Test-Driven Development - Beneficios y Desafíos para el Desarrollo de Software.

Pablo Andrés Vaca, Calixto Maldonado, Claudia Inchaurrondo, Juan Peretti, María Soledad Romero, Matías Bueno, Marcelo Cagliolo

pvaca@sistemas.frc.edu.ar, calixtomaldonado@hotmail.com, cinchaurrondo@sistemas.frc.utn.edu.ar, peretti.juan@gmail.com, romeroma.soledad@gmail.com, matiasbueno@gmail.com, marcelocagliolo@gmail.com

Universidad Tecnológica Nacional – Facultad Regional Córdoba

Abstract. Este trabajo presenta la introducción y utilización de la práctica de desarrollo de software conocida como Test-Driven Development (TDD) en proyectos de software, expone además que TDD no es solo una metodología de pruebas, sino además una metodología de diseño de software. Está basado en un proyecto de investigación perteneciente a la Universidad Tecnológica Nacional - Facultad Regional Córdoba, con el objetivo de estudiar y enmarcar este tipo de metodologías en la industria, mostrando tanto las particularidades de la misma, como los proyectos en los cuales se aplica, y las ventajas y dificultades que pueden surgir de su adopción. Se explica en el presente trabajo los beneficios y desafíos que se encontraron durante la implementación de TDD en los equipos de software y se resaltan los beneficios relacionados con la calidad del código, de las aplicaciones, la productividad y la comunicación. En contrapartida se evidenciaron desafíos en relación a la experiencia previa de los desarrolladores, ya que TDD resulta más simple de implementar en equipos con experiencia media a alta, en la bibliografía consultada se halló que esto puede ser mitigado implementando mentorías o programación extrema.

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

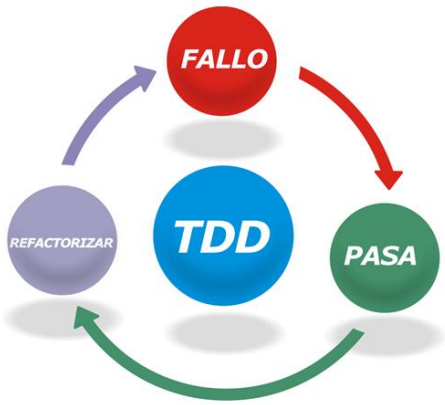
1 Introducción

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 valor a las organizaciones.

En los comienzos de la industria del software la metodología más utilizada era el desarrollo en cascada, en la cual se planteaban los requerimientos inicialmente y se avanzaba en etapas. Con la incorporación del software a todos los aspectos de la vida estos requerimientos comenzaron a cambiar a un ritmo acelerado, volviendo obsoleta la metodología que habia imperado. Acompañando esta evolución de la industria apareció "Programación Extrema"(XP) [1]. Como parte de XP [1,2], Test Driven Development (TDD) es una práctica iterativa de diseño de software orientado a objetos, que fue presentada por Kent Beck y Ward Cunningham, quienes la definieron como el núcleo de XP.

En un trabajo anterior [3], se había precisado de qué se trata TDD expresando que, entre otras cosas, se divide en 3 sub-prácticas [4] denominadas: Test-First, Automatización y Refactorizar[1]. De acuerdo a estas prácticas las pruebas inicialmente fallarán, luego el desarrollador se enfocará en escribir el código que permita la ejecución exitosa de las pruebas y por último se enfocará en eliminar código duplicado, innecesario o mejorar el mismo para cumplir con patrones de codificación establecidos que facilitan la lectura del mismo y le dan claridad [2], las pruebas son el reaseguro que garantiza que esta refactorización no alterará la funcionalidad. En este sentido Martin Fowler[5] plantea varias recetas para refactorizar con éxito.

Fig. 1. Ciclo de Vida de TDD [6].



[1]Hemos usado el término "refactorización" como traducción del término inglés "refactoring".

En la figura 1 se puede apreciar el proceso propuesto para implementar TDD como metodología de desarrollo de software[2].

Actualmente, en el desarrollo del software, se observa que la aplicación del enfoque guiado por las pruebas ha ido en aumento, en algunos casos utilizando solamente pruebas unitarias, en otros, se lo aplica avanzando hasta utilizar pruebas de integración. En varios trabajos [1,7], se observó 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 además de TDD tales como UTDD[3], STDD[4], ATDD[5], BDD[6] [4,8,9,2].

