Estudio de Test-Driven Development en el proceso de desarrollo de Software.

Ing. Pablo Andrés Vaca, Ing. Calixto Maldonado, Ing. Claudia Inchaurrondo, Ing. Juan Peretti, Ing. María Soledad Romero, Ing. Matías Bueno.
UTN, Facultad Regional Córdoba, Depto. de Ing. en Sistemas de Información.
pvaca@sistemas.frc.edu.ar, cmaldonado@sistemas.frc.edu.ar, cinchaurrondo@sistemas.frc.utn.edu.ar, romeroma.soledad@gmail.com, peretti.juan@gmail.com, matiasbueno@gmail.com.

Resumen
Este trabajo presenta el proyecto sobre estudio de la utilización del desarrollo de software conocida como Test-Driven Development (TDD) en proyectos de software, exponiendo que TDD no es una metodología de pruebas. Se explica también las dificultades que se encuentran en la implementación de TDD dentro de los equipos de software y se desprende del mismo que es una metodología muy utilizada en proyectos con ciclos de vida ágiles. Se encontraron principalmente dificultades en cuanto a incrementos de costos y tiempos de desarrollos en algunos estudios, aunque los resultados que los mismos entregan no son concluyentes.

Palabras Clave
Agile, Test-Driven Development, Pruebas Unitarias, Software, Desarrollos Ágiles, Testing, Automatización de pruebas, Automatización. Automation Tests, Unit Tests, BDD, TDD, Scrum.

Contexto

El presente trabajo es el resultado de un proyecto de investigación cuyo objetivo es estudiar y enmarcar este grupo de metodologías en la industria, mostrando tanto las particularidades propias, como los tipos de proyectos en los cuales se aplica, las ventajas y dificultades que pueden surgir de su adopción. El proyecto está acreditado en la Secretaria de Ciencia y Técnica de la Universidad desde Mayo de 2013.

Introducción

El desarrollo de software plantea muchos desafíos a los equipos de trabajo, estos retos van desde entender lo que el mercado y los clientes necesitan, hasta brindar soluciones integrales que cubran todos los aspectos de una organización manteniendo los costos controlados y brindando mejoras a las organizaciones.

Uno de los desafíos más grandes es lograr garantizar la calidad de los productos de software y mejorar la productividad a lo largo del ciclo de vida del mismo, desde el comienzo del desarrollo hasta las etapas de mantenimiento. Este es un reto importante ya que actualmente los requerimientos son mucho más volátiles que en el pasado, lo cual convierte a los cambios en el software en un momento en el cual potencialmente se introducen muchos errores, ocasionando que funcionalidades ya implementadas empiecen a fallar, aparecen muchas fallas o errores, llamados “Bugs” en la etapa de mantenimiento.

En respuesta a esto se introducen más equipos de pruebas para realizar todo tipo de ellas, desde pruebas exploratorias hasta pruebas de regresión extensivas, las cuales toman mucho tiempo.

Test Driven Development (TDD) es una práctica iterativa de diseño de software orientado a objetos, que fue presentada por Kent Beck y Ward Cunningham como parte de XP [4], la misma fue definida como el núcleo de XP.

Se divide en 3 sub-prácticas [5]:

- Test-First: las pruebas se escriben antes de escribir el propio código, y las

mismas son escritas por los propios desarrolladores, esto busca que los mismos logren un entendimiento de lo que deben desarrollar mediante la construcción del código que lo va a probar.

- Automatización: las pruebas deben ser escritas en código, y esto permite que se ejecuten automáticamente las veces que sea necesario, y el solo hecho de ejecutar las pruebas[1] debe mostrar si la ejecución fue correcta o no.
- Refactorizar[2] el código: permite mantener la calidad de la arquitectura, se cambia el diseño sin cambiar la funcionalidad, manteniendo las pruebas como reaseguro.

En la figura 1 se puede apreciar el proceso propuesto para implementar TDD como metodología de desarrollo de software[3].

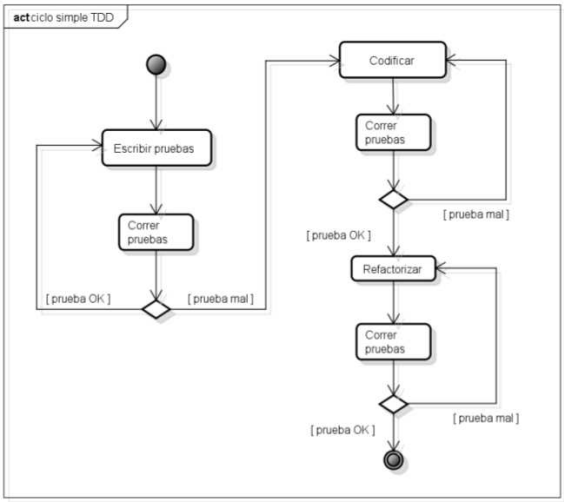

Figura 1 [5].

