Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 5, No. 4, diciembre, 2009

Web de la editorial: www.ati.es
Web de la revista: www.ati.es/reicis
E-mail: calidadsoft@ati.es
ISSN: 1885-4486

Publicado por la Asociación de Técnicos de Informática (ATI), Via Laietana, 46, 08003 Barcelona.

Secretaría de dirección: ATI Madrid, C/Padilla 66, 3º dcha., 28006 Madrid

Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS)

Contenidos

REICIS

Editorial

REICIS

Ya mencionábamos en el editorial del número anterior que la trayectoria de REICIS a través de los cinco volúmenes ya editados desde 2005 consolidada a través de su proceso de evaluación permite garantizar la calidad científica y técnica de los trabajos publicados. Además su inclusión en los índices de referencia como e-revistas, Redalyc o DOAJ avala la difusión y el reconocimiento de la revista y realza el valor que supone cada contribución para el avance de la innovación, la ingeniería y la calidad del software en toda la comunidad hispanohablante. Precisamente el interés de la comunidad internacional

En el año 2010, se plantean nuevos retos en el equipo editorial y directivo para consolidar el valor de REICIS tanto en el campo académico y científico como en la comunidad profesional, especialmente en su difusión a través de nuevos instrumentos que incrementen su impacto. También se continuará con la política de colaboración con eventos de prestigio así como la realización de, al menos, un número monográfico coordinador por un editor invitado.

Luis Fernández Sanz
Director

Presentación

REICIS

Este número de REICIS publica, tras el proceso de revisión de nuestro comité editorial, dos contribuciones extendidas y revisadas seleccionadas de entre las remitidas al taller PRIS'09 celebrado en San Sebastián el día 8 de septiembre de 2009.

En este caso, el primero de los trabajos publicados corresponde al titulado "Aplicación de Open HMI Tester como framework open-source para herramientas de pruebas de software" procedente de autores de las Universidades de Murcia. En este artículo, Pedro Luis Mateo, Gregorio Martínez y Diego Sevilla presentan una propuesta de aplicación de una arquitectura abierta para generar herramientas de ayuda para las pruebas de interfaces de usuario gráficas, incluyendo la posibilidad de generación automática de casos.

El segundo trabajo se titula "Comparativa práctica de las pruebas en entornos tradicionales y ágiles" y ha sido elaborado por Agustín Yagüe y Juan Garbajosa de la Universidad Politécnica de Madrid. Realiza una comparación entre los enfoques de prueba asociados a los entornos y metodologías tradicionales de desarrollo de software y los habitualmente asociados a los métodos ágiles, incluyendo también las herramientas que permiten mayor automatización y control de la gestión de pruebas.

Finalmente, en la columna de Actualidad Invitada, Javier Tuya que coordina el c grupo nacional de AENOR AEN/CTN71/SC7/GT 26 que actualmente trabaja para contribuir al desarrollo del futuro estándar ISO 29119 de pruebas de software. En su presentación, el profesor Tuya presenta la necesidad que existía de actualizar los estándares de pruebas de software así como las principales características de la nueva propuesta y su estado de desarrollo actual.

Luis Fernández Sanz

Aplicación de Open HMI Tester como framework open-source para herramientas de pruebas de software

Pedro Luis Mateo Navarro[1,2], Gregorio Martínez Pérez[1], Diego Sevilla Ruiz[2]
[1]Departamento de Ingeniería de la Información y las Comunicaciones
[2]Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30.071 Murcia, España
{pedromateo,gregorio,dsevilla}@um.es

Resumen

Las pruebas de software comprenden una fase muy importante del proceso de desarrollo. Este tipo de pruebas tienen como principal objetivo asegurar la calidad, fiabilidad y robustez de un software, dentro de un contexto o escenario donde está previsto que éste sea utilizado. Un subconjunto de estas pruebas corresponde con las que tienen como principal objetivo asegurar el correcto funcionamiento de las interfaces de usuario (o GUIs –*Graphical User Interfaces*–). Este tipo de pruebas de GUI representan un paso crítico antes de que un software sea puesto en funcionamiento y aceptado por el usuario final. Este artículo describe los principales resultados obtenidos como fruto de una serie de investigaciones relacionadas con las pruebas de software y de GUIs, entre los que se encuentra el diseño e implementación de una arquitectura código-abierto utilizada como entorno para el desarrollo de herramientas automáticas de pruebas sobre GUIs.

Palabras clave: Framework para herramientas de test, tests sobre GUIs, generación automática de casos de prueba en GUIs, tests de usabilidad software.

Application of the Open HMI Tester as an Open-source Framework for Software Testing Tools

Abstract

Software testing is a very important phase in the software development process. These tests are performed to ensure the quality, reliability, and robustness of software within the execution context it is expected to be used. Some of these tests are focused on ensuring that the graphical user interfaces (GUIs) are working properly. GUI Testing represents a critical step before the software is deployed and accepted by the end user. This paper describes the main results obtained from our research work in the software testing and GUI testing areas. It also describes the design and implementation of an open source architecture used as a framework for developing automated GUI testing tools.

Key words: Framework for testing tools, GUI testing, GUI test case auto-generation, software usability testing.

Mateo, P.J., Martínez, G., Sevilla, D.,"Aplicación de Open HMI Tester como framework open-source para herramientas de pruebas de software", REICIS, vol. 5, no.4, 2009, pp.6-18. Recibido: 23-7-2009; revisado: 28-10-2009; aceptado: 19-11-2009

1. Introducción

Las interfaces gráficas de usuario (GUIs) representan un elemento fundamental y crítico de las aplicaciones de hoy en día, llegando a acaparar incluso hasta el 60% del código que se produce en un proyecto software. Por lo tanto, probar la funcionalidad de las GUIs se presenta como una tarea imprescindible para asegurar la calidad, fiabilidad, robustez y usabilidad del sistema completo.

Pese a que está demostrado que la utilización de herramientas avanzadas para las pruebas de GUI permite mejorar los resultados y ahorrar tiempo y recursos a las empresas de desarrollo, la integración de éstas en los desarrollos actuales no es tan frecuente como se podría esperar. El principal motivo es que la naturaleza propia de este tipo de herramientas no facilita su integración en el proceso de desarrollo, ya que suelen poseer características muy específicas del entorno de pruebas para el que inicialmente fueron desarrolladas. Una limitación adicional relacionada directamente con las interfaces gráficas de usuario es que, hoy en día, existe una gran cantidad de sistemas de ventanas que pueden ser empleados en los desarrollos software, por lo que es fundamental el diseño de herramientas adaptables o de propósito general. Todas estas limitaciones, junto con otras tantas, son las que convierten la investigación, el diseño y el desarrollo de herramientas de pruebas abiertas y multiplataforma en un desafío muy interesante.

Este trabajo incluye una breve descripción de la arquitectura Open HMI Tester (OHT) (sección 2) y de algunas aplicaciones reales de ésta en el área de pruebas del software. En la sección 3 se describe una herramienta de captura y reproducción automática de pruebas de software desarrollada sobre la arquitectura mencionada anteriormente. En la sección 4 se explica un sistema de generación automática de casos de prueba para interfaces de usuario. Adicionalmente, en la sección 5 se describen algunas aplicaciones de esta arquitectura en el ámbito de las pruebas de usabilidad. Finalmente, se incluye una sección de conclusiones sobre el trabajo expuesto y algunas referencias sobre el trabajo futuro.

2. Arquitectura Open HMI Tester

La arquitectura *Open-HMI Tester (OHT)* [1] se trata de una arquitectura abierta (no está ligada a ningún sistema operativo ni sistema de ventanas concretos) para pruebas de

interfaz gráfica. Esta arquitectura (figura 1) está compuesta por una serie de módulos que implementan la funcionalidad genérica, y que por lo tanto nunca cambian (representados en la figura por las cajas no coloreadas), y un conjunto de módulos que deben ser adaptados (representados en la figura por las cajas coloreadas) con el fin de dotar a la arquitectura de la funcionalidad necesaria para poder operar sobre un entorno de pruebas (sistema operativo, sistema de ventanas, etc.) concreto.

