DESARROLLO DE SOFTWARE GUIADOS POR PRUEBAS (TEST-DRIVEN DEVELOPMENT)

Una Revisión

Nunes, M.; Rodríguez, A.; Vegas, R.
Ingeniería de Software
Programa de Especialización en Desarrollo de Software
Facyt-UC

***Resumen*- Este documento presenta una revisión de la técnica de desarrollo basado en pruebas (*Test-Driven Development - TDD*) así como del ciclo básico que lo compone. Dicha técnica incluye una serie de pruebas que son igualmente presentadas en el documento. La *IEEE* respecto a esta técnica propone un conjunto de estándares que deben ser aplicados para la obtención de mejoras significativas en el proceso de desarrollo, específicamente en la fase de codificación. Finalmente, se presenta un ejemplo de la inclusión del TDD en la metodología de desarrollo ágil *Scrum* y los resultados obtenidos de dicha combinación.**

Palabras clave—Test-Driven Development, pruebas, IEEE, Metodología de Desarrollo Ágil, Scrum.

***Abstract*—This paper presents a technical review of development based on test (Test Driven Development - TDD), well as the basic cycle that the component. The technical includes a series the test that are presented in this paper. The IEEE proposes on this technique a set the standards to be applied and obtaining significant improvements in the development process, specifically in the coding phase. Finally, this paper presents an example of the inclusion of TDD in the agile development methodology, Scrum and the results of the combination.**

Keywords—Test-Driven Development, testing, IEEE, Agile Development Methodology, SCRUM.

I. INTRODUCCIÓN

Cada año se pierden miles de millones de dólares a causa de los defectos del software, según cálculos proporcionados por el Instituto Nacional de Estándares y Tecnología (*NIST*) de los Estados Unidos, en el año 2002 las pérdidas por errores del software en el mercado norteamericano rondaron los 67,5 millones de dólares, lo que equivale al 0.6% del Producto Interno Bruto del país [1]. Para dar solución a este problema, se debe llevar a cabo, entre otras cosas, un proceso de pruebas de software bien estructurado y con un enfoque disciplinado.

Las pruebas de Software como una actividad integrada en el ciclo de desarrollo de software, se utiliza para ayudar a medir la exactitud, integridad, seguridad y calidad del sistema que se está desarrollando. Es un proceso inherente a garantizar que se cumplan todos los estándares de calidad, además de cumplir con los requerimientos del negocio definidos, procurando la aceptación por parte de los clientes y usuarios, actores influyentes en el éxito de cualquier proyecto. Para llevar a cabo la ejecución de las pruebas en las actividades de desarrollo de software, se necesita de un enfoque sistemático, por la cual se propone en el siguiente artículo una revisión de la Metodología de Desarrollo Basado en Pruebas (*Test-Driven Development - TDD*).

Adicionalmente, se presenta un resumen de los distintos niveles de pruebas que emplea la metodología antes mencionada, así como, algunos estándares internacionales propuestos por la *IEEE* relacionados a la disciplina de pruebas de software. Por último, se presenta un ejemplo de la inclusión del *TDD* en metodologías de desarrollo ágil.

II. METODOLOGÍA DE DESARROLLO BASADA EN PRUEBAS (*TEST DRIVEN DEVELOPMENT*)

Tal y como lo expresa Nicolás [2], el Desarrollo basado en Pruebas (*Test-Driven Development - TDD*) es un enfoque de la Ingeniería de Software que consiste en iteraciones cortas de desarrollo, donde los casos de prueba que cubren una funcionalidad son escritos antes de iniciar el desarrollo. El código necesario para pasar las pruebas es implementado y probado con los casos de prueba definidos.

En caso de que se presenten errores, estos son solventados por los desarrolladores y de ser necesario se procede a refactorizar el código.

