Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 5, No. 3, octubre, 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

El número de octubre de 2009 de REICIS se nutre, por una parte, de las contribuciones que remiten regularmente los autores a nuestro proceso editorial y, por otra, de la selección de trabajos presentados en eventos que colaboran con nuestra revista. En efecto, dos de las líneas claves de la actividad de REICIS se reflejan en este número como testimonio del compromiso del comité editorial con su filosofía de servicio a la difusión de la innovación y el conocimiento en el ámbito de la ingeniería y la calidad del software. De esta forma, pretendemos contribuir a los siguientes objetivos:

- Contribuir con una política de acceso abierto (no en vano estamos en DOAJ y e-revistas) a difundir las aportaciones de investigadores y profesionales innovadores españoles e iberoamericanos al avance de la disciplina, fomentando su valoración tanto por el política de revisión por expertos de los trabajos enviados como por su publicación en una revista incluida en repositorios nacionales e internacionales, con mayor valoración que los congresos nacionales en los baremos habituales para académicos e investigadores.
- Facilitar el acceso de profesionales, investigadores y académicos a contenidos especializados de calidad en acceso abierto en idioma español, favoreciendo que REICIS pueda considerarse referencia en este ámbito.
- Elevar la difusión de trabajos prometedores presentados en eventos que, en general, quedan lejos del alcance de la mayoría de los profesionales y de los interesados en general en la disciplina. Conseguimos así que se pueda aportar, a través de las versiones extendidas y nuevamente revisadas, de mayor número de detalles y de la matización de elementos de contenido, fruto del feedback de revisores y audiencia del propio evento y de las aportaciones de nuestro propio comité editorial.

En definitiva, creemos que la trayectoria de REICIS a través de los cinco volúmenes ya editados desde 2005 permitirá que la calidad científica y técnica de los trabajos publicados suponga una contribución para el avance de la innovación, la ingeniería y la calidad del software en toda la comunidad hispanohablante.

Luis Fernández Sanz
Juan J. Cuadrado-Gallego
Editores

Presentación

REICIS

Este número de REICIS publica, tras el proceso de revisión de nuestro comité editorial, dos contribuciones remitidas directamente por los autores a nuestro proceso regular de recepción de contribuciones.

El primero de los trabajos publicados corresponde a la contribución titulada “La gestión de riesgos en la producción de software y la formación de profesionales de la informática: experiencias de una universidad cubana” elaborado por Yeleny Zulueta, Eder Despaigne y Anaisa Hernández de la Universidad de Ciencias Informática de Cuba. El trabajo presenta un análisis de la disciplina de gestión de riesgos de proyectos de desarrollo para obtener conclusiones útiles en los programas formativos

El segundo trabajo se ha seleccionado, a través de una versión extendida y revisada para nuestra revista, entre los presentados al taller ATSE - Workshop on Automating Test Case Design, Selection and Evaluation celebrado dentro de la 4ª Conferencia Iberica de Sistemas y Tecnologias de la Informacion, en los días 17-20 de junio de 2009 en Povoa de Parzim en Portugal. ATI mantiene convenio de colaboración con la asociación AISTI organizadora de la Conferencia. El trabajo “” presentado por Pedro Reales y Macario Polo propone un método para la reducción de casos de pruebas para Junit manteniendo la cobertura de código y con soporte de un plug-in para Eclipse. A continuación, incluimos también la reseña, por parte de su organizadora, sobre el desarrollo del taller ATSE’09.

Finalmente, en la columna de Actualidad Invitada, contamos en esta ocasión con una contribución que aborda una de las grandes tendencias filosóficas actuales para el desarrollo de software: los métodos ágiles. José Ramón Díaz, miembro del grupo coordinador de Agile Spain, presenta un análisis del estado y los retos actuales del enfoque ágil de desarrollo de software.

Luis Fernández Sanz

La gestión de riesgos en la producción de software y la formación de profesionales de la informática: experiencias de una universidad cubana

Yeleny Zulueta, Eder Despaigne
Universidad de las Ciencias Informáticas
{yeleny, ederdh}@uci.cu
Anaisa Hernández
Instituto Superior Politécnico José Antonio Echeverría
anaisa@ceis.cujae.edu.cu

Resumen

La industria cubana del software está llamada a convertirse en una significativa fuente de ingresos nacional. Este reto demanda una sólida formación de los profesionales informáticos y exige a las universidades cubanas garantizar que los egresados dominen y apliquen consistentes y novedosas prácticas en el desarrollo de software. La gestión de riesgos constituye uno de los procesos medulares en el empeño de determinar continuamente qué puede ir mal en un proyecto informático: cuáles son los riesgos que lo afectan, sus posibles consecuencias y estrategias de respuesta. Su introducción orgánica en la formación y producción mejorará la preparación integral del profesional y la calidad de los procesos y productos de software. En este artículo se presentan principios a tener en cuenta para la incorporación de la gestión de riesgos en la formación de los profesionales de la informática, y se describe la experiencia de su aplicación en la Universidad de las Ciencias Informáticas como complemento y apoyo de la implementación del Modelo de Gestión de Riesgos para Proyectos de Desarrollo de Software (MoGeRi).