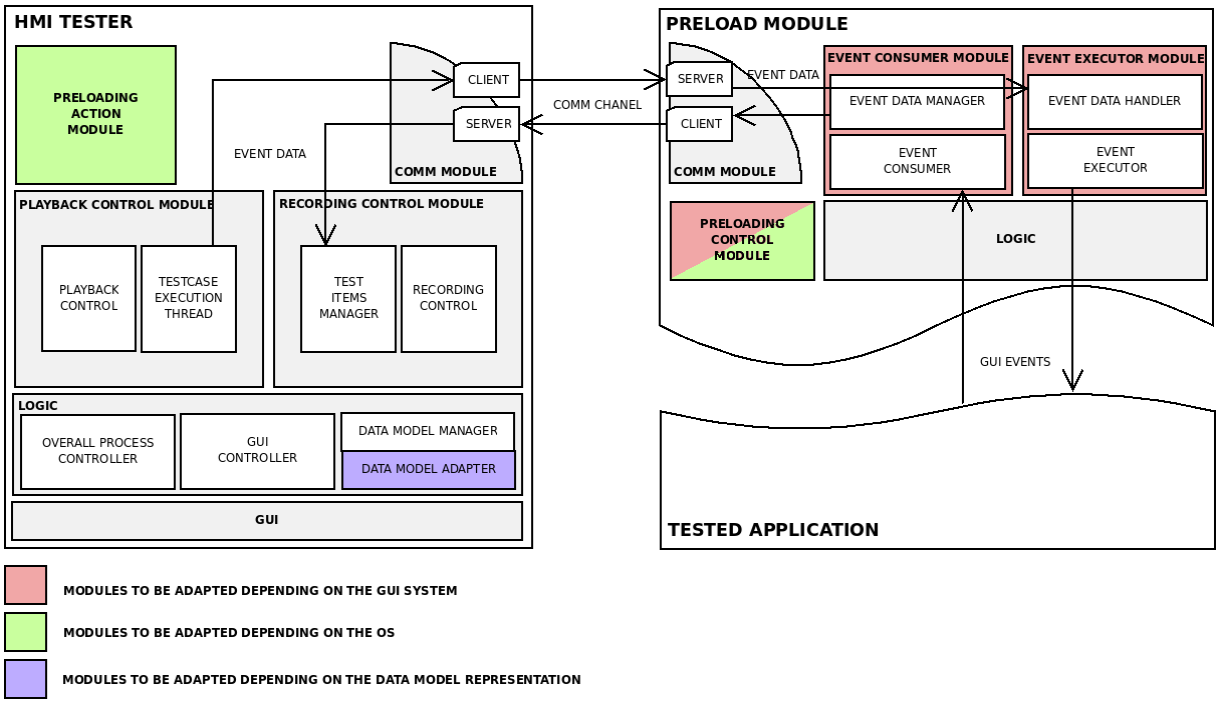

Figura 1: Arquitectura del HMI Tester y del Preload Module.

La arquitectura OHT se divide en dos módulos principales: el primero de ellos es el módulo HMI Tester, el cual se encarga principalmente del control de los procesos; el segundo de ellos corresponde con el módulo Preload, el cual es inyectado en la aplicación testada (en tiempo de ejecución) con el fin de ofrecer servicios de extracción y ejecución de eventos. Ambos módulos tienen la capacidad de comunicarse entre sí, lo que permite a la parte controladora llevar a cabo un proceso de introspección sobre la aplicación testada con dos fines principales: por una parte extraer la información correspondiente a los eventos generados a causa de las acciones que realiza el operador sobre la aplicación a probar; por otra, poder enviar nuevos eventos al núcleo de la aplicación objetivo con el fin de simular, de la manera más realista posible, las acciones del operador sobre la interfaz gráfica de la

aplicación testada. El hecho de que el proceso de introspección se realice mediante la inyección de una librería dinámica en la aplicación testada [2] evita la necesidad de ejecutar ningún tipo de código o de llamada en ésta, lo que permite que las herramientas desarrolladas bajo esta arquitectura puedan ser usadas tanto en aplicaciones en fase de desarrollo como en aplicaciones ya desarrolladas.

El módulo **HMI Tester** tiene como principal cometido el controlar los procesos de grabación (captura de eventos) y de reproducción (ejecución de eventos), así como la gestión completa de la creación y el mantenimiento de los archivos de pruebas (test suites). Como podemos observar en la figura 1, el módulo HMI Tester contiene una serie de submódulos que deben ser adaptados:

- **Data Model Manager and Adapter submodules**: estos submódulos permiten integrar en la arquitectura *Open HMI Tester* la implementación de cualquier representación del modelo de datos. Por lo tanto, deberán ser adaptados dependiendo del modelo de datos seleccionado para representar el conjunto de pruebas (test suite).
- **Preloading Action module**: el principal objetivo de este submódulo es llevar a cabo el proceso de precarga, permitiendo la inyección del módulo *Preload* al lanzar la aplicación a testar. Ya que el proceso de lanzamiento y control de aplicaciones puede incluir acciones concretas del sistema operativo, este módulo deberá ser adaptado dependiendo de las características del entorno de pruebas.

El resto de submódulos incluyen la funcionalidad genérica, y que por lo tanto se reutilizará independientemente de las características del entorno de pruebas en el que estemos trabajando; esta funcionalidad se encarga principalmente de manejar la interfaz de usuario (la correspondiente a la herramienta de pruebas) y las decisiones que se toman en ella, el sistema de comunicaciones hacia el módulo Preload, y el control de los procesos de grabación y reproducción de casos de prueba.

El módulo Preload será inyectado en la aplicación testada como una librería dinámica [2], con el fin de añadir la funcionalidad necesaria para llevar a cabo los procesos de extracción y ejecución de eventos. Su principal cometido será la captura de datos y eventos GUI, y también la ejecución de los eventos y órdenes recibidas desde el módulo

HMI Tester. Como podemos observar en la figura 1, este módulo también contiene una serie de submódulos que deben ser adaptados dependiendo de las características del entorno de pruebas:

- **Preloading Control module**: este submódulo será el encargado de desplegar todos los servicios necesarios para incorporar toda la funcionalidad del módulo Preload a la aplicación testada. Su funcionalidad deberá ser adaptada con el fin de asegurar que el conjunto de servicios necesarios para la introspección (comunicaciones, extracción de datos y ejecución de eventos y acciones) sean desplegados durante el proceso de inicialización de la aplicación testada.
- **Event Consumer module**: este submódulo se encarga de capturar y filtrar los eventos que se generan por las acciones del operador en la aplicación testada, gestionar los datos que éstos contienen y enviarlos hacia el HMI Tester para que sean tratados y almacenados. La adaptación de este submódulo dependerá del sistema de ventanas utilizado y de la jerarquía de eventos que utilice.
- **Event Executor module**: este submódulo ejecutará en la aplicación testada los eventos recibidos desde el módulo HMI Tester. La ejecución de estos eventos permitirá llevar a cabo una reproducción fiel de las acciones del operador que previamente fueron almacenadas, junto con cualquier acción adicional que pueda resultar necesaria para llevar a cabo el control del proceso de ejecución y/o validación. Los nuevos eventos recibidos serán notificados a través de un método de la interfaz. La implementación de este submódulo también dependerá del sistema de ventanas.

El resto de submódulos incluyen la funcionalidad genérica correspondiente a la lógica que gestiona los procesos en la aplicación testada, y la correspondiente al sistema de comunicaciones hacia el módulo HMI Tester.

Como también podemos apreciar en la figura 1, el proceso completo requiere la comunicación entre los dos módulos principales: HMI Tester y Preload. Esta comunicación se llevará a cabo mediante el establecimiento de un canal de datos (por ejemplo, sockets TCP) y el intercambio de elementos de información. Por otra parte, el módulo Preload también deberá establecer una comunicación con la aplicación testada a través de la captura

y envío de eventos, la cual puede ser completada con la ejecución de acciones mediante código.

3. Herramientas de captura y reproducción para GUI

Una de las principales aplicaciones de la arquitectura Open HMI Tester es la creación de herramientas de captura y reproducción para pruebas de interfaces gráficas de usuario. Este tipo de herramientas se centran en capturar la interacción de un operador con la aplicación (normalmente se almacena la secuencia de acciones que el operador ha realizado sobre la GUI), y volcarla a un fichero o soporte similar. Posteriormente, las secuencias de acciones pueden ser recuperadas y reproducidas sobre el software real tantas veces como sea necesario.

Actualmente existen herramientas parecidas que permiten grabar una secuencia de eventos y luego volver a reproducirla. Muchas de estas herramientas tienen la principal desventaja de que no capturan la interacción real del operador, sino que sólo son capaces de acceder a los eventos más externos de la GUI (es decir, clics de ratón y pulsaciones de teclado) y no al resto de los eventos que también forman parte de la ejecución de las acciones. Esta merma en la precisión de las acciones almacenadas puede provocar, por ejemplo, que un mínimo cambio en el entorno de pruebas (por ejemplo, mover la ventana principal de la aplicación 10 píxeles hacia un lado) convierta en inservible todo el esfuerzo realizado en un proceso anterior de grabación de casos de prueba. La baja tolerancia a modificaciones que presentan algunas de estas herramientas provoca que la fiabilidad y robustez de estos sistemas de pruebas quede en entredicho. En el lado opuesto tenemos las herramientas que al igual que ocurre en la arquitectura Open HMI Tester, pueden acceder al núcleo de la aplicación testada gracias a la introspección no intrusiva en código (mediante la inyección de librerías DLL) y por tanto tienen acceso sin problemas a toda la información correspondiente a cualquier evento generado en ella. Esta característica permite llevar a cabo una reproducción de secuencias de eventos en la que realmente se simula la interacción del operador sobre la aplicación, siendo ésta tolerante a ciertas modificaciones no críticas, como por ejemplo, un cambio en la localización de la ventana, la inclusión de nuevos elementos en la GUI, modificaciones en la localización de estos elementos, etc.