A. Ciclo Básico del TDD

El desarrollo guiado por pruebas requiere que los desarrolladores elaboren pruebas unitarias automatizadas que definan los requisitos del código justo antes de escribir el código. Las pruebas contienen afirmaciones que pueden ser verdaderas o falsas. Al pasar las pruebas, se confirma el comportamiento correcto del programa y posteriormente, los desarrolladores proceden a refactorizar el código [3].

En la *Fig. 1* presentada por Duka *et al.* [4] muestra el ciclo básico de TDD:

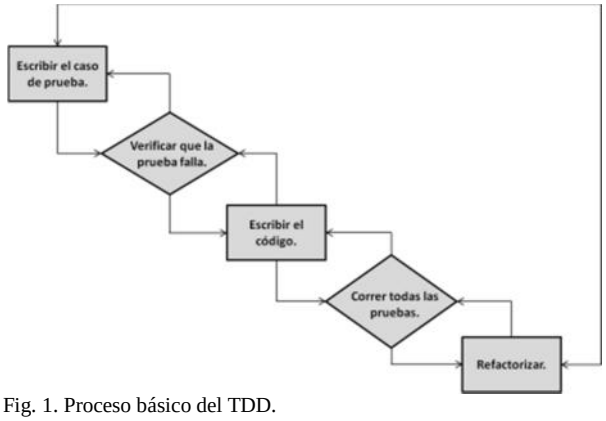

Fig. 1. Proceso básico del TDD.

Según Beck [4], un desarrollador que emplee el TDD tradicional, lleva a cabo los siguientes pasos.

1) Escribir el caso de prueba: En el desarrollo basado en pruebas, cada nueva funcionalidad comienza con la escritura de una prueba. Esta prueba debe fallar inevitablemente. Para escribir una prueba, el desarrollador debe entender claramente las especificaciones y los requisitos de la funcionalidad. Esto se puede lograr a través de la utilización de casos de uso y casos de usuario que cubren los requisitos y condiciones de excepción.

2) Ejecutar el caso de prueba y verificar que ha fallado como se esperaba: Esto confirma que el instrumento de prueba está funcionando correctamente y que la nueva prueba no pasa por error sin necesidad de ningún código nuevo. Este paso también prueba la prueba en sí, en sentido negativo: se descarta la posibilidad de que la nueva prueba siempre va a pasar.

3) Implementar el código que impide que el caso falle: El siguiente paso es escribir algo de código que hará que la prueba pase. El nuevo código escrito en esta etapa no va a ser perfecto, lo que es aceptable, ya que los pasos posteriores serán mejorarlo y perfeccionarlo. Si todos los casos de prueba ahora pasan, el programador puede estar seguro de que el código cumple con todos los requisitos de prueba. Este es un buen punto de partida para comenzar el paso final del ciclo.

4) Refactorizar el código en caso de ser necesario: Según Fowler [5], el término "Refactoring" hace referencia a una técnica disciplinada para la reestructuración de una entidad de código existente, alterando su estructura interna sin cambiar su comportamiento externo. Su corazón es una serie de pequeñas transformaciones de conservación. Cada transformación hace poco, pero una secuencia de transformaciones puede producir una reestructuración significativa. En el TDD la refactorización es realizada con el fin de eliminar duplicidades existentes o incluso mejorar el código desarrollado.

III. NIVELES DE PRUEBA

Las pruebas de software denominada por [6] como una Área de Conocimiento, se realizan normalmente a diferentes niveles durante los procesos de desarrollo y mantenimiento. A este respecto, plantea [7] que el objeto de las pruebas pueden cambiar: (1) realizar pruebas a través de cada uno de los módulos definidos dentro de la aplicación, (2) realizar pruebas a través de un conjunto de módulos y (3) validar y verificar el sistema en su totalidad. En efecto, las pruebas de software implican integrar dos o más componentes que implementan funcionalidades del sistema para luego probar estos en el producto final.

Según [6, 7, 8], se pueden distinguir 3 grandes niveles de pruebas, estos niveles son los siguientes: Pruebas Unitarias, Pruebas de Integración y Pruebas del Sistemas.