Palabras Claves: Gestión de Riesgos, Informática, Universidad, Formación.

Risk Management in software development and education of computing professionals: experiences in a Cuban University

Abstract

The Cuban Software Industry is called to become a significant source of national revenues. This challenge demands a solid education of computing professionals. Cuban universities should guarantee graduate people are proficient and capable to apply consistent and novel practice in software development. Risk management is one of the core processes to continuous control of whatever can go wrong in a software project: which are the risks that affect it, their possible consequences and possible contingency strategies. Organic inclusion in education and production processes will improve the professional's training and the quality of software processes and products. In this article principles to keep in mind for the introduction of the Risk Management discipline in the education of computing professionals are presented well as the experience of its application in the University of

Computing Sciences as complementary model and support for the implementation of the Model for Risks Management in Software Development Projects (MoGeRi) is described.

Key words: risk management, computer science, university, education.

Zulueta,Y., Despaigne,E. y Hernández A., "La gestión de riesgos en la producción de software y la formación de profesionales de la informática: experiencias de una universidad cubana", REICIS, vol. 5, no.3, 2009, pp.6-21. Recibido: 23-6-2008; revisado: 27-10-2008; aceptado: 23-9-2009

1. Introducción

La meta de transformar la informática en un campo próspero internacionalmente y en una de las ramas más productivas para Cuba, requiere inevitablemente lograr el respaldo de un sólido sistema de educación superior. Corresponde a las universidades cubanas, garantizar la formación de profesionales en la informática y la computación, listos para enfrentar la producción de software con responsabilidad y creatividad. La Universidad de las Ciencias Informáticas (UCI) debe alcanzar este propósito con la ejecución de ambiciosos programas curriculares y de producción y con la aplicación de las más modernas tecnologías en la docencia [1].

En la UCI, se renueva y afianza el modelo de formación-producción-investigación donde el estudiante puede ejercitar y comprobar en condiciones de trabajo reales, las habilidades adquiridas de forma curricular, logrando así completar el ciclo de enseñanza-aprendizaje y aplicar las mejores prácticas de producción de software basadas en modelos y estándares internacionales. Dentro de este grupo se sitúa la gestión de riesgos (GR) y precisamente en este artículo se presentan principios a tener en cuenta en su introducción en la docencia y la producción de las universidades y algunas experiencias asociadas a su implementación en la UCI.

Con esta propuesta se avanza hacia la preparación integral del profesional y hacia producciones de software con calidad que demanda hoy la sociedad cubana.

2. Fundamentos de la gestión de riesgos

Aunque se han producido amplios debates sobre la definición adecuada de *riesgo de software* y aun cuando los criterios son variados [2-8], hay acuerdo común en que el riesgo siempre implica dos dimensiones:

- Incertidumbre: El acontecimiento que caracteriza al riesgo puede, o no, ocurrir.

- Efecto en los objetivos: Si el riesgo se convierte en una realidad, esto tendrá consecuencias para el proyecto.

Es común utilizar los términos *probabilidad* e *impacto* para describir estas dos dimensiones, refiriéndose la *probabilidad* a la posibilidad de ocurrencia (la dimensión de incertidumbre), y el *impacto*, al alcance de lo que sucedería si el riesgo se materializa (la dimensión efecto). En cuanto al efecto en los objetivos, algunos autores solo consideran sus consecuencias negativas, mientras que otros [8-11], reflexionan sobre los beneficios u oportunidades que también puede entrañar.

Tanto en las definiciones como en las clasificaciones, los riesgos son analizados en la dimensión del producto, de los procesos y por supuesto, del proyecto; sin embargo, se subestima la relación directa de los riesgos en las personas, que son en definitiva quienes definen los procesos e integran los proyectos para desarrollar esos productos de software: ¿son las personas menos importantes?, ¿basta con inferir o pensar que la relación riesgo-persona puede quedar implícita en cualquier definición o tratarse como un recurso más? La respuesta a estas interrogantes es negativa en ambos casos para los autores de esta investigación, por lo que se propone que el riesgo sea razonado como *la medida de la probabilidad y la pérdida de un acontecimiento que afecta el proyecto, proceso o producto de software y/o a las personas que lo desarrollan.*

Diferentes conceptualizaciones de GR aparecen en la literatura [3, 4, 5, 8, 13, 14]. No existe la "definición perfecta" puesto que tanto ella como sus objetivos, estarán determinados en gran medida por la posición adoptada por cada autor en la visión del riesgo. Si se trabaja sobre la base del riesgo positivo, entonces las metas de GR no pueden estar solo circunscritas a limitar y/o evitar los daños, sino que estos fines deben ampliarse hacia la búsqueda de las vías para convertir estos riesgos en oportunidades, beneficios y efectos positivos.