Este tipo de herramientas permite llevar a cabo un proceso de pruebas de interfaz gráfica donde el operador es el que decide qué partes de la GUI deben probarse, evitando así la creación de pesados modelos y de enormes baterías de casos de prueba. Esto nos permite solucionar el conocido problema del coverage criteria [3] (o problema del criterio de cobertura en su traducción al español), ya que la responsabilidad de decidir qué partes son testadas y cuáles no recaen completamente sobre el operador. Así, los casos de prueba generados se centrarán en los elementos relevantes y la funcionalidad para la que la GUI fue desarrollada, evitando secuencias de eventos y combinaciones de acciones sin sentido práctico.

Este tipo de herramientas también pueden incluir procesos adicionales (normalmente con el apoyo de herramientas de edición externas o incorporadas en la propia aplicación), como por ejemplo la validación automática mediante la incorporación de eventos especiales, la creación de guías del operador interactivas, o la generación de documentación asociada a las pruebas. Esto permite complementar el proceso de pruebas en base a las necesidades de cada proyecto.

En la figura 2 podemos apreciar una captura de pantalla correspondiente a un prototipo de herramienta de captura y reproducción desarrollado sobre la arquitectura Open HMI Tester. Esta herramienta, que puede ser descargada desde la plataforma Sourceforge en la dirección http://sourceforge.net/projects/openhmitester/, fue implementada con el objetivo de mostrar la funcionalidad y el enfoque aportado por la arquitectura Open HMI Tester. Este prototipo adapta la arquitectura OHT a un entorno de pruebas con las siguientes características:

- Sistema operativo Linux.
- Sistema de ventanas Trolltech Qt4 bajo X-Window.
- Representación del fichero de pruebas con XML.

Este prototipo, desarrollado en lenguaje C++, incluye la funcionalidad básica necesaria para llevar a cabo los procesos de captura y reproducción de eventos sobre aplicaciones desarrolladas bajo las características especificadas más arriba.

Figura 2: Pantalla de una Herramienta de Captura/reproducción desarrollada sobre la arquitectura OHT

4. Generación automática de casos de prueba y procesos de validación sobre GUI

Otra de las áreas en las que se está trabajando es la de la generación automática de pruebas para interfaces de usuario. La solución propuesta en [4] describe un enfoque intermedio entre las pruebas de software basadas en modelos y las que no utilizan ningún tipo de modelado, ya que la solución propuesta se centra principalmente en la creación de un modelo ligero a partir de un conjunto de anotaciones sobre los elementos representativos de la GUI.

El proceso correspondiente a este nuevo enfoque se basa principalmente en dos elementos: el primero de ellos es un conjunto inicial de casos de uso, que será utilizado para describir el comportamiento de la GUI; el otro elemento será un conjunto de anotaciones que describirán las posibles variaciones que pueden afectar a los diferentes elementos que componen la GUI (variaciones sobre los valores contenidos por esos elementos), y adicionalmente una serie de reglas de validación que comprobarán ciertas propiedades de esos elementos.

Como podemos apreciar en la figura 3, una vez hayamos definido un conjunto inicial de casos de uso que describa el funcionamiento de la GUI, el proceso continúa con las siguientes fases:

1. **Anotación de la GUI**: durante este proceso se van anotando los elementos más representativos de la GUI, es decir, los elementos a tener en cuenta en el conjunto de pruebas. Estos elementos pueden ser anotados de dos formas totalmente complementarias: la primera de ellas consiste en indicar un conjunto o rango de posibles valores que un elemento puede tener (cada nuevo valor da lugar a un nuevo caso de prueba); la segunda consiste en comprobar cierto valor de alguna propiedad que deba ser validada (se introducirán nuevas reglas de validación para estos elementos). A la hora de anotar un elemento, el operador podrá escoger cualquiera de las dos alternativas, o las dos en el caso de que sea necesario.

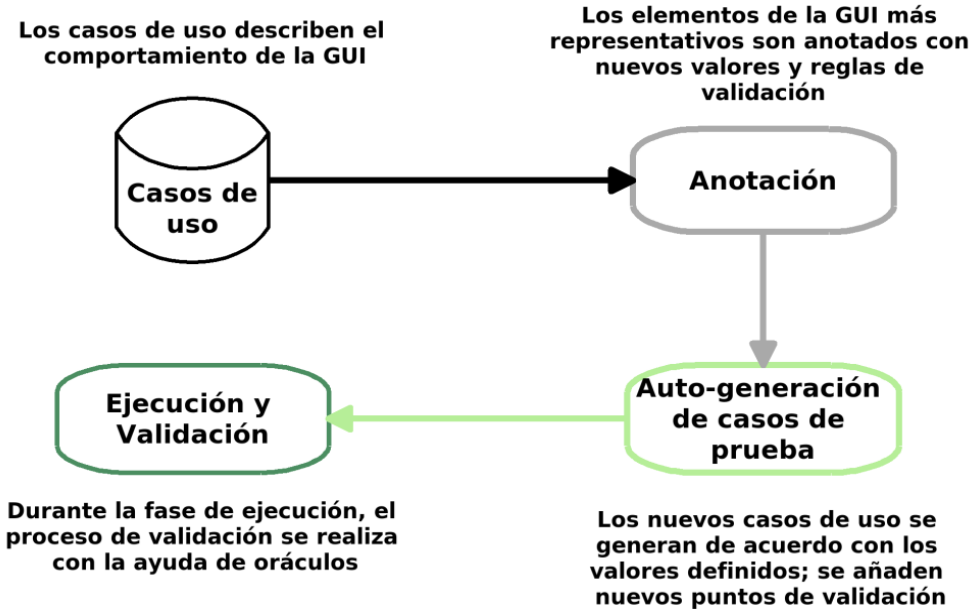

Figura 3: Proceso completo de auto-generación de casos de prueba basado en anotaciones

2. **Generación automática de casos de prueba**: durante esta fase se generará un nuevo caso de prueba para cada posible combinación de valores, es decir, se generarán todos los casos de prueba necesarios para cubrir todo el espectro de valores incluidos en las anotaciones y todas sus combinaciones. También se añadirán nuevos puntos de validación para satisfacer las reglas especificadas en la fase anotación. La figura 4 muestra un ejemplo muy sencillo en el que en la parte izquierda se muestran una serie de anotaciones sobre dos elementos de la GUI, y en la parte derecha se incluye el conjunto equivalente de casos de uso generados. Como podemos apreciar, se han generado cuatro casos de uso para cubrir todas

las posibles combinaciones de los 2x2 valores definidos. Además, se ha añadido un punto de validación para el segundo elemento anotado para poder comprobar las reglas definidas en las anotaciones.

3. **Ejecución y validación**: en este proceso los casos de prueba generados en el paso anterior son ejecutados uno a uno; al mismo tiempo se llevará a cabo el proceso de validación mediante el uso de *test oracles* [5]). Finalmente el proceso devuelve un informe completo en el que se incluirá el resultado, satisfactorio o no, de la ejecución de cada uno de los casos de prueba, y en el que también se incluirá el resultado de cada una de las validaciones.

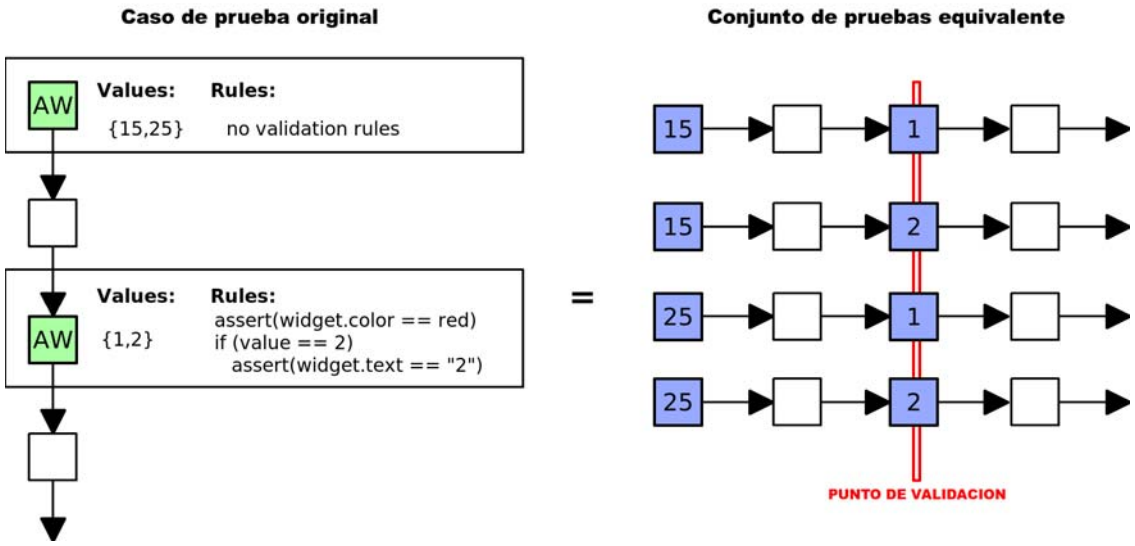

Figura 4: Ejemplo de generación automática de pruebas a partir de anotaciones

Como se ha comentado antes, en la tercera fase se llevará a cabo un proceso de validación basado en *test oracles*, u oráculos de pruebas en su traducción al español. Estos elementos se encargarán de, a partir de la información ofrecida por terceras partes (por ejemplo, una serie de adaptadores para los elementos de la GUI), llevar a cabo el proceso de validación para el que han sido diseñados. Más concretamente, se contempla la posibilidad de incorporar tres tipos diferentes de oráculos: de estado (validan que el estado actual de un elemento sea el mismo que uno previamente almacenado), de validación (validan un conjunto de reglas especificada previamente por el operador), y de *crash* o caída de la aplicación (comprueban si la aplicación ha caído por completo durante la ejecución del caso de prueba).