A. Pruebas Unitarias

Las pruebas unitarias verifican el funcionamiento de las partes de software que se pueden probar independientemente. Dependiendo del contexto, estas partes pueden ser módulos de código, así se asegura que cada módulo de la aplicación final funciona correctamente por separado. Estas pruebas fueron definidas por [9], la cual también proporciona un método para llevar a cabo estas pruebas a través de un enfoque sistemático y documentado. Normalmente, las pruebas unitarias se realizan a través del código fuente y con el soporte de herramientas de depuración, por lo cual los programadores son protagonistas en la ejecución de las mismas.

El *TDD* se basa en estas pruebas pero con la diferencia de que no se testea código fuente, sino se evalúan funcionalidades que serán definidas dentro de la aplicación con el fin de controlar el proceso de programación [10].

B. Pruebas de Integración

Las pruebas de integración es el proceso de verificar la interacción entre componentes de software. Los componentes que se integran pueden ser

componentes comerciales, componentes reutilizables que han sido adaptados a un sistema en particular o componentes nuevos desarrollados.

En el ejemplo mostrado en la *Fig. 2*, se puede apreciar cómo se lleva a cabo las pruebas de integración. A, B, C y D son componentes del sistema y desde T_1 hasta T_5 son los casos de pruebas relacionados a las funcionalidades incorporadas al software.

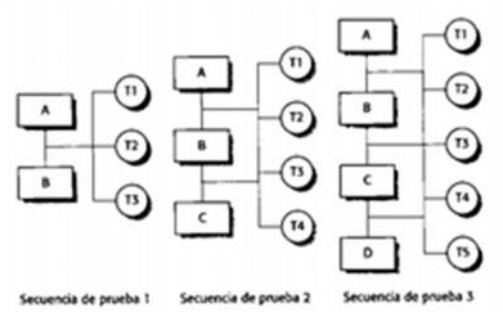

Fig. 2. Ejemplo de Pruebas de Integración [11].

C. *Pruebas del Sistema*

Es el mayor de los test de integración, ya que integra varias partes del sistema como un todo. Las pruebas del Sistema se ocupan del comportamiento del software final. Se consideran normalmente como las más apropiadas para verificar que el sistema cumpla con los requisitos no funcionales definidos, tales como: seguridad, velocidad, exactitud y confiabilidad.

Los niveles de pruebas explicados anteriormente son definidos por el *TDD* como pruebas ejecutadas por el equipo de desarrollo para verificar componentes, funcionalidades y el producto final obtenido. De igual manera, la [6] propone asignar casos de pruebas para comprobar que las especificaciones funcionales se han implementado correctamente, esto se logra a través de la ejecución de las pruebas de aceptación y las pruebas funcionales. Estas pruebas son ejecutadas por los clientes o usuarios del sistema, por la cual se recomienda que dichos actores estén inmersos en la ejecución del *TDD* [12].

D. *Pruebas de Aceptación / Cliente*

Las pruebas de aceptación comparan el comportamiento del sistema con los requisitos obtenidos por parte de los clientes y usuarios. Estos actores especifican tareas típicas como test de aceptación escritos por medio de lenguaje natural *(Fig. 3)*, con el fin de comprobar que se satisfacen sus requisitos.

- Clientes pertenecientes al grupo del consorcio no se le cobraran intereses a las compras que realicen en modalidad de crédito.
- Artículos con un precio mayor a Bs. 10000 no se le aplica descuento mayor al 10%

Fig. 3. Ejemplo de Test de Aceptación.

E. *Pruebas Funcionales*

Según [13, 14], las pruebas funcionales son aquellas que se realizan para verificar si el comportamiento del software corresponde con las especificaciones. Una prueba funcional es un subconjunto de pruebas de aceptación. Estas comprueban las funcionalidades a través de la lógica de negocio de la organización.