De estas sub-prácticas se desprenden algunas ventajas:

- Podemos desvincular el factor humano en las pruebas, ya que al automatizar las mismas cada una de ellas determina si se ejecutó con éxito o no.
- La automatización de las pruebas, implica una disminución del costo de las mismas, no sólo porque se ejecutan rápidamente, sino además porque las podemos programar para que se ejecuten sin intervención humana.
- Al escribir las pruebas en primer lugar, el autor del código no se encuentra condicionado por el código a probar, por otra parte permite además especificar el comportamiento esperado sin restringirse a una implementación en particular. También permite a los programadores aprender más de la funcionalidad a implementar, las pruebas pueden tomarse como un criterio de aceptación [4]. Lech Madeyski y Lukasz Szala en su trabajo [14] empíricamente encontraron mejoras además en la productividad medida como una combinación de líneas de código escritas, user stories completadas y pruebas de aceptación correctas al utilizar TDD.
- La refactorización, permite mantener el diseño del código limpio, mejorando su mantenibilidad. Otro aspecto que posibilita la refactorización es la eliminación de la duplicación de código, lo cual trae como consecuencia reducir la dependencia en el código [6].

En palabras de Kent Beck "Nunca escribas una nueva funcionalidad sin escribir primero una prueba que falle" [6]. Por otra parte Dave Chaplin expresa "Si no se puede escribir una prueba para el nuevo código, entonces no se debería estar pensando en incorporar el nuevo código" [7]. Estos dos axiomas de TDD implican que ninguna nueva funcionalidad se debe escribir por adelantado, sino que es necesario previamente contar con las pruebas que permitan verificar que es correcta. Y como se deben escribir las pruebas de a una a la

1-Se refiere a todas las pruebas generadas hasta el momento. Así, el conjunto de pruebas crece en forma incremental y sirve como pruebas de regresión ante agregados futuros.

2-Hemos usado el término "refactorización" como traducción del término inglés "refactoring".

3-En los tres casos en que el diagrama dice "correr pruebas" se refiere a todas las pruebas generadas hasta el momento.

vez, tenemos de esta manera un proceso de desarrollo incremental extremo. Es por esto último que Kent Beck definió a TDD como el corazón de la metodología XP.

Actualmente se observa un aumento de aplicación, en el desarrollo de software, del enfoque guiado por las pruebas, en algunos casos utilizando solamente pruebas unitarias, en otros, se lo aplica avanzando hasta utilizar pruebas de integración. Si se considera el libro publicado por Kent Beck [4] o se analiza el trabajo de Maria Siniaalto "Test-Driven Development: Empirical Body of Evidence [10]", se encuentra que hace poco mas de 10 años que se comenzó a trabajar en el desarrollo guiado por pruebas de comportamiento, estas últimas orientadas desde el punto de vista de los requerimientos funcionales o desde el punto de vista del negocio. Existen numerosas metodologías de desarrollo guiadas por pruebas tales como la estudiadaTDD, la Unit Test Driven Development (Desarrollo guiado por pruebas unitarias) o UTDD, la Story Test Driven Development (Desarrollo guiado por la historia de prueba) STDD, la Acceptance Test Driven Development o Desarrollo guiado por la prueba de aceptación ATDD y la Behaviour Driven Development o Desarrollo guiado por el Compartamiento BDD [5] [6] [13].

Fontela en su trabajo [5] encuentra algunas evidencias de que la incorporación de TDD disminuye los costos de desarrollo, pero también existen casos en los que el costo de escribir y mantener los tests o pruebas no compensa sus beneficios. Por otra parte en el mismo estudio se puede apreciar que arriba a la conclusión de que disminuyen los tiempos de desarrollo, pudiéndose asociar esto a una potencial disminución de costos. En el trabajo de Maria Siniaalto [10] no se llega a una conclusión certera, sino que encuentra contradicciones acerca de los incrementos de productividad y por consiguiente disminución de costos. En el trabajo "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [1] vemos que algunos casos estudiados reportaron efectos negativos en el tiempo de desarrollo y otros reportaron efectos positivos sobre el mismo aspecto.