Esta solución al completo evita el proceso costoso de tener que crear un modelo complejo de la GUI y gran parte de las acciones inherentes a su creación, tales como la verificación, corrección y mantenimiento del modelo, que en la mayoría de los casos implican la intervención manual de las personas que componen el equipo de pruebas. También destacar la solución propuesta al problema del criterio de cobertura, ya que en este caso se evita la inclusión en los casos de prueba de los elementos y propiedades no relevantes mediante la anotación de los elementos más representativos de la GUI. En este caso se puede decir que el criterio de cobertura está dirigido por el usuario u operador. En el lado opuesto también es necesario recordar que ya que el criterio de cobertura queda en manos del operador, y con el objetivo de asegurar un proceso de pruebas completo, éste tiene la obligación de realizar las anotaciones necesarias y suficientes para cubrir todo el funcionamiento y espectro de posibles datos de entrada de la GUI.

Este método nos permite llevar a cabo un proceso de pruebas más ágil, con una mayor escalabilidad y mejor tolerancia a modificaciones respecto a los enfoques centrados en crear un modelo completo; esto se traduce en desarrollos más rápidos, ya que la productividad de los activos aumenta. Este enfoque también permite llevar a cabo un proceso de pruebas iterativo, sobre todo gracias a la posibilidad de reutilizar los casos de uso y las anotaciones. El tamaño del fichero de pruebas puede ir creciendo a la vez que el desarrollo avanza y las pruebas necesitan de un mayor refinamiento.

5. Evaluación de la Usabilidad en GUIs

Recientemente, los esfuerzos sobre la arquitectura *Open HMI Tester* están siendo trasladados hacia otro tipo de pruebas software, más enfocadas hacia la evaluación de la usabilidad de las interfaces gráficas de usuario. De acuerdo a la norma ISO 9241 [6], la usabilidad puede ser vista como “la medida con la que un producto puede ser utilizado por ciertos usuarios con el fin de conseguir sus objetivos con efectividad, eficiencia y satisfacción, en un contexto de uso específico”. Una de las principales motivaciones es que, hasta hace unos años, las investigaciones sobre la evaluación de la usabilidad en el software han estado algo abandonadas, abarcando casi todo el protagonismo la evaluación de la usabilidad en sitios web.

La idea principal es utilizar todo el potencial de la arquitectura *Open HMI Tester* (principalmente la introspección no intrusiva en código) para poder implementar herramientas que permitan automatizar los diferentes procesos de evaluación de la usabilidad. Sin embargo, se cree necesario dotar a la arquitectura de una mayor genericidad con el objetivo de no restringir el funcionamiento general a un proceso de grabación de eventos y a otro de reproducción. Este cambio permitiría al desarrollador añadir su propia lógica de operación, convirtiendo así a la arquitectura *Open HMI Tester* en una arquitectura con un propósito más general, y facilitando así la integración de este tipo de herramientas en el proceso de desarrollo [7].

El hecho de que la lógica de aplicación pueda ser adaptada permite encapsular el conocimiento de los expertos en usabilidad en la propia herramienta de pruebas, facilitando así la automatización completa de los procesos de evaluación. Al mismo tiempo, esta característica facilita la integración de los usuarios finales en el proceso de desarrollo, ya que evita la coincidencia de éstos y los expertos durante la fase de análisis y evaluación.

6. Conclusiones y trabajos futuros

En este trabajo se han presentado algunas de las principales líneas de investigación y desarrollo existentes en la actualidad en relación con los sistemas de pruebas para interfaces gráficas. Se ha presentado la arquitectura *Open HMI Tester*, sobre la que se han diseñado y desarrollado algunas herramientas destinadas a la realización de pruebas automáticas sobre GUIs. Algunos de los ejemplos propuestos son las herramientas de captura y reproducción y la generación automática de casos de prueba basada en anotaciones de los elementos de la GUI. También se ha descrito un nuevo enfoque para generalizar la arquitectura con el objetivo de aumentar el número de submódulos adaptables del *OHT*, donde destaca la posibilidad de adaptar la propia lógica de la herramienta. Con este nuevo enfoque más general se pretende ampliar el abanico de utilidades que pueden desarrollarse sobre esta arquitectura, como por ejemplo herramientas para la evaluación de la usabilidad de un software.

Actualmente se está diseñando una nueva arquitectura más general que permita implementar sobre ella un número mayor de herramientas para las pruebas de software. También se está trabajando para poder hacer realidad algunas de las cuestiones que se

presentan en este trabajo, y que hasta ahora son simples ideas en proceso de maduración, como por ejemplo el desarrollo de una serie de prototipos de herramientas para la evaluación de la usabilidad en las aplicaciones software. Por último, también se está trabajando en otros usos paralelos de la arquitectura, como por ejemplo la posibilidad de extender las pruebas de GUI hacia la lógica de aplicación, o utilizar la arquitectura *OHT* como un elemento independiente para la validación de propiedades y datos de entrada, siguiendo así la filosofía de la programación orientada a aspectos.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Cátedra SAES de la Universidad de Murcia. Este acuerdo engloba un esfuerzo conjunto entre la empresa SAES (Sociedad Anónima de Electrónica Submarina, http://www.electronica-submarina.com/) y la Universidad de Murcia, para trabajar en software de código abierto y en sistemas de tiempo real y aplicaciones críticas.

Referencias

[1] Mateo Navarro, P., Martínez Pérez, G., Sevilla Ruiz, D., "OpenHMI-Tester: An Open and Cross-Platform Architecture for GUI Testing and Certification", International Journal of Computer Systems Science and Engineering (IJCSSE), Special Issue on Open Source Certification, in press.

[2] Nasika, R., Dasgupta, P., "Transparent Migration of Distributed Communicating Processes", *13th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS)*, Las Vegas (Nevada, USA), November 2000.

[3] Memon, A., Soffa, M., Pollack, M., "Coverage Criteria for GUI Testing", *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 256-267, New York (USA), 2001.

[4] Mateo Navarro, P., Sevilla Ruiz, D., Martínez Pérez, G., "Automated GUI Testing Validation guided by Annotated Use Cases", *Informatik 2009: Model-based Testing (MoTes09) - 4th Workshop in conjunction with the annual national conference of German Assoc. for Informatics (GI)*, Lübeck (Germany), September 2009.

[5] Xie, Q., Memon, A.M., "Designing and Comparing Automated Test Oracles for GUI-Based Software Applications", *ACM Transactions on Software Engineering and Methodology*, 16, 1, Article 4, 2007.

[6] International Organization for Standardization, "ISO 9241-11 – Guidance on usability", 1998.

[7] Ferré, X., Juristo, N., "How to Integrate Usability into the Software Development Process", *ACM: 28th International Conference on Software Engineering (ICSE 2006)*, Shanghai, China, May 2006.

Comparativa práctica de las pruebas en entornos tradicionales y ágiles

Agustin Yagüe y Juan Garbajosa
System and Software Technology Group (SYST)
E.U. Informatica, Univ.Politecnica de Madrid (UPM), Ctra. Valencia Km. 7, Madrid 28031
ayague@eui.upm.es, jgs@eui.upm.es

Resumen

Las pruebas han adquirido relevancia en el desarrollo de software. Sin embargo cómo y cuándo se aplican las técnicas de pruebas puede ser diferente dependiendo de la comunidad que las use, incluso aunque en ambas se usen las mismas técnicas. Para algunas comunidades las pruebas del software son un proceso en sí mismo, mientras que para otras es un actividad o una tarea más dentro del proceso de verificación y validación. Por otro lado, las metodologías ágiles están cambiando el paisaje del desarrollo. Cuando se aplican metodologías ágiles, se escribe código para superar las pruebas que, previamente, se han especificado. En este entorno, las pruebas pueden sustituir a la especificación de requisitos. Por lo tanto, los conceptos que subyacen a las pruebas son diferentes en ambos enfoques. En esta contribución se analizan las perspectivas convencionales y ágiles y se presentan algunas implicaciones desde el punto de vista de la ingeniería del software.

Palabras clave: Técnicas de pruebas, semántica de pruebas, metodologías convencionales, metodologías tradicionales, metodologías agiles.