A continuación, se presenta en la *Fig. 4* el esquema de los niveles de pruebas definidos anteriormente y que son utilizados por el *TDD*:

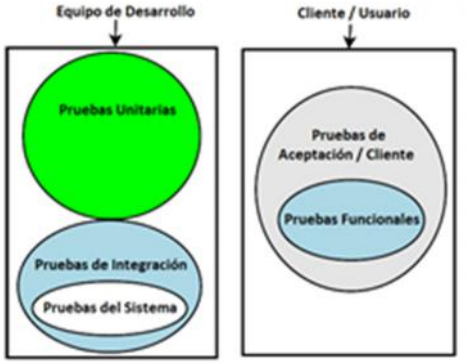

Fig. 4. Pruebas utilizadas en el enfoque *TDD.*

Ahora bien, una vez definido cada uno de los niveles de pruebas enmarcados en el *TDD (Fig. 4)*, se necesita de ciertos lineamientos y estándares que proporcionen un marco de trabajo para la ejecución sistemática y disciplinada de los niveles de pruebas antes definidos o la mayoría de ellos. En efecto, organizaciones como la *IEEE* definen ciertos estándares relacionados con la disciplina de pruebas de software.

IV. ESTÁNDARES DE PRUEBAS DE SOFTWARE

Por medio de los estándares se establece el enfoque sobre el cual estará basada la prueba de software. Entre los estándares más importantes de acuerdo al estudio de [15], tenemos:

A. Estándar ANSI/IEEE Std 1008-1987

Según [15], el principal objetivo de este estándar es proponer un enfoque para la aplicación de las pruebas unitarias. Este enfoque definido por [9], está basado en tres fases:

1) Fase de planificación de pruebas: enfoque de plan general, recursos y programación.
2) Obtener casos de prueba: Diseña los casos de pruebas, implementa, refina el plan y el diseño.
3) Fase de medición de pruebas unitarias: Ejecuta los procedimientos de las pruebas, verifica el fin de la prueba, evalúa el esfuerzo y unidad de la prueba.

En la *Fig. 5*, se puede apreciar que las fases que se realizan de manera secuencial, a excepción de la ejecución y la verificación, debido a que puede surgir la necesidad de retroceder a alguna de las fases anteriores para luego volver a la fase actual.

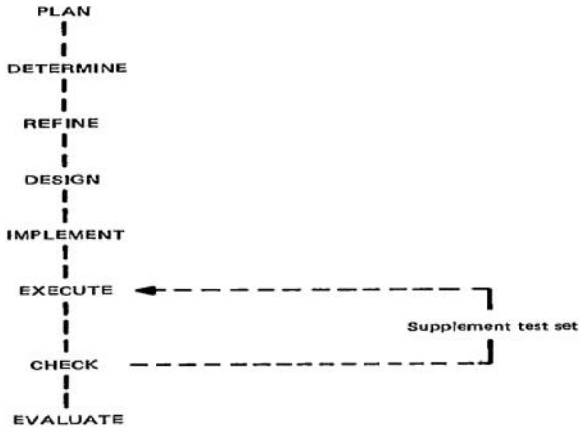

Fig. 5. Fases de las Pruebas Unitarias [9].

B. Estándar para la Documentación de las Pruebas de Software IEEE Std 829-1998

La documentación formal permite tener una referencia de las diferentes fases y actividades de las pruebas de software que lleva a cabo el equipo de desarrollo. De acuerdo a [16], este estándar describe un conjunto de documentos que están asociados con los aspectos dinámicos de software prueba (es decir, la ejecución de procedimientos y códigos). El estándar define el propósito, el contorno y contenido de cada documento.

En este sentido, los documentos que tienen las especificaciones de las pruebas son:

- La especificación del diseño de prueba
- La especificación de los casos de prueba.
- Especificación de los procedimientos de prueba.

Adicionalmente, se generan los siguientes informes:

- Informe de los elementos que están siendo probados.
- Informe histórico de las pruebas.
- Informe de incidentes de prueba.
- Informe de actividades de prueba.