Muchos de los que proponen el uso de TDD[2,10,11,12,13] han advertido que su implementación debería dar como resultado el logro de mejoras en la calidad del código, la calidad de las aplicaciones, la calidad de las pruebas, si se lo compara con los resultados obtenidos por Test Later (TL). En adición a ello plantean mejoras en la productividad, en el entendimiento temprano por parte de los desarrolladores de los requerimientos y en simultáneo los desarrolladores lograrán satisfacción y confianza en su trabajo [11].

Fontela, en su trabajo [4] 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 [7] no se llega a una conclusión certera, sino que encuentran contradicciones acerca de los incrementos de productividad y por consiguiente una disminución de costos. En el trabajo "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [14] vemos que algunos casos estudiados reportaron efectos negativos en el tiempo de desarrollo y otros reportaron efectos positivos sobre el mismo aspecto.

En este contexto en el presente trabajo se expondrán los beneficios y retos que se desprenden de estudios y de la aplicación práctica de TDD en proyectos de desarrollo de software.

[2]Fallo: Escribir una prueba que falle. Pasa: Escribir código que pase la prueba. Refactorizar: Reorganizar o limpiar el código.
[3]Unit Test Driven Development.
[4] Story Test Driven Development.
[5] Acceptance Test Driven Development.
[6] Behavior Driven Development.

2 Elementos del Trabajo y Metodología

Para llevar a cabo esta propuesta se realizó una revisión sistematizada de la literatura existente. Una revisión sistematizada es un estudio empírico de trabajos primarios y secundarios publicados, para ello, hemos utilizado como soporte de guía un trabajo de un grupo de investigadores de Gran Bretaña [14] y otro trabajo conjunto de un grupo de investigadores Españoles y Croatas [15]. Además, nos hemos basado en las guías que se proveen en el trabajo "PRESENTACIÓN POR ESCRITO DE LA REVISIÓN BIBLIOGRÁFICA" [16].

La búsqueda de información bibliográfica también incluyó la utilización de buscadores en la web. En esta tarea 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". 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 textos 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 esta presentación.

Luego de esta revisión, 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 los beneficios y desventajas que se presentan en la implementación de TDD en desarrollos de productos de software.

3 Resultados

Los detractores de TDD argumentan que no se puede utilizar esta metodología en grandes proyectos, este punto se puede contrastar con lo que dice Carlos Ble Jurado en su libro [2]: "¿Y TDD sirve para proyectos grandes? Un proyecto grande no es sino la agrupación de pequeños sub- proyectos y es ahora cuando toca aplicar aquello de **divide y vencerás** [2]. El tamaño del proyecto no guarda relación con la aplicabilidad de TDD. La clave está en saber dividir, en saber priorizar. De ahí la ayuda de

Scrum[7] o Agile[8] para gestionar adecuadamente el backlog[9] del producto. Por eso tanta gente combina XP y Agile. Todavía no he encontrado ningún proyecto en el que se desaconseje aplicar TDD". Esto es correcto debido a que los grandes proyectos se componen de módulos y en todos los casos se llega a porciones de código en los cuales es posible implementar TDD. En los grandes proyectos será necesario contar con un equipo de mayor tamaño para implementar TDD.

Investigadores de la Escuela de Computación y Ciencias Matemáticas de la Universidad de Auckland en Nueva Zelanda realizaron un trabajo [13] basados en entrevistas a diferentes empresas que previamente a implementar TDD desarrollaban software con metodología en cascada. En este trabajo se propuso determinar qué beneficios y desafíos se perseguían como resultado de la implementación de TDD.

3.1 Como aspectos positivos se encontraron los siguientes:

Código de calidad: todos los entrevistados encontraron que el hecho de disponer de las pruebas previamente predispone a los desarrolladores a escribir código significativo de forma simple y limpia en comparación con el uso de la técnica tradicional de escribir las pruebas después de escribir el código. La percepción es que TDD hace que los desarrolladores adquieran hábitos que hagan que cada vez escriban mejor código.

Por otra parte TDD también es percibido como custodio de la calidad ante las presiones que se generan para entregar código terminado.