Comparison in practice of software testing in conventional and agile approaches

Abstract

The relevance of tests in software development has grown up.. Nevertheless, how and when testing techniques are applied could be very different depending on the development community, even when different development communities use the same techniques. Software testing is a process in some communities, but sometimes, it is an activity or a task of the verification and validation process. Also, agile methodologies are changing the trend in software development. In agile methodologies, the source code of a program is written to pass a set of tests that have been defined in advance. In this scenario, tests are being used to substitute the software requirements specification as well. Therefore, the concepts underlying software testing are different in conventional and agile approaches. This contribution analyzes software testing from conventional and agile approaches and we present some findings from the software engineering perspective.

Keywords: Software testing techniques, testing approaches, conventional methodologies, traditional methodologies, agile methodologies.

Yagüe A., Garbajosa, J.,"Comparativa práctica de las pruebas en entornos tradicionales y ágiles", REICIS, vol. 5, no.4, 2009, pp.19-18.
Recibido: 23-7-2009; revisado: 28-10-2009; aceptado: 19-11-2009

1. Introducción

El término "prueba" se ha utilizado a lo largo de los años para referenciar diferentes conceptos: haciendo mención a técnicas para realizar pruebas (pruebas de caja blanca y de caja negra), dando nombre a diferentes actividades y objetivos en la forma de aplicar las pruebas (unitarias, integración, aceptación o sistema), presentando diferentes metodologías de desarrollo de software centrados en la realización de las pruebas (TDD – Test Driven Development, ATDD – Acceptance Test Driven Development, STDD – Story Test Driven Development) [20-21] o, incluso para dar soporte a nuevas metodologías relacionadas con los proyectos que tienen como objetivo realizar pruebas como TMAP [1].

Por otra parte, los organismos internacionales de normalización han documentado desde diferentes puntos de vista y en forma de múltiples estándares, las prácticas relacionadas con las pruebas; algunos de los cuales son [2-6]. En SWEBOK [7] las pruebas se presentan como una actividad que se desarrolla para evaluar la calidad de un producto y mejorarlo mediante la identificación de los defectos y los problemas. Este preámbulo muestra que las pruebas han sido ampliamente estudiadas y analizadas desde perspectivas muy dispares. De hecho, las pruebas son utilizadas por todas las comunidades de desarrollo de software y sistemas. Aun cuando las técnicas y enfoques son compartidas por las diferentes comunidades, es bastante corriente que las apliquen en diferentes fases del proceso de desarrollo, incluso en ámbitos distintos y mediante actores diferentes.

Las consideradas como metodologías convencionales consideran la ejecución de las pruebas como una actividad que se lleva a cabo una vez terminada la fase de codificación y que tiene como propósito la identificación de fallos tal como se describe en [2][3][4][5] y [17]. Esto no está en contradicción con que las pruebas puedan empezar a diseñarse desde las fases tempranas del proceso de desarrollo. Este entendimiento ha ido evolucionando y en la actualidad, las pruebas se consideran como una actividad integrada en todo el proceso de desarrollo.

Las metodologías ágiles han emergido como una reacción para superar algunos retos que la industria del software había identificado. Entre estos se encuentran los, impredecibles a veces, cambios en el mercado y una progresiva reducción del tiempo requerido para la comercialización [8]. En este sentido, las metodologías ágiles intentan

incrementar la calidad del producto y reducir el coste que se deriva de los cambios en los requisitos. Este objetivo se alcanza mediante la simplificación de los procesos relativos a los requisitos y las tareas de documentación. Para conseguirlo, se definen un conjunto de valores, principios y prácticas que, tomando las pruebas como punto central, promueven una rápida, flexible y continua comunicación entre los clientes y el equipo de desarrollo [9, 10].

En resumen, se puede considerar que las pruebas alcanzan objetivos diferentes: en las metodologías convencionales se utilizan principalmente como base de la verificación y la validación del producto desarrollado, mientras que en las metodologías ágiles se llegan a utilizar en sustitución de las especificaciones de requisitos y como guía para el desarrollo de software. Esto nos permite afirmar que las pruebas del software han ido evolucionado para poder desempeñar nuevos papeles en el desarrollo de software y sistemas.

En este artículo se analizan las pruebas del software desde ambas perspectivas, convencionales y ágiles con el objetivo de comparar el papel que desempeñan las pruebas en las metodologías ágiles y las convencionales, identificar similitudes y diferencias tanto en las técnicas, como en las estrategias y enfoques. Se muestra cómo pueden tener objetivos claramente diferentes. Este análisis se ha realizado en base a la información de los datos prácticos recogidos en múltiples encuestas en diferentes paises y en diferentes sectores. Las encuestas aparecen referenciadas al final del documento, además se ha realizado una encuesta a nivel del estado español1 sobre el nivel de implantación de las pruebas y que se encuentra en proceso de análisis.

El resto del artículo está organizado como sigue: La sección 2 presentará las técnicas básicas de pruebas. La sección 3 presentan las pruebas dentro de los enfoques convencionales. La sección 4 muestra las pruebas desde el punto de vista de las metodologías ágiles. Finalmente, la sección 5 presenta las principales conclusiones del artículo.

2. Técnicas básicas de pruebas

En esta sección se describen brevemente las técnicas de pruebas y su aplicación en ambos enfoques, convencional y ágil. Se han considerado dos categorías: caja blanca y caja negra

[1] La encuesta se realizó durante el 1er Agile Open Spain celebrado en Madrid los días 23 y 24 de octubre de 2009.

[7]. Por un lado, las técnicas de caja blanca (caminos básicos, control de flujo, control de datos o pruebas de ramificación) están basadas en estudiar el código fuente y se utilizan, principalmente, desde una perspectiva interna en el desarrollo de software. Como están basadas en el estado actual del código fuente, si se realizan cambios en la implementación, en la mayoría de los casos, también habrá que realizar cambios en los casos de pruebas. Este tipo de pruebas requieren una alta cualificación en el equipo de pruebas (o en su caso del de desarrollo), tanto para la identificación de los casos de pruebas como para su implementación. Incluso, aunque las pruebas de caja blanca se pueden aplicar en diferentes actividades de pruebas (unitarias, integración y sistema) fundamentalmente se aplican en el ámbito de las pruebas unitarias y a través de herramientas de ejecución automática de pruebas como por ejemplo Clover[2] o Cobertura[3] que permiten conocer la cobertura de código de las pruebas diseñadas.

No se han encontrado estudios, tanto nacionales como internacionales, que analicen en detalle la adopción práctica en los proyectos de desarrollo de cada una de las técnicas de caja blanca. En oposición a las pruebas de caja blanca, se encuentran las pruebas de caja negra (particiones de equivalencia, análisis de valores límite, pruebas transversales, etc.) tienen una visión externa del producto software y no están centradas en el código fuente. Estas pruebas están centradas en analizar la funcionalidad. Por lo tanto los casos de prueba se basan en las diferentes entradas que puede recibir el software y sus correspondientes valores de salida. Estas técnicas se pueden aplicar a cualquiera de los niveles de pruebas (unitarias, integración, aceptación) con diferentes niveles de abstracción en la definición de los casos de prueba.

Un aspecto interesante a resaltar es el perfil de los actores que llevan a cabo las pruebas del software en las metodologías ágiles y en las convencionales. En las metodologías ágiles son los ingenieros de desarrollo los encargados de ejecutar este tipo de pruebas mediante pruebas unitarias. Como estos ingenieros están trabajando a nivel de código fuente, conocen la estructura del software y, por lo tanto, pueden definir sin un mayor esfuerzo los casos de prueba adecuados para probar el código. En el caso de que se produzca un fallo, conocen las líneas de código que se ven afectadas por el caso de prueba

[2] http://www.atlassian.com/software/clover/ - Visitado en junio de 2009
[3] http://cobertura.sourceforge.net/ - Visitado en junio de 2009

que ha fallado y que ha revelado el problema. En las metodologías convencionales, puede ocurrir que el equipo de desarrollo defina y ejecute ciertas pruebas, pero son los ingenieros de pruebas los que definen los casos de pruebas mayoritariamente y el equipo de pruebas es el responsable de ejecutarlo.

Fundamentalmente, en las metodologías convencionales la pruebas de caja negra las lleva a cabo el equipo de pruebas o equipos de pruebas independientes4. Sin embargo, en las metodologías ágiles se presentan dos perspectivas: por un lado, los ingenieros de desarrollo que ejecutan las pruebas de caja negra a nivel de pruebas unitarias, puesto que conocen los tipos de datos de cada uno de los parámetros. Por otro lado, el resto del equipo, que aunque no está directamente involucrado en el desarrollo, también participa en el diseño y ejecución de las pruebas [15]. Aunque es cierto que estos aspectos pueden depender de los equipos de trabajo. No se puede considerar que existan unas técnicas que se utilicen más en unas metodologías que en otras y, por lo tanto, no se puede afirmar que las técnicas básicas tengan un papel diferente en las metodologías convencionales y en ágiles. En ambas metodologías, es muy importante la existencia de herramientas que, de forma automática, tomen medidas sobre la calidad del código o sobre el grado de cobertura que alcanzan los test diseñados (ya sea sobre pruebas de caja blanca o pruebas de caja negra).

