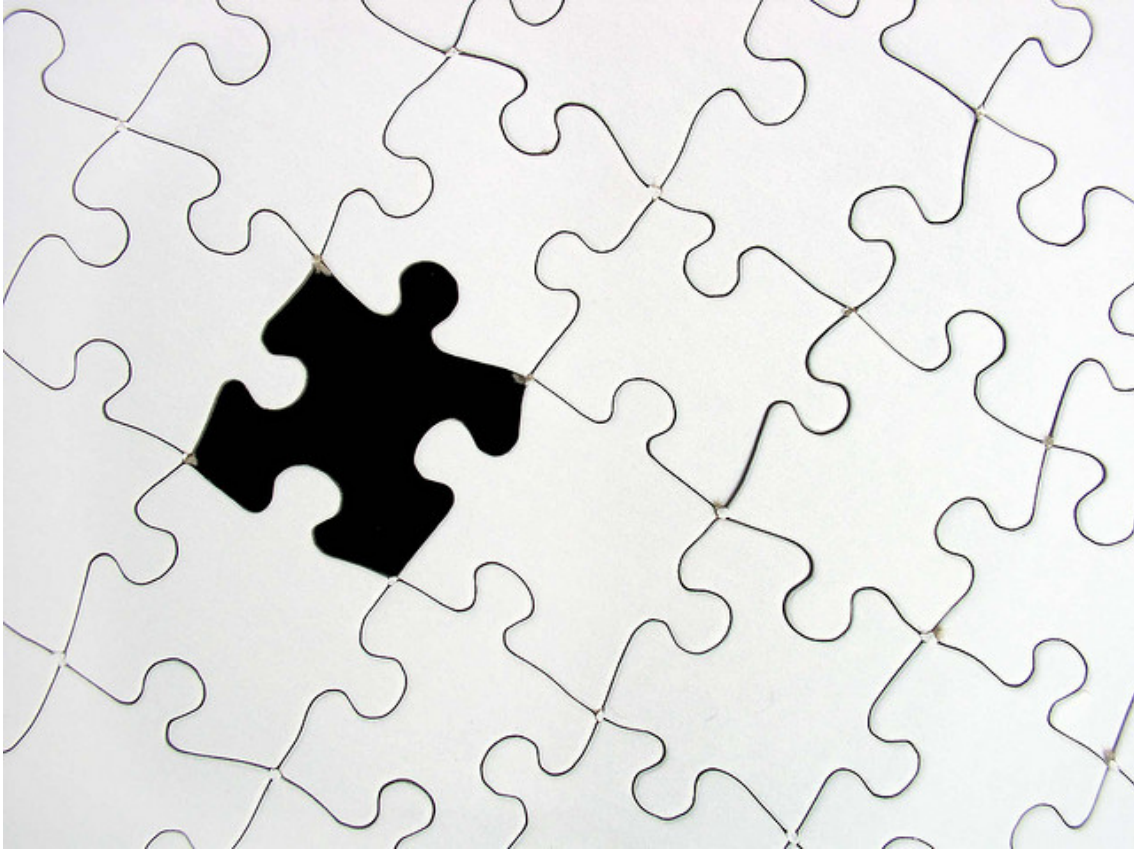
Ejecutar los tests nos ayuda a detectar los errores cuanto antes y a solucionarlos antes de que sea tarde. Si los tests tardan poco en hacer su labor no nos importará ejecutarlos las veces que haga falta, pero si tardan mucho intentaremos utilizarlos con menos frecuencia con lo que perderán efectividad, de ahí que su rapidez sea un factor importante.

Cuando estemos desarrollando una funcionalidad como, por ejemplo, sacar dinero de un cajero del banco, necesitaremos comprobar rápidamente que un cliente no puede sacar más dinero que el que tiene, o más dinero que el límite que tenga por día. Necesitamos que los tests unitarios sean rápidos, y que esas comprobaciones no tarden más que unos pocos segundos para lanzarlos cuantas más veces, mejor.

Pero claro, tipos de tests hay muchos. Aparte de los tests unitarios hay otros como tests de integración, funcionales, de estrés, etc. Estos tests pueden tardar de unos pocos a muchos minutos en ejecutarse, con lo cual quizá lo mejor sea planificarlos o delegar en otro sistema que se encargue de lanzarlos cuando sea oportuno.