C. Estándar para Pruebas de Componentes de Software BS 7925-2

En términos formales, y de acuerdo con [17], un componente es una pieza de software que cumple con dos características: no depende de la aplicación que la utiliza y, se puede emplear en diversas aplicaciones. La interacción de los componentes puede exponer el sistema generando fallas considerables, sin duda alguna, las pruebas de componentes se hacen necesarias para minimizar los riesgos; con esta finalidad, se sugiere la adopción de estándares que generen la documentación con el fin de facilitar la ejecución de todas las fases de las pruebas. De acuerdo a [15], el estándar para las pruebas de componentes de software *BS 7925-2* prescribe las características del proceso de pruebas y las técnicas para diseñar y medir un caso de prueba, siendo el instituto "*British Computer Society"* el ente que promueve este estándar *(Fig. 6)*. Las técnicas de diseño que se definen son las siguientes:

- Partición equivalente.
- Análisis de valor límite.
- Transición de estados.
- Grafo causa- efecto.
- Prueba de ramificación.
- Prueba de combinación de ramificaciones.
- Pruebas aleatorias.

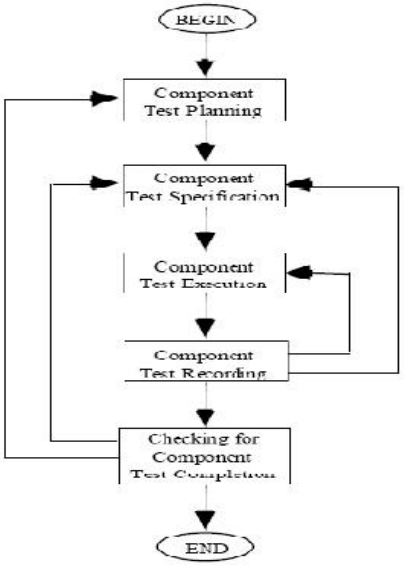

Fig. 6. Procesos de Prueba de Componentes [15].

Por otra parte, surge el estándar *ISO/IEC 29119* que de acuerdo a [18], tiene como objetivo cubrir todo el ciclo de vida de las pruebas de sistemas software incluyendo los aspectos relativos a la organización, gestión, diseño y ejecución de las pruebas, para reemplazar varios estándares entre los que se encuentran la *IEEE* y *BSI*.

Según [19], la *ISO/IEC 29119* comprende cinco partes:

1) Definiciones y vocabulario.
2) Proceso de prueba.
3) Documentación de prueba: La fase de documentación de prueba de acuerdo a lo que plantea[19] comprende plantillas que se pueden adaptar a cada una de las necesidades de la organización, estás plantillas incluyen:

a) Documentación de los procesos de prueba organizacional:

- Prueba de organización política.
- Estrategia organizacional de prueba.

b) Documentación de los procesos de gestión de pruebas:

- Plan de pruebas (incluye estrategia de prueba).
- Informe sobre la situación de prueba.
- Informe de fin de prueba.

c) Documentación de los procesos dinámicos de prueba:

- Prueba de las especificaciones de diseño.
- Especificación de pruebas del caso.
- Especificación del procedimiento de prueba.
- Requisitos de los datos de prueba.
- Requisitos detallados del procedimiento de prueba.
- Informe de prueba de preparación para el Medio Ambiente.
- Resultado de la prueba.
- Registro de la prueba de ejecución.
- Informe de Incidentes de prueba.

4) Técnicas de prueba.
5) ISO/IEC 33063 modelo del proceso de evaluación para los procesos de pruebas de software.

En sus fases iniciales la *ISO/IEC 29119* se basó en el estándar de documentación de prueba de la *IEEE 829*, sin duda y de acuerdo a lo expuesto por [19], la *ISO/IEC 29119* reemplazará a la *IEEE 829*.

Si bien es cierto que los estándares formalizan las pruebas de software y permiten detectar los errores que pueden presentarse, estos deben ir de la mano con una metodología que de manera sistémica lleve a cabo las distintas actividades de una organización.

