9 de febrero de 2017
Bogotá D.C.

William Sierra william.sierra@exsis.com.co
Carlos Cardona carlos.cardona@exsis.com.co
Ervid Molina ervid.molina@exsis.com.co

The Clean Coder Capítulo 5: Test Drive Development

El desarrollo impulsado por pruebas hizo su incursión en el mercado años antes de finalizar el siglo XX. Entró junto a la programación extrema y hoy en día está contenido en SCRUM y las demás metodologías ágiles. Incluso equipos que no usan metodologías ágiles también usan TDD.

¿Cómo se puede considerar profesional si no se sabe cómo funciona el código? ¿Cómo se puede saber que todo el código funciona si no se hace una prueba para cada cambio? ¿Cómo se pueden hacer pruebas con cada cambio si no se tienen pruebas unitarias? ¿Cómo se pueden tener pruebas unitarias que cubran todo sin la práctica de TDD?

La TDD se entiende como una metodología de software que involucra dos prácticas: *Test First Development* (Escribir primero las pruebas), y *Refactoring* (Refactorización). Lo primero es escribir las pruebas y verificar que fallen. Después se implementa el código que hace que la prueba pase satisfactoriamente y por último se refactoriza el código escrito.

A continuación tenemos las leyes de la TDD, enunciadas como:

1. Solo puede escribir el código de producción (para implementar una funcionalidad) si ya ha escrito su respectivo código de prueba.
2. Solo puede escribir el código de prueba mínimo necesario que haga que el código de producción falle.
3. Solo puede escribir el código de producción necesario para hacer que éste pase su código de prueba

Se debe añadir líneas al código de prueba y otras líneas al de código de producción, ya que ambos procesos se deben hacer simultáneamente. Las pruebas se ajustan al código de producción como un anticuerpo se adapta a un antígeno.

Ahora, analizamos los beneficios que nos ofrece el Desarrollo Impulsado por Pruebas.

Primero vamos a encontrar la certeza. En esta industria no se puede estar “casi seguro” sobre un hecho (proyecto o módulo). Un profesional asume la TDD como una disciplina profesional, donde

tiene docenas de pruebas al día y miles al año, las cuales estarán disponibles para funcionar con cualquier cambio que se le aplique al código.

En segunda instancia encontramos la tasa de inyección de defectos. De acuerdo a la cantidad de cambios realizados, no se deberían incrementar la cantidad de errores que un código arroja. Al contrario, una prueba óptima es la que me indica que los cambios que se realizaron son pertinentes y nos disminuye la cantidad de errores presentados. Un profesional no hace caso omiso de estos datos.

Después encontramos el valor. ¿Por qué no se corrige un código defectuoso cuando se detecta? La primera reacción es decir que ese código debe ser limpiado, la segunda reacción es no tocarlo, porque si se toca se corre el riesgo de romperlo, y si se rompe se convierte en propio. ¿Pero qué pasa si se quiere estar seguro de no romper nada? Este es uno de los beneficios de la TDD. Cuando se tiene un conjunto de pruebas en las que se confía, entonces se pierde el temor a hacer cambios. Cuando se percibe un mal código, simplemente lo corrige en el acto. El código se puede ver o comparar como arcilla para un escultor, que se puede esculpir en estructuras simples y agradables día a día.

Seguidamente encontramos la documentación. Partamos del hecho de que los manuales y sus anexos son importantes para el entendimiento del software. Cuando se utilizan documentos de terceros, usualmente lo mejor es remitirse a los códigos de ejemplo, porque es allí donde se encuentra la esencia del producto. Esto consume mucho tiempo y esfuerzo por entender la visión subjetiva de su autor; pero las pruebas unitarias que se escriben siguiendo las tres leyes se entienden como documentación. Ellas describen un nivel granular del diseño del sistema. Son inequívocas, exactas, escritas en un lenguaje que su revisor entiende. Es la mejor documentación que puede existir.

Finalmente encontramos su diseño. Su complejidad radica en que se deben aislar las pruebas y es aún más difícil cuando se sabe que tiene funciones anidadas. Debe existir una manera de desacoplar las funciones. Por tanto, se debe pensar en una estructura flexible. Si las pruebas son creadas en una fase posterior al desarrollo, es muy difícil desacoplar las funciones para poderlas evaluar individualmente.

Concluimos que el desarrollo impulsado por pruebas es una disciplina que reduce defectos, mejora la seguridad, la documentación y el diseño. Se podría considerar poco profesional no usarla. Pero cuidado, no la confundamos como un axioma, ya que las tres leyes no aseguran el éxito ni del objetivo ni de las pruebas.