Con respecto a las técnicas de pruebas, puede deducirse del estudio realizado por Fernández [29] sobre la aplicación de pruebas del software (aunque no se mencionan este tipo de pruebas de forma explícita) que existe un largo camino por recorrer en la planificación y el análisis de la cobertura de las pruebas, ya sean de caja blanca o de caja negra. Más del 50% de los encuestados diseñaban las pruebas justo antes de ejecutarlas o cuando ya estaba disponible el código.

3 Las pruebas en los enfoques convencionales

Analizando las pruebas desde el punto de vista de los enfoques convencionales, fundamentalmente se identifican cuatro actividades relacionadas: pruebas unitarias, pruebas de integración, pruebas de aceptación y pruebas de sistema [7]. En estas metodologías se presta especial atención a la elaboración de la especificación de requisitos software. Dicha especificación se refina dentro del proceso de identificación de los requisitos del software

[4] http://www.cdainfo.com/Down/5-Test/Status.pdf - Visitado en noviembre 2009

mediante iteraciones. En este enfoque, el proceso de desarrollo se dirige desde la especificación de requisitos hacia el código y posteriormente desde el código hasta las pruebas.

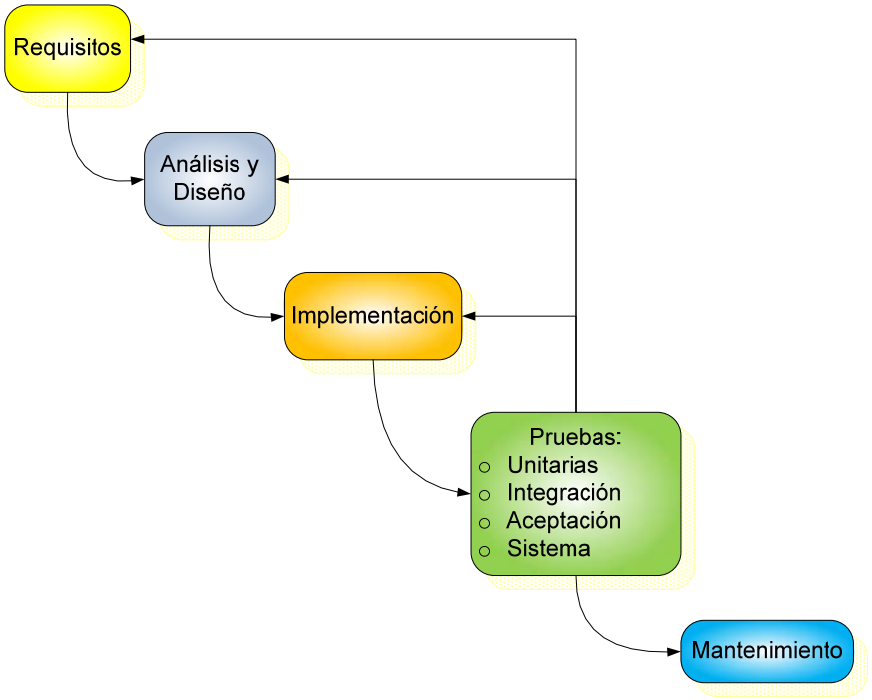

Figura 1. Enfoque tradicional de prueba.

En consecuencia, aunque en las metodologías convencionales las pruebas son importantes, éstas dependen directamente del resto de actividades del proceso de desarrollo. En [29] se comprueba que casi en el 60% de los proyectos, las pruebas se diseñan cuando ya está el código o cuando se comienza la fase de pruebas. Se puede considerar que en estos casos, el proceso de validación se entiende como un complemento del proceso de desarrollo.

La Figura. 1 muestra una visión general del modelo convencional de desarrollo donde todas las actividades se han aplicado en secuencia; esto quiere decir que las pruebas de integración no se pueden llevar a cabo hasta que no se han terminado todas la pruebas unitarias, lo que es extensible al resto de actividades. Por supuesto, este es un caso extremo de secuencialidad. Por otro lado, en [16] se puede encontrar un ejemplo de aplicación donde el proceso de pruebas no puede comenzar hasta que el proceso anterior en el tiempo no haya finalizado. Eso no quiere decir que la especificación de los procedimientos de pruebas no haya comenzado antes en el ciclo de vida. Mientras tanto, en el modelo en V

[17] y también en [7], la definición de los casos de prueba comienza en las primeras fases del modelo de procesos: las pruebas de aceptación y las pruebas de sistema se derivan del documento de especificación de requisitos, las pruebas de integración se derivan de la documentación del diseño y las pruebas unitarias de las de implementación de los módulos. Dentro de los enfoque convencionales y según se menciona en los resultados del estudio sobre las pruebas del software5 elaborado por el Quality Assurance Institute entre los asistentes a la International Conference on Software Testing, más del 70% de los proyectos desarrollados, disponen de procesos de pruebas documentados. Además, alrededor del 80% de los proyectos disponen de planes de pruebas detallados, casos de prueba, etc. Asimismo, en dicho informe se refleja que los resultados de las pruebas se utilizan para informar de los errores detectados y de los porcentajes de pruebas ejecutadas satisfactoriamente. Resulta bastante evidente que podemos esperar que justamente los asistentes a este congreso estén muy concienciados con el papel de las pruebas, por lo que estos resultados no son extrapolables a otras comunidades.

Como resumen, el enfoque de las pruebas en las metodologías convencionales se basa en la definición de los casos de prueba en función de unos requisitos previamente establecidos, mediante los lenguajes adecuados, por ejmeplo XUnit, FIT[17] o similares. Por lo tanto, el objetivo principal es comprobar que el desarrollo realizado cumple los requisitos definidos.

4. Las pruebas en las metodologías ágiles

Las metodologías ágiles en general, y particularmente Extreme Programming (XP) [19], son de alguna manera los responsables del aumento de popularidad de las pruebas del software. En XP las necesidades de los usuarios no se representan mediante documentos estandarizados de requisitos, sino que se representan mediante unos artefactos denominados “historias de usuario” que representan de una forma particular los requisitos del sistema. Cada historia de usuario, entre otras cosas, lleva asociada una lista de criterios de aceptación y, para cada criterio de aceptación, se define el conjunto de casos de prueba necesarios para validar el criterio de aceptación. Estas tareas de definición se realizan en las fases tempranas del proyecto y antes de que se comience la implementación del sistema.

[5] http://www.cdainfo.com/Down/5-Test/Status.pdf - Visitado en noviembre 2009

Con esto se consigue que las pruebas se utilicen para validar las necesidades de los usuarios y para dirigir la implementación. Se puede considerar que las historias de usuario y, por extensión, las pruebas asociadas a cada una juegan el papel de especificaciones del sistema.

En los enfoque ágiles las pruebas son el centro de la metodología y, por lo tanto son ellas las dirigen el proceso de desarrollo. Las metodologías ágiles plantean que el desarrollo no es un conjunto de fases en las que las pruebas son una fase más, sino que abogan porque las prácticas y el desarrollo estén completamente integradas, lo que puede llevar a modificar las estructuras organizativas de las empresas [15]. La irrupción de las metodologías ágiles, como se puede comprobar en [23-24], ha llevado a que cerca del 70% de las empresas ya estén incorporando algunas prácticas ágiles en su proceso de desarrollo. Según estos autores, esto ha traído como consecuencia una mejora de la calidad de los productos entregados en casi un 70% de los proyectos, como puede verse en la Figura. 2.

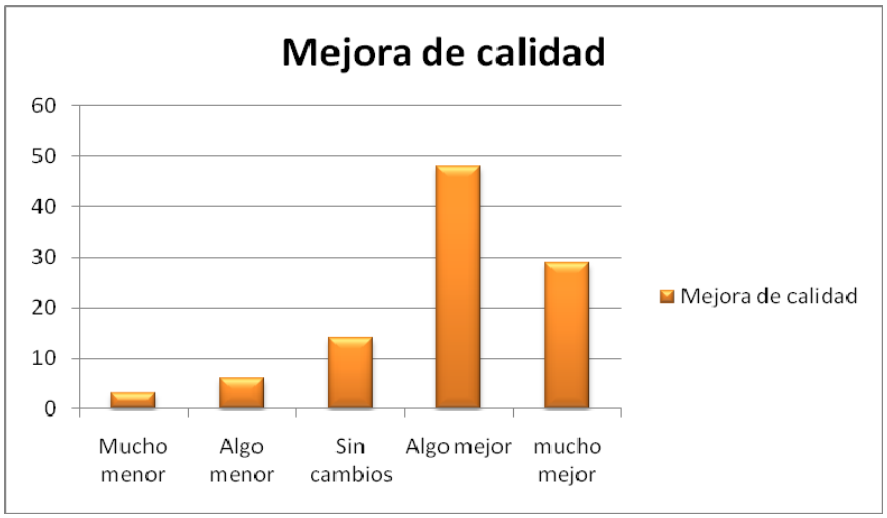

Figura. 2 Impacto de las metodologías ágiles en el aumento de calidad. Basado en [23]

Además de todo lo mencionado, es importante destacar que las metodologías ágiles no consideran las pruebas como un conjunto de niveles que haya que ir superando para alcanzar la validación final del sistema que se está desarrollando. Las metodologías ágiles presentan distintos enfoques del proceso de desarrollo que vienen determinados por los tipos de pruebas que se realizan.