Imagen: <http://www.flickr.com/photos/7843389@N02/2300190277/>

Segunda cualidad: **I**ndependent (independiente)



Para facilitarnos la tarea de detección de errores es muy importante que los tests sean **independientes** los unos de los otros. Para lograr esto deberemos evitar a toda costa que las salidas unos tests se utilicen como entradas de otros. Es más, no deberíamos preocuparnos del orden en el cual se vayan a ejecutar los tests ya que cada ejecución puede tener de hecho una ordenación distinta.

Si cada vez que ejecutamos los tests obtenemos resultados diferentes, como por ejemplo que unas veces fallen unos tests y otras veces otros, ¿cómo podremos hacer para saber qué es lo que realmente está fallando?.

Pongamos un ejemplo. Supongamos que estamos probando el comportamiento de un puesto de churros, en el cual siempre hay:

- Churros listos para ser vendidos según venga un cliente.
- Churros que están haciéndose.
- Clientes que compran churros.

Partamos de la base de que a la hora de ejecutar los tests disponemos de 100 churros para vender, tenemos 20 preparándose y hay un cliente que quiere comprar 12. En este escenario podríamos probar dos cosas diferentes:

1. Que cuando los churros que se están haciendo están listos se incrementa la cantidad de churros listos para ser vendidos. De este modo pasaría a haber $100 + 20 = 120$ churros listos, con lo que probaremos que tras hacerse los churros tengamos 120 disponibles.
2. Que cuando los clientes compran churros se reduce la cantidad de churros listos para ser vendidos. De este modo pasaría a haber $100 - 12 = 88$ churros listos, con lo que probaremos que tras la compra del cliente tengamos 88 disponibles.

Si no diseñamos los tests para que estos sean independientes ocurrirá lo siguiente:

1. Una primera ejecución de los tests empieza con 100 churros.
 - a. Se preparan 20, se prueba que haya $100+20=120$ churros y la prueba dirá que el resultado está bien. Nos quedamos con 120 churros en el puesto
 - b. Se venden 12, habrá $120-12=108$ churros y la prueba dirá que hay un error puesto que esperaba encontrar 88.
2. Una segunda ejecución de los tests empieza con 100 churros.
 - a. Se venden 12, se prueba que haya $100-12=88$ y la prueba dirá que el resultado está bien. Nos quedamos con 88 churros en el puesto.
 - b. Se preparan 20, habrá $88+20=108$ y la prueba dirá que hay un error, puesto que esperaba encontrar 120.

Si os fijáis estamos creando una dependencia artificial entre las pruebas, ya que ambas utilizan una cantidad común de churros para hacer sus cálculos. El problema viene dado porque ambas pruebas no son independientes entre sí: si una añade o quita churros está perjudicando a la otra.

La solución pasaría por, o bien dejar de nuevo la cantidad de churros listos a 100 al terminar la ejecución, o bien no utilizar esa cantidad común para hacer los cálculos y crear test independientes.

En cada una de las dos ejecuciones hemos visto que había tests que fallaban, y lo peor es que cada vez ha fallado algo distinto. En este caso todo ha sido muy sencillo porque sólo hay dos tests y el problema es fácil de detectar, pero podríamos haber tenido una batería con cientos de tests y encontrarnos con un auténtico problema.

Imagen: <http://www.flickr.com/photos/86399392@N00/109403306/>

Tercera cualidad: **R**epeteable (repetible)



¿A quién no le ha pasado alguna vez que ha visto que algo no funciona en su ordenador y sí en el de otra persona?. Cuando desarrollamos software, ¿cuántas veces habréis podido ver que una característica funciona única y exclusivamente en el ordenador del desarrollador que la ha creado?

Esto es un problema mayor del que a priori pudiera parecer. Si todo tu desarrollo funciona únicamente en un ordenador te estás poniendo la zancadilla tú mismo al limitar el desarrollo de tu propio software. Si ese ordenador se estropease o algo cambiase, ¿qué harías?

Con la ejecución de los tests de prueba ocurre algo parecido. No podemos permitir que las pruebas funcionen en unos ordenadores si y en otros no por una razón muy sencilla: todos los miembros del equipo de desarrollo deben poder probar que cualquier cambio que hayan hecho no haya afectado al software que están construyendo entre todos.

De este modo, si los tests pasan para un desarrollador deben pasar para todos, y al revés, si algún test falla, por pequeño que sea, debe fallarle a todos los desarrolladores por igual. Los resultados deben ser **repetibles** en distintos ordenadores.