V. TDD EMPLEADO EN METODOLOGÍAS ÁGILES

El concepto de desarrollo basado en pruebas se ha utilizado esporádicamente durante largo tiempo, se tiene conocimiento de su uso desde finales de 1960. Sin embargo, se hizo popular con la aparición de la práctica de desarrollo *eXtreme Programming (XP)* propuesta por Kent Beck a finales de los años 1990. El *TDD* se deriva del desarrollo de tecnologías ágiles, específicamente de *XP* [20].

Brown [21] comenta que la utilización del *TDD* como una técnica específica, es igualmente atribuible a Kent Beck, quien lo planteó formalmente en su libro *Test Driven Development: By Example*, en el año 2003. Otros enfoques de desarrollo como *Scrum* y el desarrollo rápido de aplicaciones *(RAD)* han incorporado el *TDD* a la fase de codificación.

Como ejemplo de la incorporación del *TDD* a metodologías ágiles, Schmidkonz *et al.* [22] presentan un resumen de lo que ha sido el resultado de la combinación de *Test Driven Development* y *Scrum* para la realización de proyectos de desarrollo sobre el *ERP SAP*. En la imagen que se presenta a continuación, se muestran las prácticas de la metodología *XP* y como éstas son integradas a las fases contempladas por *Scrum*.

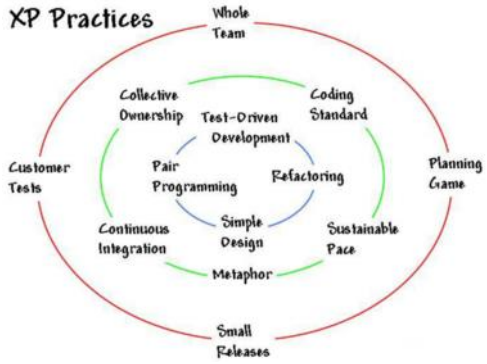

Fig. 7. Prácticas de XP.

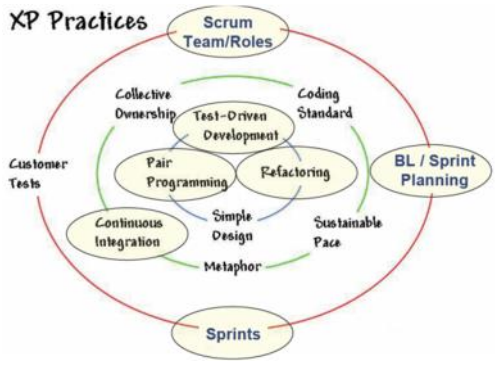

Fig. 8. Scrum y TDD.

Schmidkonz *et al.* [22] presentan el resultado de su trabajo explicando los principales beneficios que produjo la utilización del *TDD* desde el punto de vista de los distintos roles involucrados en la metodología *Scrum.*

A. Dueño del producto (Product owner)

Obtención de funcionalidades más completas, con pruebas automatizadas y trabajo continuo.

B. Equipo (Team)

Planificación más fácil, menos depuración y mayor confianza.

C. Scrum master

Mayor confianza en la entrega.

D. Producto final

Mayor calidad tanto interna como externa y reducción de riesgos a través del conocimiento compartido.

VI. CONCLUSIONES Y RECOMENDACIONES

Si bien es cierto que los analistas deben responder de manera rápida a los requerimientos de los usuarios, estos no deben dejar de lado la necesidad de generar un software de calidad. Una de las causas por las que un software fracasa es debido a que no se llevan a cabo de manera sistémica las pruebas del software. El enfoque de desarrollo basado en pruebas *(Test-Driven Development - TDD)* considera las funcionalidades que formarán parte de un conjunto de pruebas durante un ciclo iterativo que arrojará los errores necesarios y afinará el código. De manera inconsciente, los desarrolladores al momento de hacer las pruebas toman los casos que satisfacen los resultados, es por ello que es necesario integrar en el *TDD* un conjunto de pruebas que permiten arrojar todos esos errores que no logran ser captados por el analista. Adicionalmente, las pruebas nos permiten seleccionar los casos de relevancia ya que es prácticamente imposible poder probar el 100% de estos. Aun cuando se hagan las pruebas respectivas y el software sea puesto en marcha siempre va requerir mantenimiento, es por ello que se han definido estándares que formalizan las pruebas de software y facilitan el mantenimiento. Ante lo planteado, los analistas deben considerar todas las pruebas de relevancia que les permitan ubicar la mayor cantidad de errores para generar un software de calidad y facilitar el mantenimiento del mismo.

