Modelando Casos de uso basados en el enfoque Iconix/DDT

-Diseño guiado por Testing-

Susana Beatriz Chavez[1], María Inés Lund[2], Adriana Martin[1]
Emilio Gustavo Ormeño[2]

[1] Departamento de Informática de la FCEFN de la UNSJ
[2] Instituto de Informática de la FCEFN de la UNSJ
Complejo Islas Malvinas. Cereceto y Meglioli. 5400. Rivadavia. San Juan

{schavez, mlund}@iinfo.unsj.edu.ar, adrianamartin1@gmail.com, eormeno@iinfo.unsj.edu.ar

Resumen

El testing es una actividad realizada para evaluar la calidad de un producto y mejorarlo a través de la identificación de defectos y problemas [1].

Como se menciona en varias bibliografías, [2], [3], [4] el fracaso de un proyecto de software es a menudo consecuencia de la mala gestión. En general, el desarrollo de software es propenso a subestimar el esfuerzo que se requiere, incluso se deja de lado los procesos de testing. Esto ha revelado ser parte de los problemas o incluso ser la causa principal, sobre todo si se tiene en cuenta que algunas fallas se podrían haber evitado si el software hubiese sido testeado más a fondo.
Los testing de software, como una parte fundamental del proceso de desarrollo, son esenciales para el éxito de la producción de software de alta calidad. Los defectos y las fallas pueden deberse a diferentes causas, como a los problemas de comunicación entre el cliente y los diseñadores, a los procesos de diseño inmaduros -desde la recopilación de requerimientos hasta el diseño detallado y arquitectura- o incluso a los malentendidos o transcripciones incorrectas de los requerimientos.
Ahora bien, qué es testing? Se lo podría definir como un conjunto de actividades que tiene por objetivo identificar las fallas en un software y evaluar su nivel de calidad para obtener la satisfacción del usuario. Se trata de un conjunto de tareas con objetivos claramente definidos [4].
La visión de las pruebas de software o testing ha evolucionado hacia una postura más constructiva. La prueba ya no es vista como una actividad que se inicia sólo después de que termine la fase de codificación, con el propósito de detectar fallas. El testing es visto ahora como una actividad que debe abarcar todo el proceso de desarrollo y mantenimiento y es en sí mismo una parte importante en la construcción actual del producto.

Por tal motivo, las metodologías ágiles eligen *testing* como una técnica para aumentar la confiabilidad (reliability[1]) de sus productos de software [5].

[1] Reability (ISO 9126) es la capacidad del producto de software para mantener el nivel especificado de

Palabras clave: testing, proceso de desarrollo de software, modelo de casos de uso.

Contexto

El presente trabajo se encuadra dentro del área de I/D Innovación en Sistemas de Software, y se enmarca dentro del proyecto de investigación "Modelo de casos de uso. Línea base para el desarrollo de software", el cual tiene como unidades ejecutoras al Departamento e Instituto de Informática de la FCEFyN de la UNSJ. Los trabajos iniciados en el citado proyecto pretenden obtener, en forma automática o semi-automática, modelos que sean útiles a las diferentes etapas del desarrollo de software para construir un producto de calidad.

Introducción

A diferencia de lo que algunos puedan pensar, el testing no es algo que se hace, si hay tiempo, entre el final de las actividades de diseño y la entrega del producto. Los testing deberían estar presentes en todo el ciclo de vida del software, desde su diseño hasta el final de su mantenimiento, y durante toda la operación del software. Las rigurosas pruebas de software, incluyendo su documentación, permiten una reducción de la probabilidad de fallas durante la ejecución del software y contribuyen a mejorar la calidad [2].

Los testing de software se centran en dos aspectos complementarios pero distintos:

- La detección de fallas: que deberán ser resueltas para garantizar la calidad del producto entregado a los clientes y
- La toma de decisiones: basada en la información relacionada con el nivel de riesgo asociado con la entrega del software en el mercado, y con la eficiencia de los procesos de la organización que son la causa de las fallas detectadas.

Con respecto a los objetivos formulados para cada testeo depende de la fase del ciclo de vida del software y por lo tanto varían desde el diseño inicial hasta el mantenimiento:

- Durante el diseño general o fase de diseño detallado, los tests se centrarán en encontrar el mayor número de defectos (o fracasos), en el menor tiempo posible, con el fin de ofrecer software de alta calidad.

- Durante la fase de aceptación de los clientes, los tests demostrarán que el software funciona correctamente para obtener la aprobación del cliente.

- Durante las fases de operación, donde se está utilizando el software, los tests se centrará en asegurar que los niveles de exigencia (SLA: Acuerdo de nivel de servicio, explícita o implícita) están logrados.

- Durante el mantenimiento evolutivo y correctivo del software, los tests tienen como objetivo garantizar la ausencia de fallas y de efectos secundarios (regresión).