Ahora bien, ¿qué puede propiciar el que haya esta disparidad?. Muchas veces son cosas sencillas, como por ejemplo:

- Que las aplicaciones que se estén probando en ordenadores diferentes estén configuradas de maneras diferentes.
- Si la aplicación usa, por ejemplo, una base de datos, que las bases de datos instaladas en los ordenadores de desarrollo tengan versiones ligeramente diferentes.
- Que los sistemas operativos sean distintos, de tal modo que se en unos sistemas se distinguen mayúsculas de minúsculas y en otro no, etc.

Una solución a este problema y a otros es contar con un servidor de integración continua que actúe como referente. Dicho servidor deberá estar instalado en un ordenador que no estén utilizando los desarrolladores para programar.

Cada vez que un desarrollador termina una funcionalidad y la deja a disposición de todos, el servidor de integración continua generará el software al completo y lanzará los tests sobre él. Los resultados de ese servidor deben ser la referencia para todos: si los tests pasan allí deberían pasar en todos los ordenadores de los desarrolladores, y si falla algún test, ese mismo test debería fallar en todos los ordenadores de los desarrolladores. En caso de no ser así, es importante determinar la causa que provoca esto y solucionarlo cuanto antes.

Imagen: <http://www.flickr.com/photos/17537227@N00/284788704/>

Cuarta cualidad: Self-validating (auto evaluable)



Sin duda, una de las mayores ventajas que tienen los test automáticos es que trabajan por nosotros. Nos ahorran hacer tareas repetitivas, tiempo, esfuerzo, y gracias a ellos es muy sencillo comprobar que al cambiar algo en la actualidad no estamos estropeando algo que funcionaba en el pasado.

Pero, aparte de todo esto, ¿podrían hacer algo más por nosotros?. Sí, podrían ayudarnos a saber si hemos obtenido los resultados que esperábamos o no siendo **auto evaluables**.

A nosotros lo que nos interesa es poder lanzar los tests y que ellos mismos sean capaces de determinar si todo ha ido bien o si, por el contrario, algo ha ido mal, de tal modo que nosotros con un simple vistazo podamos saber cuál ha sido el resultado de la ejecución de los tests.

Para conseguir esta meta es necesario que los desarrolladores no tengamos que ver la traza que ha seguido el test para ejecutarse ni saber de forma pormenorizada qué ha hecho para alcanzar el objetivo final ni cuál es, a no ser que nos encontremos problemas, en cuyo caso siempre será interesante saber qué ha fallado y por qué.

Si nuestro test, por ejemplo, calcula la inversa de matrices cuadradas, ¿qué será mejor?

1. Que mientras se ejecute el test vaya registrando qué va haciendo paso a paso, de tal modo que al terminar la ejecución comprobemos nosotros, y no el test, si el cálculo ha sido correcto o no.
2. Ejecutar el test, el cual conoce de antemano cuál es la inversa de la matriz, y que sea él quien compare el resultado obtenido con el correcto y determine si el test ha funcionado correctamente o no.

El segundo caso es mejor, ¿verdad?. Nos ahorra tiempo y nos facilita la obtención de resultados. Es por ello que deberemos conseguir que todos nuestros tests tengan la capacidad de autoevaluarse.

Imagen: <http://www.flickr.com/photos/92486261@N00/236353428/>

Quinta cualidad: **I**timely (oportuno)



En el mundo del desarrollo de pruebas de software una pregunta mil veces formulada es la de “¿Cuándo debo definir los tests?”, cuya respuesta es “Antes de ponerte a implementar la funcionalidad que necesites”.

El motivo es el muy sencillo: es más fácil hacer tests para un código que todavía no está escrito que para uno que ya ha sido creado, del mismo modo que es más fácil hacer crecer recto un árbol que todavía no ha brotado con una guía que enderezar uno que tiene varios metros de altura.

Otra de las ventajas de crear test en el momento **oportuno** es que si programas con la mentalidad de que tienes que hacer tests que probarán la funcionalidad harás código que sea fácilmente testeable.

Otro punto que juega a favor de definir los tests con antelación es la de que te permitirán saber cuándo has terminado la implementación de la funcionalidad, que será cuando tu código pase todos los tests. Un efecto colateral es que se evita desviar el desarrollo a la implementación de funcionalidades que no es necesario implementar, pudiendo centrarnos únicamente en solucionar el problema concreto que queremos resolver.

Imagen: <http://www.flickr.com/photos/21046489@N06/3387189144/>