Test-Driven Development - Una aproximación para entender su utilidad 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.

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 una metodología de pruebas. Está basado en un proyecto de investigación perteneciente a la Universidad Tecnológica Nacional - Facultad Regisonal 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, las ventajas y dificultades que pueden surgir de su adopción. Se explica también en el presente trabajo 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.

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 enorme 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. Esto ocasiona que en muchos casos no se puedan utilizar metodologías ágiles, tales como eXtreme Programming (XP), ya que éstas se basan en períodos de desarrollo cortos (dos a cuatro semanas) y al final de los mismos se entrega un producto de software funcional. Esta práctica es posible sólo si se logra incorporar herramientas que agilicen la ejecución de grandes cantidades de casos de pruebas en cuestión de minutos, permitiendo a los miembros del equipo efectuar un desarrollo incremental y continuo del software, manteniendo la integridad del producto.

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].

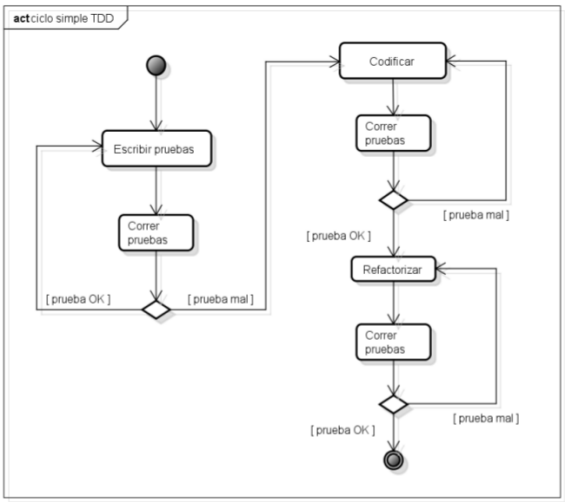

Figura 1 [5].

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 [6] en su trabajo empíricamente encontraron además mejoras 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 [7].

Son palabras de Kent Beck “Nunca escribas una nueva funcionalidad sin escribir primero una prueba que falle” [7]. 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” [8]. 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

---

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.

permitan verificar que es correcta. Y como se deben escribir las pruebas de a una a la 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 consideramos el libro publicado por Kent Beck [4] o analizamos el trabajo de Maria Siniaalto "Test-Driven Development: Empirical Body of Evidence [9]", vemos 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 TDD, UTDD[4], STDD[5], ATDD[6], BDD[7] [5], [7], [10].
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 [9] 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.
En el presente trabajo buscaremos mostrar TDD, explicar qué es, su finalidad, para qué sirve, para qué no sirve y cómo se puede utilizar para lograr mejores resultados.
También procuraremos exponer detalladamente el estado actual de TDD y determinar si es útil para todo tipo de proyecto o sólo aplicable en ciertos casos.
Como se puede apreciar se ha descripto una metodología de diseño, TDD, y no de pruebas.

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 [1] y otro trabajo conjunto de un grupo de investigadores Españoles y Croatas [2]. Además, nos hemos basado en las guías que se proveen en el trabajo "PRESENTACIÓN POR ESCRITO DE LA REVISIÓN BIBLIOGRÁFICA" [3].

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

4-Unit Test Driven Development.
5-Story Test Driven Development.
6-Acceptance Test Driven Development.
7-Behavior Driven Development.

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 si las mismas aportan beneficios o, por el contrario, presentan limitaciones para su implementación en desarrollos de productos de software.

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 [11]: "¿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**[8]. 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[9] o Agile[10] para gestionar adecuadamente el backlog[11] 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.

Por otra parte se encontró que existen varias clases de tests que se pueden implementar.

Tipos de Tests:

Test de aceptación: es un test que permite comprobar que se está cumpliendo con un requerimiento del negocio. Son pruebas escritas en lenguaje del cliente pero que puede ser ejecutado con la máquina. Esto nos permite probar que el software que estamos desarrollando cumple con las expectativas del cliente y de los usuarios[12].