Un aspecto importante que debe decidir el equipo de desarrolladores es el modelo de desarrollo que guiará todo el ciclo de vida del producto de software. Hay muchos ciclos de desarrollo y los testing se pueden aplicar a todos estos modelos, dado que el testing se basa en la misma información de entrada que el desarrollo de software y proporciona los datos utilizados para mejorar la calidad del producto (software o sistema). En general, cualquier actividad de diseño puede presentar defectos, y debe estar asociado a uno o más actividades

performance cuando es usado bajo condiciones específicas.

encargadas de identificar y extraer estos defectos.
Los modelos de desarrollo de software se pueden agrupar en cuatro categorías principales:

- Modelos secuenciales: donde las actividades se ejecutan en secuencia, una después de la otra, con las etapas que permiten el cumplimiento de los objetivos. Por ejemplo: Cascada, modelo-V,
- Modelos de desarrollo iterativo: donde las actividades se ejecutan de forma iterativa hasta que se logra el nivel de calidad requerido. Estos modelos utilizan testing de regresión. Por ejemplo: Espiral
- Modelo incremental: puede combinar los dos modelos anteriores.
- Modelos ágiles: se basan en los principios expuestos en el "Manifiesto Ágil" [6] y se basan en aspectos del aprendizaje, la innovación y los cambios continuos para obtener mejores resultados. Estos métodos se basan en equipos multifuncionales que incluyen a desarrolladores, testeadores y clientes, facultados para tomar decisiones de diseño o ejecución. Propone numerosas y rápidas iteraciones, con retroalimentación de los usuarios, en lugar de entregas grandes y poco frecuentes. Por ejemplo: XP, Scrum, TDD (Test driven development), DDT (Design driven testing).

Es primordial, elegir metodologías de desarrollo que permitan construir aplicaciones en el menor tiempo posible, que cumplan con los requisitos de verificación y de validación del software. Esto es *verificar* que el software reúna las especificaciones establecidas y *validar* que lo diseñado haga lo que el cliente quiere [5].

Líneas de investigación y desarrollo

Los testing de software como una parte fundamental del proceso de desarrollo son esenciales para el éxito de la producción de software de alta calidad. No es de ninguna manera una tarea subordinada pero es intelectualmente desafiante.
Los testing están siempre presentes en el ciclo de desarrollo de software, a veces se lleva a cabo de manera diferente dependiendo del modelo, pero los principios básicos son siempre aplicables.
Es de profundo interés para este grupo de investigación trabajar con la metodología ágil Design-Driven Testing (en adelante DDT) que surgió como resultado de fusionar el análisis inicial y el diseño con una mentalidad ágil basado en tests. En muchos sentidos, es una revolución de la idea en la que se basa Test-Driven Development (en adelante TDD).
En cierto modo, el DDT es una respuesta a los problemas que se manifiestan con otras metodologías de testeos, tales como TDD, y también a problemas mucho más grandes que se producen cuando:

• no se hacen los tests (o no se escribe ningún test automatizado),

• se realizan algunos tests, pero sin rumbo y ad hoc y

• se hacen tests en exceso. Si los tests son contraproducentes o repetitivos, el tiempo extra dedicado a los tests podría ser tiempo perdido. La idea central de DDT es que los tests que se escriban sean los que necesitan ser comprobados, es decir, aquellos que están estrechamente vinculados a las necesidades del cliente.

Si bien DDT puede ser adaptado a cualquier proceso OOAD, fue diseñado originalmente para ser utilizado con el Proceso Iconix, un proceso OOAD ágil que utiliza un subconjunto básico de UML. La idea es que DDT proporcione información instantánea

validando cada paso en el proceso de análisis/diseño. [8]

Estructura del DDT

La Fig. 1 muestra los cuatro test principales: tests unitarios, tests de control, tests de escenarios y tests de requisitos. Como se puede ver, los tests unitarios son fundamentalmente el principio en el espacio de diseño/solución/aplicación. Están escritos y "apropiados" por los codificadores. Por encima de estos, los tests de control se intercalan entre el espacio del análisis y el diseño, y ayudan a proporcionar un puente entre los dos. Los tests de escenario pertenecen al espacio de análisis y son especificaciones de test manual con instrucciones paso a paso para los testeadores, que se expanden por todas las permutaciones de un caso de uso.

Por último, los tests de requisitos de negocio son casi siempre las especificaciones de test manual, que facilitan el "test humano" antes de liberar una nueva versión del producto.

Fig 1. Principales tests en DDT

Una vez, que todo el proceso esté trabajando es recomendable basar los tests de integración "end-to-end"[2] en las especificaciones de test de escenarios.

Proceso Iconix es un enfoque para "desarrollo *conducido* por *casos de usos",* que tiene por objetivo producir un diseño orientado a objetos que pueda servir de base para la codificación. Por lo tanto, es necesario vincular los escenarios a los objetos.