La GR debe integrar de forma sistémica los procesos que se encargan tanto de planificar, identificar y analizar, como de responder al riesgo y seguir, controlar y comunicar las actividades planificadas al respecto.

3. La Gestión de Riesgos: una necesidad en el entorno docente y productivo de las universidades cubanas

A pesar del auge que en las últimas décadas ha tomado el tema y aunque las organizaciones muestren el uso de procesos formales de Gestión de Proyectos, diversos estudios [14-16] demuestran que la GR continúa siendo débil.

En las entrevistas y encuestas realizadas durante la investigación al personal involucrado en los proyectos de desarrollo de software en Cuba y en la UCI (gestores, ingenieros de software, clientes, estudiantes, profesores), se reconoce la carencia de conocimientos relacionados con la GR y por tanto de su aplicación. El 86% de los entrevistados considera que se conocen algunos riesgos que pueden afectar el desarrollo del proyecto, pero el 100% reconoce que no son debidamente identificados utilizando alguna guía formal. En la encuestas el 100% de los interpelados concede gran importancia a la GR para el cumplimiento de los objetivos del proyecto y considera necesario en consecuencia, la creación y aplicación de un modelo con este propósito en la UCI [17].

Aunque las actividades de GR son aplicadas en la producción de manera insuficiente y poco sistemáticas, sí son incluidas en los planes de estudio de las carreras afines de gran parte de las instituciones a nivel mundial [18-21], incluyendo América Latina; ya sea como asignatura optativa o troncal o como tema en asignaturas relacionadas con la planificación y gestión de proyectos informáticos.

4. MoGeRi: Un Modelo para la Gestión de Riesgos en proyectos de desarrollo de software

MoGeRi [17] surge tras la identificación de las características y tendencias de la GR, analizar los principales marcos de GR y su evolución, comprender la necesidad de uso en la UCI y las peculiaridades del proceso productivo en esta institución**.** Los fundamentos teóricos de MoGeRi provienen de las propuestas del Software Engineering Institute (SEI), el Project Management Institute (PMI) y la Metodología de Análisis y Gestión de Riesgos (MAGERIT) del Ministerio de Administraciones Públicas de España. Las etapas fundamentales se describen en la Figura 1.

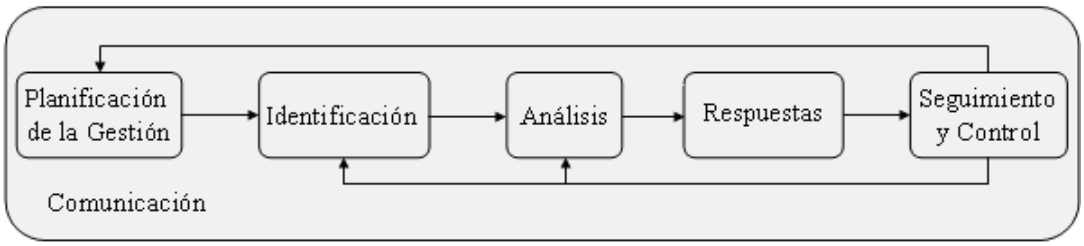

Figura 1. Procesos del MoGeRi.

Las actividades correspondientes a cada proceso son:

Planificación de la gestión de los riesgos.

- Determinación del alcance.
- Planificación de la GR.
- Factibilidad de la GR.
- Comunicación de resultados.

Identificación de los riesgos.

- Selección de herramientas y técnicas a aplicar.
- Identificación de riesgos.
- Comunicación de resultados.

Análisis de los riesgos

- Análisis de los riesgos.
- Priorización de los riesgos.
- Comunicación de resultados.

Respuestas a los riesgos.

- Valoración de la estrategia para enfrentar el riesgo.
- Planificación de las respuestas.
- Comunicación de resultados.

Seguimiento y control de los riesgos.

- Control del cumplimiento de las respuestas a los riesgos.
- Control del cumplimiento de los hitos de GR.
- Aplicación de métricas para valoración de la calidad de procesos, técnicas y herramientas y resultados.

- Comunicación de resultados.

Comunicación de la información sobre los riesgos.

- La comunicación debe ser continua desde el inicio de la GR, lo cual puede apreciarse con la inclusión de una actividad al respecto en cada uno de los procesos anteriormente descritos. Pero no es solo un canal para que fluyan datos en el proyecto, la comunicación debe ganar dimensiones y convertirse en la vía para estipular la información de manera formal y reutilizable: el mismo proyecto y otros, podrán utilizarla como información histórica y aprender de ella.