Objetivos

El proyecto de investigación tiene como objetivo estudiar y enmarcar este grupo de metodologías en la industria, mostrando tanto las particularidades propias, como los tipos de proyectos en los cuales se aplica, las ventajas y dificultades que pueden surgir de su adopción.

Para ello, se realizó una revisión bibliográfica de trabajos existentes, evaluando cada uno de ellos, sus hipótesis y sus conclusiones.

Además a esta exploración bibliográfica se ha sumado los aspectos de una revisión sistematizada de la literatura existente. Una revisión sistematizada es un estudio empírico de trabajos primarios y secundarios publicados, sobre los cuales se realizó una lectura, se evaluaron sus resultados y luego se publicarán las conclusiones. Para la exploración se ha utilizado como guía un trabajo de un grupo de investigadores de Gran Bretaña [1] y otro trabajo conjunto de un grupo de investigadores Españoles y Croatas [2]. Se realizó una búsqueda de información bibliográfica utilizando buscadores en la web. En esta búsqueda se utilizaron principalmente criterios de búsqueda definidos por palabras claves como: "TDD - Limitaciones a la implementación - Factores de éxito - Casos de éxito - TDD y el desarrollo de software - Test-driven development - Introducción al Test-Driven Development – Pros and Cons of TDD – Agile – Extreme Programming, XP".

Resultados

El resultado de estas búsquedas fue una lista de documentos y blogs, sobre los cuales se realizó un trabajo de recopilación de información. El criterio empleado

consistió en realizar una lectura de cada uno de los trabajos con la finalidad de detectar aquellos que consideramos realizaban aportes significativos a nuestro trabajo. Hay que destacar que la mayoría de la documentación encontrada ha sido escrita por autores que son parte del desarrollo de esta tendencia, es decir, están involucrados en su difusión. Se ha encontrado poca documentación que haga un estudio objetivo de las ventajas y desventajas, razón por la cual a esta información hemos adicionado algunos blogs online. A éstos se los inspeccionó y apartó considerando para su clasificación los más recientes y aquellos que refieren a bibliografía conocida y que es utilizada en este trabajo. Estos blogs son considerados como notas y no como referencias en el presente trabajo.

Luego de esta selección y evaluación, se utilizaron los elementos resultantes como base y soporte para nuestro trabajo, permitiendo establecer el estado actual de las prácticas de desarrollo de software guiado por pruebas automatizadas y arribar a una conclusión sobre si las mismas aportan beneficios o, por el contrario, presentan limitaciones para su implementación en desarrollos de productos de software.

Con esta revisión se examinó el estado del arte en el que se encuentra TDD dentro de la industria y se elaboró una presentación, basado en las guías que se proveen en el trabajo "Presentación por escrito de la revisión bibliográfica" [3], para el Congreso Nacional de Ingeniería Informática y Sistemas de Información (Conaiisi) del año 2013 [15].

Conclusiones

Al comenzar este trabajo se describió de qué se trata TDD, en qué consiste, si es posible utilizarlo en proyectos de distintos tamaños.

TDD consiste en tomar una funcionalidad y en primer lugar desarrollar las pruebas, inicialmente estas fallarán y en muchos casos será necesario la implementación de mocks[4] que nos permitan simular las clases con las cuales interactúa nuestra clase y que aún no fueron desarrollados, también puede ser necesario implementar objetos del tipo stubs[5] generalmente para implementar las interfaces con bases de datos. Una vez que se dispone de las pruebas el desarrollador implementa el código para lograr que la prueba deje de fallar y luego refactoriza el código para lograr un código limpio y prolijo.

Podemos afirmar que TDD es una metodología de desarrollo de software y no de pruebas de software, esto se desprende de los trabajos y libros publicados por quien es considerado como uno de los creadores de la misma, Kent Beck [4][6] al plantear que primero se deben escribir las pruebas que inicialmente fallarán y luego se debe escribir el código que haga que dichas pruebas dejen de fallar. Esto significa además una ventaja para los desarrolladores, pues logran una mejor comprensión de las funcionalidades que están desarrollando. Kent Beck planteó TDD como parte de XP, luego se fue extendiendo a otras metodologías de gestión del desarrollo de software, logrando un muy buen acople con los métodos Agile [10]. Tal como lo expresa Carlos Fontela en la conclusión de su trabajo de especialización citando a Freeman y Pryce "TDD es una técnica que combina pruebas, especificación y diseño en una única actividad holística", y que tiene entre sus objetivos iniciales el acortamiento de los ciclos de desarrollo, algo que se ajusta como dijimos a las metodologías ágiles.