Test funcionales: si bien, siendo estrictos, todos los tests son funcionales ya que prueban alguna funcionalidad, esta expresión es utilizada para determinar a aquellas pruebas que agrupan a varios tests de aceptación y prueban alguna funcionalidad del negocio propiamente dicha[13].

Test de sistema: integra varias partes del sistema, incluso puede probar toda la aplicación o varias funcionalidades juntas. Estos tests se comportan de manera similar y buscan emular el comportamiento de los usuarios del sistema[14].

Test unitarios: son los tests ineludibles, son los necesarios y los más importantes para los desarrolladores, todo test unitario debe ser rápido, atómico, inocuo e

8-Página 55 de su libro "Diseño Ágil con TDD" [11]

9-Metodología ágil para el desarrollo de software.

10-Metodología ágil para el desarrollo de software.

11-Listado "User Stories" a ser desarrollados durante un proyecto que utilice Agile o Scrum.

12-ver en la página 70 del ebook "Diseño Ágil con TDD" [11].

13-ver en la página 71 del ebook "Diseño Ágil con TDD" [11].

14-ver en la página 71 del ebook "Diseño Ágil con TDD" [11].

independiente, sino cumple con estas cuatro premisas no es un test unitario. Un test tiene que ser rápido porque se ejecutan muchos de ellos todo el tiempo, tiene que ser inocuo, ya que no debe alterar el estado del sistema, para ser atómico debe probar la menor cantidad posible de código y por último para ser independiente no debe depender de que otros tests unitarios sean exitosos o fallen para ser él mismo exitoso. Un test unitario se aplica a una parte específica del código[15].

Otros estudios, los cuales se encuentran resumidos en un trabajo de investigadores suecos [12], plantean factores que limitan la adopción industrial de TDD, es decir en el desarrollo de software industrializado, a gran escala. A continuación expondremos estos factores y las conclusiones a las que arribaron dichos investigadores.

Factores que limitan la implementación de TDD:

Incrementan el tiempo de desarrollo: El tiempo de desarrollo se refiere al tiempo necesario para implementar un conjunto dado de requerimientos, en el estudio antes mencionado [9] no se precisa si dentro de este lapso se incluye o no el tiempo de retrabajo necesario para la corrección de los errores que aparecen en etapas de testing posteriores, en nuestra opinión este lapso debe ser incluido como tiempo de desarrollo.

Se plantea que la adopción de TDD incrementa los tiempos en la mayoría de los estudios, aunque en algunos los disminuye. Se realiza la observación de que la madurez de los equipos de desarrollo es fundamental para la adopción de nuevas prácticas dentro de una organización, esto cumple un papel fundamental y también se observa en otros factores que limitan la adopción de TDD. Por otra parte en "Test driven development: empirical body of evidence" [10] exponen que en ambientes de desarrollo industrial de software en algunos estudios se detecta que se reducen los defectos, con lo cual si extrapolamos, se reduce el tiempo de re-trabajo, y en otros estudios se expone que se incrementa la productividad o lo que es lo mismo se reduce el tiempo de desarrollo por requerimiento. En el trabajo de Lech Madeyski y Lukasz Szala [6] se arribó a la conclusión que se incrementa la productividad, en contraposición con el trabajo de Boby George y Laurie Williams [13], para ellos la conclusión es que disminuye.

El primero de los trabajos es mas reciente y toma como base al segundo, nuestro punto de vista es que el mismo mide la productividad de una manera más completa, pues usa una combinación de factores y en el otro trabajo sólo miden el tiempo de desarrollo.