Resultados y Objetivos

Resultados Esperados

En teoría todos los aspectos de UML son potencialmente útiles, sin embargo en la práctica el tiempo nunca es suficiente para hacer el modelado, análisis y diseño. Proceso Iconix es un enfoque minimalista y estilizado que se centra en esa zona que se encuentra entre los casos de uso y el código.

Los casos de uso dan una forma estructurada a la captura de los requerimientos de comportamiento del sistema, de modo que se pueda crear un diseño a partir de ellos.

El objetivo de este grupo de trabajo en esta línea de investigación es aplicar y analizar el enfoque Iconix/DDT que en primera instancia permitirá separar las historias de usuario en los casos de uso que describen las interacciones del usuario con el sistema, de los requerimientos que especifican lo que el cliente quiere que haga el sistema. Se espera obtener un beneficio significativo del tratamiento de estos dos "tipos" de historias.

Formación de recursos humanos

Dirección a postulante a beca de iniciación a la investigación de la egresada Sabrina Cruz Introini, asesoramiento y dirección de tesis de licenciatura de los alumnos: Cecilia Marcuzzi, Silvina Balmaceda, Gerardo Jofré, Viviana Alferillo, Diego Checarelli.

Referencias

[1] P. Bourque y R. Dupuis, «Guide to the Software Engineering Body of Knowledge 2004 Version», *IEEE Press*, p. 191, 2004.

[2] H. B. Christensen, «*Flexible, Reliable Software: Using Patterns and Agile Development*.» CRC Press, 2011.

[2] "end-to-end," significa que el test verifica el escena completo desde la entrada inicial hasta concluir el escenario, pasando por cada interacciófn usuario – sistema.

[3] B. Homes, «*Fundamentals of Software Testing*.» John Wiley & Sons, 2013.

[4] T. A. Majchrzak, «*Improving Software Testing: Technical and Organizational Developments*.» Springer, 2012.

[5] S. Chavez, A. Martín, N. R. Rodríguez, M. A. Murazzo, y A. Valenzuela, «*Metodología AGIL para el desarrollo SaaS*», presentado en XIV Workshop de Investigadores en Ciencias de la Computación, 2012.

[6] Manifiesto Ágil http://agilemanifesto.org/iso/es/principles.html

[7] Matt Stephens, Doug Rosenberg. «*Design Driven Testing. Test Smarter, Not Harder.*» Apress, 2010.

[8] Doug Rosenberg, Matt Stephens. «*Use Case Driven Object Modeling with UML: Theory and Practice.*» Apress, 2007.

[9] Armando Fox, David Patterson. «*Engineering long-lasting Software.*» Copyright 2012 S trawberry Canyon LLC.

[10] Marten deinum, Koen Serneels «Spring MVC with Web Flow. *Capitulo 9*» Apress 2012.

[11] IEEE Computer Society, «IEEE Standard Glossary of Software Engineering Terminology. Std 610.12-1990», *IEEE Comput. Soc.*, n.º 1, p. 83, 1990.

[12] «Java EE at a Glance». [En línea]. Disponible en: http://tinyurl.com/j2ee2010. [Accedido: 17-nov-2010].

[13] «Microsoft .NET Framework». [En línea]. Disponible en: http://www.microsoft.com/net/.

[14] «Enterprise JavaBeans Technology». [En línea]. Disponible en: http://tinyurl.com/EJB3-0-2010. [Accedido: 17-nov-2010].

[15] R. S. Pressman, «Chapter 21: Object-Oriented Analysis», en *Software Engineering: a Practitioner's Approach*, 5Rev Ed., McGraw-Hill, 2000, pp. 571-572.

[16] C. Larman, «Chapter 6: Use-Case Model: Writing Requirements in Context», en *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 2.ª ed., Prentice Hall PTR, 2001, pp. 45-46.

[17] «Web services programming tips and tricks: Documenting a use case». [En línea]. Disponible en: http://www.ibm.com/developerworks/webservices/library/ws-tip-docusecase.html. [Accedido: 18-nov-2010].

[18] D. Kulak y E. Guiney, «Chapter 3: A Use-Case-Driven Approach to Requirements Gahtering. Requirements Specification Tools.», en *Use Cases: Requirements in Context*, 2.ª ed., Addison-Wesley Professional, 2003, pp. 53-55.

[19] J. Grudin, «Computer-supported cooperative work: history and focus», *Computer*, vol. 27, n.º 5, pp. 19-26, may 1994.

[20] B. Kitchenham y S. Charters, «Guidelines for performing Systematic Literature Reviews in Software Engineering», 2007.

[21] B. A. Kitchenham, D. Budgen, y O. Pearl Brereton, «Using mapping studies as the basis for further research - A participant-observer case study», *Inf Softw Technol*, vol. 53, n.º 6, pp. 638–651, jun. 2011.