Otro aspecto que se deriva de los trabajos es expresado también por Carlos Fontela en su trabajo, TDD no reemplaza las pruebas funcionales de humanos ya que toda la

4-Mocks son objectos que nos permiten maquetar el comportamiento de objetos reales que implementan clases que aún no fueron escritas.

5-Un stub es un objeto que implementa una instancia de una clase que simula a un ente externo tal como una base de datos.

seguridad que nos da TDD viene de código escrito por humanos, el cual puede contener fallas. Algo que si nos da TDD si ha sido aplicado rigurosamente y revisado a conciencia es un nivel de calidad y seguridad al momento de incorporar cambios en las funcionalidades que ya existen.

Es necesario realizar más estudios a fin de determinar aquellas adaptaciones requeridas para TDD que permitan su adopción en cualquier etapa de un proyecto. Al mismo tiempo se pueden realizar estudios de campo para comprobar si la industria ya encontró estas adaptaciones.

Formación de Recursos Humanos

Tres integrantes del Grupo son tesistas de Posgrado en la Maestría de Ingeniería de Software en la UTN-FRC, un integrante es tesista de un Doctorado en Ingeniería de Software en la Universidad de Vigo y un estudiante es becario. Excepto el estudiante los integrantes desarrollan tareas relacionadas en empresas del medio, nacionales e internacionales, donde se espera aplicar los resultados del proyecto.

Referencias.

[1] Barbara Kitchenham (a), O. Pearl Brereton (a), David Budgen (b), Mark Turner (a), John Bailey (b), Stephen Linkman (a) - (a): Software Engineering Group, School of Computer Science and Mathematics, Keele University, Keele Village, Keele Staffs. ST5 5BG, UK. (b): Department of Computer Science, Durham University, Durham, UK - "Systematic literature reviews in software engineering – A systematicliterature review". 2009.

[2] Zlatko Stapic, Vjeran Strahonja, Facultad de Organización e Informática, Universidad de Zagreb. Eva García López, Antonio García Cabot, Luis de Marcos Ortega, Departamento de Ciencia de la Computación, Universidad de Alcalá - "Performing Systematic Literature Review in Software Engineering". 2012.

[3] Fernando Aranda Fraga - "PRESENTACIÓN POR ESCRITO DE LA REVISIÓN BIBLIOGRÁFICA" - Secretaría de Ciencia y Técnica - Universidad Adventista del Plata.

[4] Kent Beck, "Extreme Programming Explained: Embrace Change", Addison-Wesley Professional, 1999. ISBN: 978-0321278654

[5] Carlos Fontela - Trabajo Final de Especialización en Ingeniería de Software. "Estado del arte y tendencias en Test-Driven Development" - Junio de 2011

[6] Kent Beck, "Test Driven Development: By Example", Addison-Wesley Professional, 2002.

[7] Dave Chaplin, "Test First Programming", Tech Zone, 2001.

[8] Carlos Ble Jurado, "Diseño Ágil con TDD", eBook. Primera Edición. Enero 2010. Página 55

[9] Adnan Causevic, Daniel Sundmark, Sasikumar Punnekkat - Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review - Mälardalen University, School of Innovation, Design and Engineering, Västerås, Sweden. 2010. página 5

[10] Maria Siniaalto - "Test driven development: empirical body of evidence" - ITEA (Information Technology for European Advancement). 2006.

[11] Martin Fowler - Continuous Integration. 10 September 2000 y 1 Mayo de 2006. http://martinfowler.com/articles/continuousIntegration.html, como estaba en junio de 2011.

[12] Steve Freeman, Nat Pryce, "Growing Object-Oriented Software, Guided by Tests", Addison-Wesley Professional, 2010. ISBN-13: 978-0321503626.

[13] Elisabeth Hendrickson. Driving Development with Tests: ATDD and TDD. Quality Tree Software, Inc. 2008.

[14] Lech Madeyski, Lukasz Szala - The Impact of Test-Driven Development on Software Development Productivity — An Empirical Study 2007

[15] Vaca, Pablo, . et al. "Test-Driven Development - Una aproximación para entender su utilidad en el proceso de desarrollo de Software."
Conaiisi- Cordoba, Octubre de 2013 - http://conaiisi.frc.utn.edu.ar/ISSGP - (En línea) ISSN : 2347-0372