De ambos análisis podemos deducir que dependiendo de los casos de estudio y la forma en la que los mismos son efectuados el tiempo de desarrollo puede ser superior o disminuir, lo cual no arroja claridad sino por lo contrario expone que existen factores que afectan al tiempo de desarrollo y que no es la práctica de TDD, sino que pueden ser por ejemplo la madurez del equipo de desarrollo, los ambientes de desarrollo, el conocimiento de la práctica, de la tecnología utilizada en el desarrollo y porque no hasta la forma de medir los tiempos de desarrollo. Cabe destacar que en las publicaciones de Kent Beck [4] se hace referencia dentro de XP a la refactorización del código para lograr escribir código de calidad, las pruebas unitarias permiten agilizar este proceso. Por otra parte en la misma publicación se especifica que se trabajó con iteraciones cortas con lo cual la funcionalidad se va mejorando e incrementando con el tiempo, en este punto es crucial el contar con una manera rápida de realizar regresiones con la finalidad de estar seguros que el código que estaba

15-ver en la página 71 del ebook "Diseño Ágil con TDD" [11].

funcionando correctamente lo siga haciendo después de los cambios.

Poca experiencia en TDD: En algunos estudios se ha observado que ciertos participantes habían recibido entrenamiento en cómo realizar TDD, en otros casos se observó que adquirían las habilidades necesarias a medida que avanzaban los experimentos[16]. Esta exploración no sólo encontró que la falta de conocimientos es un problema, sino que se encuentran diferencias entre desarrolladores con experiencia y aquellos que están comenzando. Lo que puede ser atribuido a que los desarrolladores experimentados si bien no necesariamente tienen conocimientos de TDD, ya tienen internalizada alguna metodología de desarrollo de software, lo cual puede facilitar el aprendizaje y adopción de una nueva.

Diseño insuficiente: No se ha encontrado suficiente evidencia empírica sobre que la pérdida de diseño o un insuficiente diseño en el software fuese un factor limitante en la implementación de TDD[17]. De todas maneras desde los comienzos de TDD se ha criticado que el mismo ocasiona problemas en el diseño de los sistemas, en especial los de considerable tamaño, aunque la evidencia contradiga esto. En este punto es curioso encontrar como en algunos blogs[18] se ve como un beneficio que al necesitar simplificar y trabajar con iteraciones más cortas se consigue un mejor diseño o que el mismo puede ser ajustado rápidamente.

Habilidades insuficientes en testing por parte de los desarrolladores: Si bien TDD es una técnica o metodología de desarrollo, para utilizar la misma primero es necesario contar con los casos de pruebas, y luego los desarrolladores escriben el código que satisface las pruebas, es decir las pruebas que antes fallaban ahora se ejecutan correctamente. En algunos estudios se encontró que la falta de habilidades para escribir casos de pruebas correctos ha sido considerada como una limitante al momento de adoptar TDD[19]. Esto se ve resaltado en algunos blogs escritos por desarrolladores o en comunidades en las cuales se comparte información entre desarrolladores[20], se enfatiza que desarrollar pruebas de calidad requiere esfuerzo y habilidades, por lo tanto su escasez puede volver difusos los beneficios que se obtienen con esta técnica de desarrollo.

Insuficiente apego al protocolo de TDD por parte de los desarrolladores: Estudios empíricos han mostrado que diversos elementos ocasionan la baja adhesión de los desarrolladores a la utilización estricta de TDD, en muchos casos debido a factores como tiempos reducidos para el desarrollo, poca disciplina o no percibir los beneficios en el corto plazo, esto puede deberse a que TDD es apreciado como lento al principio porque los desarrolladores deben agregar las pruebas[21] o porque los desarrolladores de software encuentran tedioso estar restringidos a seguir un proceso estructurado en el desarrollo del software y la adopción de TDD está ligada a un cambio cultural en el modo de desarrollar software[22]. Si bien no se ve una relación entre la baja adhesión a TDD y la baja calidad en el software, muchas veces se

16-página 5 del paper "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [12].

17-página 5 del paper "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [12].

18-The Pros & Cons of TDD - TechBlog – 11 de Junio de 2012 – PathFinder Solutions - http://www.pathfindersolns.com/archives/1684

19-página 5 del paper "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [12].