VII. REFERENCIAS

[1] P. Casper, M. Blokpoel, “Los Costos de Subestimar los Defectos del Software”, Sogeti, España, 2002. [Online]. Available: http://www.es.sogeti.com/Global/ART%C3%8DCULOS/ART_MARTIN_CASPER.pdf

[2] P. Nicolas, “Introduction to Test Driven Development Methodology”. March 2006. [Online]. Available: http://www.pnexpert.com/files/Test_Driven_Development.pdf

[3] Duka, L. Hribar, “Test Driven Development Method in Software Development Process.” Research & Development Center. Split, Croacia.

[4] K. Beck, Test-Driven Development by Example. The Addison-Wesley Signature Series, 2003.

[5] M. Fowler, “Refactoring Home Page”. [Online]. Available: http://www.refactoring.com/

[6] IEEE Computer Society, “SWEBOK - Guide to the Software Engineering Body of Knowledge”, 2004. [Online]. Available: http://www.computer.org/portal/web/swebok/htmlformat.

[7] Beizer, “Software Testing Techniques”, International Thomson Press, 1990.

[8] S. Pfleeger, Software Engineering: Theory and Practice, 1995.

[9] An American National Standard, “IEEE Standard for Software Unit Testing”, ANSI/IEEE Std 1008-1987.

[10] M. Hennessy, “A test-driven development strategy for the construction of grammar-based software”, Ph.D. dissertation, Departament of Computer Science, National University of Ireland, Kildare, Ireland, 2006.

[11] Sommerville, Ingeniería de Software, Pearson Addison Wesley, Madrid, 2004.

[12] Blé, J. Beas, Diseño Agil con TDD, Primera Edición, SafeCreative, Enero, 2010.

[13] Kaner, J. Frank, H. Nguyen, “Testing Computer Software”, John Wiley & Sons, 1999.

[14] Weyuker, S. Weiss, D. Hamlet, “Comparison of Program Test Strategies”, presentado en la Conferencia de Pruebas, Análisis y Verificación, Victoria, Canadá, 1991.

[15] H. Parada, “Contribución a la Gestión de los Procesos de Pruebas de Software y Servicios”. Ph.D. dissertation, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, España, 2010.

[16] IEEE Computer Society, ”IEEE Standard for Software Test Documentation”, IEEE Std 829-1998

[17] R. Martin, H. Odell, “Object oriented methodology: pragmatic considerations”, Prentice Hall, Upper Saddle River.1996.

[18] Grupo de Trabajo AEN/CTN71/SC7/GT26, “Pruebas de Software”. [Online]. Available: http://in2test.lsi.uniovi.es/gt26/

[19] R. Stuart, “ISO/IEC 29119 Software Testing”. [Online]. Available: http://softwaretestingstandard.org/

[20] L. Damm, L. Lundberg, “Early and Cost-Effective Software Fault Detection”, International Conference on Software Engineering, Proceedings of the 29th international conference on Software Engineering, pp. 560–570.

[21] Brown, “Test Driven Development (TDD)”. [Online]. Available: http://www.cs.nott.ac.uk/~cah/G53QAT/Reports09/dxb17u/QAT09Report-dxb17u.doc

[22] C. Schmidkonz, J. Staader, SAP AG. “Piloting of Test Driven Development in Combination with Scrum. An experience report.”, London Scrum Gathering, Noviembre, 2007. [Online]. Available: http://www.scrumalliance.org/resources/267