Investigadores suecos [17] encontraron que la calidad del código mejora en los casos que se escriben primero las pruebas.

Aplicaciones de Calidad: el aumento de la confiabilidad del código se ve como un beneficio que se deriva de la implementación de TDD. Esto es resultado de que los desarrolladores deben dedicar tiempo a escribir los casos límites que prueben el código, no solamente los casos más comunes. Esto conlleva el aumento de cobertura y densidad de las pruebas, algo que no es captado cuando las pruebas se implementan a posteriori. Esta mayor cobertura es vista como resultado de aplicar TDD, algo que no es percibido en TL ya que muchas veces ni siquiera se dispone del tiempo para escribir las pruebas.

En estudios realizados en Suecia [17] no se encontraron diferencias significativas en la calidad del producto entre TDD y TL, solamente encontraron algunas diferencias en la cantidad de test-cases positivos, concluyendo que esto se puede deber a que TDD es una metodología de diseño y no de pruebas.

[7] Metodología ágil para el desarrollo de software.
[8] Metodología ágil para el desarrollo de software.
[9] Listado "User Stories" a ser desarrollados durante un proyecto que utilice Agile o Scrum.

Productividad: este estudio encontró además que todos los entrevistados coincidieron en que se incrementó la productividad en relación con sus experiencias previas. Ven esto como resultado de que escribir las pruebas previamente e ir incrementando cada funcionalidad de a una por vez generan que haya menos retrabajo, comparando con TL. David Janzen [18] halló que utilizando TDD la productividad se incrementa si se compara con no escribir los test o escribir los test luego del código. De todas maneras el mismo trabajo no denota grandes diferencias en cuanto a la cantidad de código que se genera y la cobertura de las pruebas en los casos en que las mismas son escritas.

También se encontraron desafíos interesantes en profundizar y analizar, el principal es que los desarrolladores advierten los beneficios después de un tiempo de estar trabajando con TDD. En algunos casos algunos desarrolladores no hallaban natural esta forma de trabajo y era necesario recordar los beneficios con más frecuencia. Otro desafío que se encontró es que los beneficios están fuertemente vinculados a la capacidad que tengan los desarrolladores de escribir pruebas de calidad. Si bien visto como un desafío, Roberto Latorre concluye en su trabajo "Effects of developer experience on learning andapplying Unit Test-Driven Development" [19] que si los desarrolladores tienen los conocimientos y habilidades necesarias, aunque no tengan experiencia previa en la metodología, pueden aprender rápidamente Unit Test-Driven Development (UTDD), en esta publicación se deduce que la gran mayoría de los participantes consideraron que fue fácil de aprender la metodología. De todas maneras plantea que la percepción general es que UTDD es difícil de aprender para desarrolladores menos experimentados, lo cual puede ocasionar que en ambientes industriales y, bajo presión puede provocar que se deje de lado la metodología. Latorre plantea que esto se puede minimizar con la implementación de mentorías relacionadas con UTDD o programación en pares[10] armando equipos mixtos en relación a la experiencia de cada uno de los programadores.

Un desafío no menor, es el relacionado con cambiar la visión que la alta dirección ve como no productivo el tiempo que los desarrolladores usan en escribir pruebas.

En "Test-Driven Database Development: Unlocking Agility" [20] Max Guernsey, en su tarea de coach de equipos de desarrollo resalta que es tan necesario trabajar con la metodología como con las habilidades del equipo hasta que el mismo alcanza la madurez.

Calidad en Comunicación: Fang Huang [21] resalta en su libro "Test Driven Development for Multithreaded Embedded Systems" a modo de conclusión que TDD contribuye a disminuir los defectos y mejora la comunicación entre el cliente y los

[10] Latorre [14]. En sus conclusiones, en la página 14, plantea establecer mentorías o programación extrema.

desarrolladores, tal como ha podido comprobar en el caso la aplicación de TDD en sistemas embebidos.

4 Conclusión

En este trabajo describimos los aspectos positivos que derivan de la implementación de TDD.

Se han podido observar beneficios en la productividad de los equipos de trabajos y en la calidad del código, al escribir las pruebas primero, también se aprecian mejoras en cuanto a la calidad de los productos, con el solo hecho de escribir las pruebas, ya sean antes o después del código. Si se toman en cuenta todos los beneficios se arriba a la conclusión que es deseable escribir las pruebas antes que el código. De todas maneras, es un punto común en todos los trabajos analizados, que la implementación de TDD requiere experiencia en desarrollo, compromiso, disciplina y constancia para lograr los mejores resultados es un punto común en todos los trabajos analizados.