20-The Pros & Cons of TDD - TechBlog – 11 de Junio de 2012 – PathFinder Solutions - http://www.pathfindersolns.com/archives/1684

21-The Pros & Cons of TDD - TechBlog – 11 de Junio de 2012 – PathFinder Solutions - http://www.pathfindersolns.com/archives/1684

22-The Pros & Cons of TDD - TechBlog – 11 de Junio de 2012 – PathFinder Solutions - http://www.pathfindersolns.com/archives/1684

confunden los factores y ambos por ejemplo son atribuidos a los tiempos reducidos para el desarrollo y no a que están relacionados entre ellos[23].

Limitaciones en el dominio de las herramientas específicas de TDD: Muchos trabajos formularon haber tenido experiencias negativas con la implementación de herramientas para TDD. Es vital que al momento de implementar esta técnica se cuente con las herramientas apropiadas para el desarrollo de la misma. Dado que existen numerosos estudios que exponen esta problemática, es algo muy difícil de ignorar al momento de implementar TDD[24]. Cabe destacar que no sólo es necesario contar con las herramientas, sino que además hay que tomar en cuenta el tiempo necesario para el entrenamiento del equipo en el uso de dichas herramientas[25].

Código legado: como se expone en algunos trabajos [12] TDD en su forma original no expresa nada sobre cómo trabajar con código legado, es más, en muchos casos presupone que todo el código se hace desde cero cuando se comienza con TDD. Esto es algo raro de esperar en la mayoría de las organizaciones que desean adoptar TDD[26] y puede ser un problema al momento de comenzar con TDD, pues se podrían encontrar problemas sobre como los nuevos cambios afectarán los viejos, produciendo preocupaciones entre los desarrolladores.

Discusión

Como primer punto de discusión se cree necesario establecer que TDD como metodología de software aparece hace unos 10 a 12 años, como parte de XP, por lo tanto no existe mucho trabajo formal sobre la misma, sino que la mayoría son estudios empíricos [12], [6], [13] y que se basan en trabajos realizados sobre grupos que desarrollan software en forma industrial y sobre grupos que se formaron en Universidades para realizar investigaciones. De los trabajos relevados y estudiados se pueden inferir ciertas cuestiones, la primera que TDD no es una metodología para equipos de testing, sino que es para equipos de desarrollo, 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. En adición a esto los desarrolladores deben lograr que las pruebas no fallen más, como paso previo a la refactorización. Esta forma de implementación de TDD es la que origina que los que no están de acuerdo con esta metodología argumenten que es necesario contar con equipos altamente calificados [12]. Referido a esto en algunos trabajos [11] se han expresado que por lo contrario los desarrolladores menos experimentados aprenden al implementar TDD, ya que se ven obligados a escribir los tests primero con lo cual se enfocan en la funcionalidad que deben implementar desde un ángulo distinto, el de imaginar cómo probar dicha funcionalidad antes que imaginar cómo implementar el código que la satisfaga. Esto significa además una ventaja para los desarrolladores, pues logran una mejor comprensión de las funcionalidades que están desarrollando. Esta ventaja trae aparejada una mejora en la productividad ya que como se observa en el trabajo de Lech Madeyski y Lukasz Szala, trabajo mas reciente que otro que usaron como base [13] en el cual la productividad disminuía. La productividad está ligada a la naturaleza

23-página 5 del paper "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [12].

24-página 5 del paper "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [12]

25-The Pros & Cons of TDD - TechBlog – 11 de Junio de 2012 – PathFinder Solutions - http://www.pathfindersolns.com/archives/1684

26-página 6 del paper "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review" [12].

del problema y la madurez del equipo. Esto lo deducimos de analizar lo que se plantea como limitaciones para la implementación [12], una de las cuales son las habilidades del equipo, esto no es una limitación, sino que es algo a tener en cuenta al momento de realizar comparaciones entre equipos. Tal como lo expresa Carlos Fontela [5] 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.