En este artículo se van a considerar dos metodologías TDD [22] y ATDD [20]. Ambas metodologías, según se muestra en [25] son las más demandadas y aquellas que las

personas relacionadas con desarrollo ágil quieren adoptar (cerca del 50% de las personas encuestadas y que no estaban todavía aplicando este tipo de metodologías).

TDD y ATDD comparten el punto de vista desde el punto de vista del proceso de desarrollo pero utilizan diferentes herramientas de trabajo: las pruebas unitarias en TDD y las pruebas de aceptación en ATDD.

En el caso de TDD, cada requisito se representa como un artefacto denominado historia de usuario al que se le asocian sus correspondientes pruebas unitarias. Cuando se aplica TDD, en primer lugar define el repositorio de historias de usuario, conocido como "*Product backlog*", que representa los requisitos del sistema. Posteriormente, los desarrolladores escriben las pruebas unitarias necesarias para satisfacer cada una de las historias de usuario y, una vez escritas las pruebas, comienzan a implementar el código fuente necesario para superarlas. Por lo tanto, la lista completa de las historias de usuario y sus casos de prueba adoptan el mismo papel que la especificación de requisitos software en las metodologías convencionales y, las pruebas, son las que guían el desarrollo. Existen trabajos empíricos sobre el papel de las pruebas, en [28] se muestra que se empiezan a considerar las pruebas como mecanismo de especificación de requisitos (45%), sólo superados por documentos de texto (52%). Además, cuando se trata de la captura de especificaciones de diseño, se convierte en la práctica más utilizada (57%). El papel de las pruebas, como sustitutivo de los requisitos convencionales y según se presenta en [11-15], nos permite subsanar algunos problemas que tienen las metodologías ágiles en relación con la identificación de requisitos.

Un aspecto importante que provoca confusión es que no es suficiente escribir pruebas unitarias para que pueda considerar que se está aplicando TDD. Las pruebas unitarias podrían limitarse a probar una parte del código aislada del resto, mientras que TDD representa el proceso de desarrollo y tiene como objetivo comprobar que la aplicación funcionará correctamente. Por lo tanto en TDD las pruebas unitarias guían el proceso de desarrollo. Cuando se definen qué pruebas vamos a realizar en TDD no se tienen en cuenta decisiones de diseño mientras que cuando se escribe cada prueba unitaria, hay que tener en cuenta lo que hace el código implementado.

ATDD se basa en las pruebas de aceptación. Este enfoque toma como punto de referencia al usuario. El proceso es semejante a TDD pero guiado por las pruebas de

aceptación. Como se presenta en [28], TDD es el enfoque más aplicado para validación y pruebas, siendo su aplicación casi el doble que ATDD (71% y 40%).

Cuando se aumenta el nivel de abstracción y se aplica ATDD, cada historia de usuario tiene asociados un conjunto de criterios de aceptación. Posteriormente, para cada criterio de aceptación se definen las pruebas de aceptación que se deben superar. Una vez que las pruebas de aceptación están claramente definidas, el equipo de desarrollo comienza a escribir el código fuente que es necesario para superar los criterios de aceptación. Durante el proceso de desarrollo también será necesaria la definición de pruebas unitarias, asociadas al código que se implementa que garanticen que el código cumple con los requisitos. Como ocurre en TDD, la lista completa de historias de usuario (*product backlog*) junto con sus criterios de aceptación y las pruebas de aceptación desempeñan un papel equivalente a las especificaciones de requisitos software en las metodologías convencionales.

Como en el caso de TDD, las pruebas de aceptación no son ATDD. Las pruebas de aceptación tienen como objetivo probar la funcionalidad del sistema mientras que ATDD representa el proceso de desarrollo y tiene como objetivo que el sistema se construya de acuerdo a la funcionalidad determinada por el usuario. ATDD se lleva a cabo en colaboración con el usuario y en un lenguaje que usuario pueda entender. Para ello, las pruebas de aceptación se realizan en un lenguaje que sea entendible por parte del usuario; sirvan como ejemplo FIT o Easyaccept [18, 21].

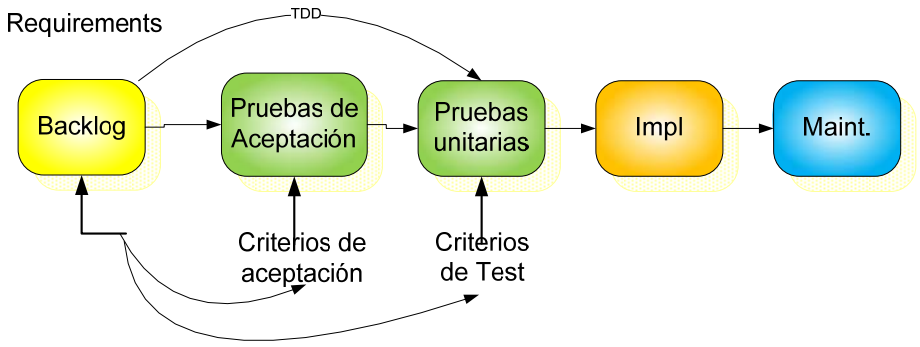

Figura. 3. Enfoque de ágil de pruebas.

La Figura 3 muestra la principal estrategia seguida en las metodologías ágiles. En este caso, las metodologías ágiles utilizan las pruebas unitarias y de aceptación como principales herramientas para dar soporte a los enfoques de pruebas. Considerando los

hábitos y su aplicación metodológica, hace que tanto TDD como ATDD se encuentren como una de las prácticas más adecuadas en cada sesión [25].

Debe realizarse una mención aparte sobre las pruebas de integración y las pruebas de sistema. Ambas se aplican de forma implícita en el proceso de desarrollo, es decir, cada nueva parte del código se escribe y se integra con el sistema completo, por lo tanto, no es necesario escribir pruebas específicas de integración sino que éstas se encuentran incorporadas en el resto de pruebas que se han desarrollado.

Para poder dar soporte a estos enfoques de pruebas es necesario disponer de una mínima infraestructura que permita automatizar la ejecución del elevado número de pruebas que se diseñan y la toma de medidas que permitan conocer el nivel de calidad que está alcanzando el desarrollo. Esta infraestructura mínima recibe el nombre de entorno de integración continua. Los entornos de integración continua, si bien son aplicables tanto en enfoques convencionales como ágiles, han tomado especial relevancia en éstos últimos. Estos entornos se caracterizan por la integración de múltiples herramientas que dan soporte al respaldo automatizado del código, su compilación, la ejecución de las pruebas, la toma de medidas de calidad, la generación de documentación y el despliegue de la aplicación. Dado que el proceso de ejecución de las pruebas está automatizado, cada vez que se realiza la integración de una nueva funcionalidad al sistema desarrollado, se llevan a cabo pruebas de regresión que aseguren que no se ha introducido ningún error nuevo en el sistema.

5 Conclusión

Como ya se ha justificado previamente, las pruebas desempeñan un papel importante en los diferentes modelos de procesos que van desde los modelos convencionales a los ágiles. Este artículo ha discutido cómo el papel de las pruebas es diferente en las metodologías convencionales y ágiles. Mientras en las metodologías convencionales, las pruebas, formalmente hablando, se ejecutarán mayoritariamente una vez que el código esté terminado, aunque puedan diseñarse antes de la codificación, en las ágiles las pruebas se escriben antes comenzar la codificación y el código que se escribe es el exclusivamente necesario para superar las pruebas y guían el proceso de desarrollo. El hecho de que las pruebas se realicen tan pronto como sea posible, está alineado con la reducción del impacto del coste de la detección de errores en las fases de pruebas.

En las metodologías ágiles, no se distingue de forma específica las pruebas de integración sino que éstas simplemente se incorporan al conjunto de las pruebas que se definen ya sea en la aplicación de TDD o de ATDD.

Desde el punto de vista de las técnicas básicas de pruebas, ambas metodologías aplican las mismas técnicas básicas de pruebas: caja blanca y caja negra. Los actores y el papel que desempeñan las técnicas de pruebas son algo diferentes. Aunque las prácticas puedan parecer semejantes a nivel práctico, no lo son a nivel semántico.

En lo referente a automatización, para lograr la integración en cada una de las iteraciones, se aplican entornos de integración continua. Estos entornos además permiten la realización automática de las pruebas de regresión del sistema desarrollado.

Finalmente, este papel cambiante de las pruebas también tiene una fuerte implicación en la definición general del proceso de desarrollo y que debe ser tenido en cuenta por los estándares de desarrollo.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Educación y Ciencia a través del proyecto OVAL/PM y por el Ministerio de Ciencia y Tecnología en el marco del proyecto FLEXI FIT-340005-2007-37 (ITEA2 6022).

Referencias

[1] Koomen, T., van der Aalst, L., Broekman, B., Vroon, M.: *TMap Next, for result driven testing*. UTN Publishers, 2006.