Algo que también se desprende de los trabajos y bibliografía es que resta mucho por hacer con respecto a la forma de lograr aceptación por parte de los equipos de desarrollo como así también en la forma de conformar estos equipos. Se estableció que en varios casos los equipos no veían a TDD como algo que pudiera aportar mejoras a su trabajo. Este es un punto en el que hay que profundizar, en cómo lograr equipos balanceados que aprovechen al máximo esta metodología y consigan resultados de calidad en su trabajo.

Referencias:

1. Kent Beck, “Extreme Programming Explained: Embrace Change”, Addison-Wesley Professional, 1999. ISBN: 978-0321278654.
2. Carlos Ble Jurado, “Diseño Ágil con TDD”, eBook. Primera Edición. Enero 2010.
3. Ing. Pablo Andrés Vaca, Ing. Calixto Maldonado, Ing. Claudia Inchaurrondo, Ing. Juan Peretti, Ing. María Soledad Romero, Ing. Matías Bueno - “Test-Driven Development - Una aproximación para entender su utilidad en el proceso de desarrollo de Software”.
4. Carlos Fontela - Trabajo Final de Especialización en Ingeniería de Software. "Estado del arte y tendencias en Test-Driven Development" - Junio de 2011.
5. Martin Fowler. “Refactoring: Improving the Design of Existing Code”. Addison-Wesley, 2012 – ISBN: 9780133065268.
6. Dave Chaplin, “Test First Programming”, Tech Zone, 2001.
7. Maria Siniaalto - "Test driven development: empirical body of evidence" - ITEA (Information mechnology for European Advancement). 2006.
8. Kent Beck, “Test Driven Development: By Example”, Addison-Wesley Professional, 2002. ISBN: 9780321146533
9. Elisabeth Hendrickson. “Driving Development with Tests: ATDD and TDD” - Quality Tree Software, Inc. 2008.

10. "Comparative Study of Test Driven Development with Traditional Techniques" International Journal of Soft Computing and Engineering (IJSCE) - March 2013. Shaweta Kumar, Sanjeev Bansal
Disponible: http://www.ijsce.org/attachments/File/v3i1/A1351033113.pdf - Fecha de acceso: Febrero 2014.
11. K. Beck, "Aim, fire [test-first coding]," Software, IEEE, vol. 18, pp. 87-89, 2001.
12. Test-Driven Development: A Practical Guide, 2003.
13. Buchan, J., Li, L., & MacDonell, S.G. "Causal Factors, Benefits and Challenges of Test-Driven Development: Practitioner Perceptions, in Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC 2011)". Hochiminh City, Vietnam, IEEE Computer Society Press, pp.405-413. - doi: 10.1109/APSEC.2011.44. - 2011
14. 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.
15. Zlatko Stapic, Vjeran Strahonja -"Performing Systematic Literature Review in Software Engineering" 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 -. 2012.
16. Fernando Aranda Fraga - "PRESENTACIÓN POR ESCRITO DE LA REVISIÓN BIBLIOGRÁFICA" - Secretaría de Ciencia y Técnica - Universidad Adventista del Plata.
17. Adnan Causevic, Sasikumar Punnekkat y Daniel Sundmark – "Quality of Testing in Test-driven Development" – Universidad de Malardalen – Suecia. Disponible: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6511824&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6511824- Fecha de acceso: Marzo 2014
18. David Janzen – "On the influence of Test-Driven Development on Software Design – Conference and Software Engineering Education and Training" PhD Candidate in Computer Science – Universidad de Kansas –.- 2006
19. Roberto Latorre – "Effects of developer experience on learning and applying Unit Test-Driven Development". Disponible:
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6690135 – Fecha de acceso Febrero 2014
20. Max Guernsey "Test-Driven Database Development: Unlocking Agility" - Addison-Wesley, 2013 ISBN 9780132776462
21. Fang Huang "Test Driven Development for Multithreaded Embedded Systems" University of Calgary (Canada), 2008 ISBN 9780494382721.