El otro punto en el que se encuentran oposiciones entre los partidarios de TDD y los que plantean objeciones a su implementación es la aplicación de los mismos a grandes proyectos. En este punto se puede ver la explicación de Carlos Ble Jurado [11] sobre que podemos atacar los grandes proyectos como una colección de proyectos más pequeños.

En esta cuestión también, podemos sumar la propia experiencia personal de un miembro del grupo que escribe el presente trabajo por haber participado en el desarrollo del sistema de reservas de vuelo de una aerolínea estadounidense , el cual no sólo permite la búsqueda de vuelos y la reserva de los mismos para todo Estados Unidos, con sus combinaciones, sino además la reserva de automóviles y hoteles entre otras funcionalidades. Lo más importante es que en este proyecto participan alrededor de 300 personas, se aplicó Agile como proceso para la gestión y todo el desarrollo se hacía utilizando TDD e "Integración Continua"[27] [14], lo cual permitió que cada desarrollador fuera responsable de sumar código sin romper el código previo, tarea que es controlada con las pruebas unitarias previamente desarrolladas, y además sumaban pruebas de integración y pruebas funcionales automatizadas, las cuales pueden o no ser una derivación de las pruebas unitarias.

Un aspecto que es comúnmente hallado en los trabajos que plantean limitaciones o problemas en la adopción de TDD[28] como parte del desarrollo de software es que los equipos de trabajos confunden TDD como una técnica de testing, esta confusión lleva a que los equipos de desarrollo no escriban las pruebas unitarias antes que el código que implementa una funcionalidad dada, sino que las dejen para más tarde. Esto provoca dos situaciones, primero que no se aproveche el potencial que implica TDD como metodología de diseño de software y herramienta de aprendizaje para los equipos de desarrollo. La segunda situación es que al dejar las pruebas para el último se cae en la subjetividad de ver a TDD como una pérdida de tiempo, como algo que aporta pocos beneficios en contrapartida del esfuerzo que hay que realizar, o algo que suele ser aún peor, y es que al no escribirse las pruebas antes, no se las considera dentro de la estimación con lo cual luego no hay tiempo para escribirlas. Esta confusión se explica con las palabras que Carlos Fontela [5] utiliza en su trabajo de especialización "El primer problema proviene del propio nombre de TDD, que incluye la palabra "test". En efecto, mientras por un lado se afirma que es una técnica de diseño y no de pruebas, por otro el nombre invita a pensar lo contrario. Incluso las herramientas que fueron surgiendo a partir de TDD, desarrolladas incluso por los mentores de la práctica, requerían que el código de pruebas tuviese métodos cuyos nombres empezasen con test y las clases fueran descendientes de algo como TestCase (si bien NUnit y la versión 4 de JUnit mejoraron esto, sigue estando presente la palabra Test en las anotaciones que utilizan). Nadie niega que TDD genera un buen conjunto de pruebas

27-Es una de las prácticas enunciadas en XP.

28-"Problems with TDD" - http://dalkescientific.com/writings/diary/archive/2009/12/29/problems_with_tdd.html - Copyright © 2001-2010 Dalke Scientific Software, LLC.

de regresión, pero ese no pretende ser su objetivo principal, sino más bien un efecto lateral positivo"[29][5].
Otro aspecto que se deriva de los trabajos es expresado también por Carlos Fontela[30], TDD no reemplaza las pruebas funcionales de humanos ya que toda la seguridad que nos da TDD viene de código escrito por humanos, el cual puede contener fallas. Por otro lado si TDD ha sido aplicado rigurosamente y revisado a conciencia brinda un nivel de calidad y seguridad al momento de incorporar cambios en las funcionalidades que ya existen.
En varios estudios se apreció una posible confusión al considerar a TDD como una técnica de testing, con lo cual muchos desarrolladores lo ven como algo a realizar a posterior del desarrollo del código, y en muchos casos se toma la decisión de no hacerlo por falta de tiempo.