[2] ISO: *ISO/IEC 12207. Software Engineering - Life Cycle Processes*. ISO, 2008.

[3] ISO: *ISO/IEC 15288. Systems Engineering - System Life Cycle Processes*. ISO, 2008.

[4] IEEE: *IEEE Std 1008, Standard for Software Unit Testing. IEEE*, pub-IEEE-STD, 2002.

[5] BSI: *BS 7925-2:1998 - Software testing. Software component testing*. BSI, 1998.

[6] European Consortium for Space Standardisation, E.: *Software - part 1: Principles and requirements*. E-40, 2003.

[7] Abran, A., Moore, J.W., Bourque, P., Dupuis, R., Tripp, L.L.: *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE, 2004.

[8] Boehm, B.: *"A view of 20th and 21st century software engineering"*. *"Proceedings of the 28th Intl. Conf. on Software engineering*", ACM, pp. 12-29, 2006.

[9] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., et al.: *Manifesto for agile software development*, http://agilemanifesto.org/, 2001.

[10] Shore, J., Warden, S., *The Art of Agile Development*. O'Reilly Media, 2007.

[11] Paetsch, F., Eberlein, A., Maurer, F., *"Requirements engineering and agile software development*". En *Proceedings of the Twelfth InternationalWorkshop on Enabling Technologies*, 2003.

[12] Cao, L. y Ramesh, B., *"Agile requirements engineering practices: An empirical study"*, IEEE Software, vol. 25, nº 1, pp. 60-67, 2008.

[13] Eberlein, A. y Leite, J., *"Agile requirements definition: A view from requirements engineering"*. En *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02*), pp. 4-8, 2002,

[14] Dyba, T. y Dingsoyr, T.: *"Empirical studies of agile software development: A systematic review*", *Information and Software Technology*, vol.50, nº9-10, pp. 833-859, 2008.

[15] Talby, D., Hazzan, O., Dubinsky, Y. y Keren, A. *"Agile Software Testing in a Large-Scale Project"*, IEEE Software, vol. 23, nº 4, pp. 30-37, 2006.

[16] Royce, W.W., *"Managing the development of large software systems: concepts and techniques"*. En: *Proceedings of the 9th international conference on Software Engineering*, pp. 328-338, 1987.

[17] A. Bröhl y W. Dröschl, *Das V-Model*, Oldenbourg Verlag, 1995

[18] Cunningham, W., *FIT: Framework for integrated acceptance testing*. http://fit.c2.com, consultado en junio de 2009.

[19] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison Wesley Professional, 2004.

[20] Koskela, L., *Test Driven: TDD and Acceptance TDD for Java Developers*, Manning Publications, 2007.

[21] Sauve, J.P., Neto, O.L.A. y Cirne, W., "Easyaccept: a tool to easily create, run and drive development with automated acceptance tests". En: *Proceedings of the 2006 international workshop on Automation of software test*, 2006, pp. 111-117

[22] Janzen, D. y Saiedian, H., "Test-driven development: Concepts, taxonomy, and future direction", Computer, vol. 38, nº 9, 2005, pp. 43-50.

[23] Ambler, S., *Agile Adoption Survey 2008,* www.ambysoft.com/surveys/agileFebruary2008.html, consultado en junio de 2009.

[24] Versionone, *The State of Agile Development, 3rd Annual Survey: 2008.* www.versionone.com/AgileSurvey/, consultado en junio de 2009.

[25] Ambler, S. y Vizdos, M., *Agile Practices Survey 2009,* www.ambysoft.com/surveys/practices2009.html, consultado en junio de 2009.

[26] Ambler, S. y Vizdos, M., *Agile Practices and Principles Survey 2008,* www.ambysoft.com/surveys/practicesPrinciples2008.html, consultado en junio de 2009.

[27] Ambler, S. y Vizdos, M., *Agile Project Initiation 2008,* www.ambysoft.com/surveys/projectInitiation2009.html, consultado en junio de 2009.

[28] Ambler, S., *Test-Driven Devlopment Survey 2008,* www.ambysoft.com/surveys/tdd2008.html, consultado en junio de 2009.

[29] Fernández, L. "Un sondeo sobre la práctica actual de pruebas de software en España", *REICIS, Revista Española de Innovación, Calidad e Ingeniería del Software*, vol. 1, nº 2, 2005, pp. 41-54.

REICIS

ACTUALIDAD INVITADA

El futuro estándar ISO/IEC 29119 - Software Testing

Javier Tuya
Universidad de Oviedo, Departamento de Informática
http://www.di.uniovi.es/~tuya/
tuya@uniovi.es

Introducción

En el editorial del último número de la revista Software Testing, Verification and Reliability (diciembre de 2009) su editor, Jeff Offutt, presenta el conocido símil que compara la ingeniería del software con otras disciplinas de ingeniería, pero en esta ocasión aplicado a las pruebas de software. El editorial, titulado "Testing my new building" concluye con una interesante reflexión: la ingeniería del software no será considerada un campo maduro hasta que los mejores programadores y diseñadores esperen ser promocionados a puestos específicos de pruebas.

Algo similar se podría indicar sobre los estándares de ingeniería del software que contribuyen a la madurez de esta profesión, de los que ya existe una amplia variedad publicados por diversas organizaciones. Sin embargo, en lo relacionado con las pruebas de software, los estándares existentes actualmente o bien son parciales (como los de IEEE y BSI) o bien son particulares de sectores específicos y muy regulados (como los relacionados con aviación o salud). Este vacío es el que se pretende cubrir con el estándar ISO/IEC 29119 Software Testing, actualmente en elaboración.

Este estándar comenzó su andadura en 2007, año en el que ISO aprobó la constitución del grupo de trabajo 26 (WG26) dentro del subcomité ISO/IEC JTC1/SC7 "Software and Systems Enginering". Posteriormente, AENOR constituyó el correspondiente grupo de trabajo AEN/CTN 71/SC7/GT26 - Pruebas de Software (GT26) para colaborar en la elaboración del estándar.

La estructura de ISO/IEC 29119 consta de cuatro partes:

1. Conceptos y Vocabulario
2. Proceso de Pruebas
3. Documentación de Pruebas
4. Técnicas de Prueba

Las partes 2 (procesos) y 3 (documentación) han sido planificadas en una primera secuencia, que tras sucesivas revisiones por expertos externos e internos a los diferentes comités producirán los primeros borradores del comité (*Committee Draft*: CD) para continuar con los procesos de tramitación y revisión habituales en la elaboración de los estándares ISO.

El modelo de procesos

En particular, el modelo de procesos tal y como figura en el último borrador de trabajo está formado por tres niveles:

1. Procesos de la organización
2. Procesos de gestión
3. Procesos “fundamentales”

En el nivel superior se encuentran los procesos de la organización, que no son específicos de un determinado proyecto de pruebas. Permiten definir las políticas y estrategias aplicables a toda la organización o a una línea de proyectos.

Para un proyecto de pruebas se tienen los niveles de procesos de gestión y procesos “fundamentales”. Se define un conjunto de procesos de gestión genéricos para permitir flexibilidad y adaptación a diferentes contextos. Estos procesos son la planificación, monitorización y control, y finalización de las pruebas. Los diferentes procesos de gestión se podrán instanciar en uno o en varios dependiendo de la situación. Por ejemplo, en un proyecto simple puede existir solamente un plan de pruebas global (una sola instancia). En un proyecto más complejo puede existir un plan maestro y otros planes subordinados a éste para cubrir diferentes niveles de prueba (por ejemplo, pruebas de integración, sistema o aceptación) o diferentes tipos de prueba. Cada uno de ellos sería una instancia del proceso de gestión.

Los procesos fundamentales abarcan los aspectos técnicos de las pruebas: diseño e implementación, puesta a punto del entorno, ejecución de pruebas y la notificación de resultados de las pruebas. También se incluirán variantes de los procesos para contemplar tanto pruebas dinámicas como estáticas.

En la dirección http://in2test.lsi.uniovi.es/gt26/ se puede acceder a la información y a los enlaces relativos a la actividad de los grupos de trabajo, así como las últimas novedades y presentaciones relacionadas con este estándar.

Perfil profesional

Javier Tuya es Doctor por la Universidad de Oviedo y actualmente Profesor Titular de Universidad en el Depto. de Informática. Dirige el Grupo de Investigación en Ingeniería del Software (http://giis.uniovi.es/), especializado en pruebas de software y en particular en arquitecturas orientadas a servicios y aplicaciones con bases de datos.Es el coordinador de la Red RePRIS para la Promoción de las Pruebas en Ingeniería del Software (http://in2test.lsi.uniovi.es/repris/) y miembro de asociaciones profesionales como IEEE, IEEE/CS, ACM, Association for Software Testing y SISTEDES. Actualmente es miembro del workgroup de ISO JTC1/SC7/WG26 - Software Testing que elabora el nuevo estándar ISO/IEC 29119 y coordinador del correspondiente grupo nacional de AENOR AEN/CTN71/SC7/GT 26.

-