Conclusión

Al comenzar este trabajo describimos de qué se trata TDD, en qué consiste, cuál es su estado de adopción actualmente en el desarrollo del software, para qué sirve realmente, si es posible utilizarlo en proyectos de distintos tamaños y cuáles factores limitan la utilización de esta metodología.
De lo expuesto se deduce que es necesario realizar más estudios sobre los beneficios y limitaciones que se plantean a TDD, en especial estudios que se enfoquen en los diferentes escenarios que se presentan en el desarrollo de software. Es preciso tener en cuenta que esta industria evoluciona y cambia constantemente, lo cual hace que aquellas limitaciones que existen en determinados momentos luego desaparecen o cambian de forma. Además los estudios se deberían enfocar en la manera como las personas trabajan dentro de los grupos de desarrollo de software y como ven el desarrollo de software, para lograr enfocar TDD como una práctica útil en la industria.
Otro aspecto interesante que se encontró durante la revisión de los trabajos es que los principales promotores de la implementación de TDD como metodología de desarrollo de software explican los fundamentos, las prácticas y la metodología utilizando siempre ejemplos de proyecto nuevos que están comenzando[31] [4], [5], [7].
Esta forma de presentar la metodología ha dado lugar a una de las principales objeciones o limitaciones que se le encuentra a TDD, que es la utilización del mismo en proyectos ya comenzados o "Código Legado" [32] en algunos trabajos [12] se lo marca como una limitación para implementar TDD. En este punto pensamos que hay escasa documentación en ambos sentidos, en un aspecto más pragmático, se pueden utilizar las palabras de Carlos Ble Jurado[33] [11] para decir que si estamos trabajando con código legado podemos empezar con la nueva funcionalidad a aplicar TDD e ir sumando TDD en los cambios que vaya sufriendo el código ya existente.

Algo que se desprende de los trabajos 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.
De todos los trabajos que se analizaron asimismo se llegó a establecer que TDD se

29-página 9 del trabajo de especialización.
30-Página 43 en la conclusión de su trabajo de especialista [5].
31-Kent Beck explica en "TDD by Example" mediante un proyecto desde cero.
32-Código heredado o sistemas heredados que lleva mucho tiempo de desarrollo, el cual muchas veces no se dispone documentación.
33-Página 55 del su libre "Diseño Ágil con TDD" [11]

puede aplicar a cualquier tipo de proyecto, no importa el tamaño ya que un proyecto grande es necesario subdividirlo para lograr alcanzar el éxito y que la metodología se debe utilizar para lo que fue creada, es decir una metodología de diseño de software y para la programación del mismo y no utilizarlo para el testeo del software una vez que fue desarrollado.

En conclusión, TDD lleva más de 10 años en la industria del software y su principal aplicación en la actualidad es en proyectos con metodologías ágiles tal como XP, Agile o Scrum, pero se puede ver que es posible aplicarlo en otros proyectos y no sólo en proyectos nuevos. 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, caso contrario formularlas.

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

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

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

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

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

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

[12] 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.

[13] Boby George, Laurie Williams - An Initial Investigation of Test Driven Development in Industry, 2003.

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

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

Datos de Contacto:

Ing. Pablo Andrés Vaca. Universidad Tecnológica Nacional – Facultad Regional Córdoba. pvaca@sistemas.frc.edu.ar.

Ing. María Soledad Romero. Universidad Tecnológica Nacional – Facultad Regional Córdoba. romeroma.soledad@gmail.com.

Mg Claudia Inés Inchaurrondo. Universidad Tecnológica Nacional – Facultad Regional Córdoba. *cinchaurrondo@sistemas. frc. utn.edu.ar.*

Ing. Calixto Maldonado. Universidad Tecnológica Nacional – Facultad Regional Córdoba. cmaldonado@sistemas.frc.utn.edu.ar.

Ing. Juan Peretti. Universidad Tecnológica Nacional – Facultad Regional Córdoba. *peretti.juan@gmail.com.*

Ing. Matías Bueno. Universidad Tecnológica Nacional – Facultad Regional Córdoba. matiasbueno@gmail.com.