Los roles implicados en estos procesos son:

- Gestor de riesgos: es el encargado de dirigir los procesos para una GR exitosa, delimitar debidamente el alcance y dominio de la GR, planificar las actividades, priorizar los riesgos, guiar el flujo de la comunicación, valorar la efectividad de la GR y decidir los cambios y mejoras en los procesos.
- Equipo de gestión de riesgos: es el encargado de ejecutar las actividades planificadas para la GR, en especial de identificar, analizar y presentar las respuestas ante los riesgos en los planes de mitigación y contingencia y además mantener actualizado el Registro de Riesgos.
- Equipo de seguimiento y control: es el encargado de verificar el cumplimiento de las actividades y planes de mitigación y contingencia, asegurar la colaboración de todos los involucrados en las actividades y aplicar métricas que permitan monitorear y mejorar la GR.

MoGeRi propone la utilización de varias herramientas durante el ciclo de GR, las más importantes son el Plan de Gestión de Riesgos y el Registro de Riesgos. El primero recoge los objetivos, alcance, dominio, restricciones y las actividades planificadas con los recursos y limitaciones temporales correspondientes. La planificación permite definir los objetivos del proceso dentro del proyecto, su dominio y sus límites, pues un perímetro demasiado amplio o ambiguo podría ser inabarcable, por muy general o de muy largo plazo, con perjuicio en las estimaciones de los elementos del análisis. El registro muestra la evolución de los riesgos del proyecto desde su identificación, pasando por el análisis de su impacto y probabilidad, hasta las estrategias para responder ante ellos y su efecto real en la exposición al riesgo.

Actualmente la tendencia en el análisis de los riesgos se centra en la elección entre el cualitativo y el cuantitativo. La mayoría de los modelos plantean para uno u otro las mismas actividades, técnicas y herramientas, enfocando las diferencias solo en que el primero se basa en estimaciones y el segundo en cuantificaciones. Sin embargo cualquier análisis cuantitativo siempre implica la valoración de los activos: el riesgo en sí no es cuantificable hasta tanto no se valoren los activos que son afectados con su impacto. En este sentido MoGeRi, a diferencia de la propuesta del PMI, enfoca la decisión en el análisis hacia la utilización o no, de las técnicas basadas en activos, que provee MAGERIT, pero dando a los usuarios la posibilidad de elegir en función de las características del proyecto.

La GR es parte de la Gestión del Proyecto y del proceso de desarrollo de software y por tanto, debe también ser controlada y mejorada. La definición de métricas es indispensable para cumplir tal propósito. Como puede apreciarse en la Tabla 1, en la mayoría de los modelos de GR, las mediciones generalmente están orientadas a caracterizar y evaluar el impacto y probabilidad de ocurrencia de los riesgos, y en consecuencia la exposición al riesgo.

MoGeRi ha sido complementado con una *Guía de Métricas* que permite realizar valoraciones sobre el costo de la GR, la efectividad de las herramientas y técnicas empleadas, las facilidades para desarrollar los procesos y actividades, la idoneidad de la definición de roles, el nivel de conocimiento con que cuenta el personal de las responsabilidades y actividades que le han sido asignadas, y por supuesto, sobre el desenvolvimiento de la GR en el proyecto de manera general.

Modelo	**P**	**I**	**A**	**R**	**S-C**	**C**	**Métricas**
Boehm		x	x	x	x		Caracterizar los riesgos
SEI		x	x	x	x	x	Caracterizar los riesgos
PMI	x	x	x	x	x		Caracterizar los riesgos
MAGERIT	x	x	x	x	x		Caracterizar los activos, las amenazas y los riesgos
MoGeRi	x	x	x	x	x	x	Caracterizar los riesgos Medir los resultados Mejorar los resultados

Tabla 1. Planificación (P), Identificación (I), Análisis (A), Planificación de Respuestas (R), Seguimiento y Control (S-C), Comunicación (C) y Métricas, en Modelos de GR.

5. La enseñanza de la GR

No basta con la concepción de un modelo para realizar la GR en los proyectos y que la capacitación a los recursos humanos quede implícita en sus actividades:

- La aplicación directa y aislada del modelo de GR en la producción, no crea la necesaria cultura de que el tiempo invertido en identificar, analizar y planificar las respuestas a los riesgos, constituyen factores que solicitan del proyecto recursos y esfuerzo que luego se revierten calidad de los procesos y resultados finales.
- En los proyectos donde, por limitaciones temporales, decisiones estratégicas o peculiaridades de la fase del ciclo de vida, se resuelva no implementar la GR, ¿los integrantes del equipo serán privados de los conocimientos al respecto?
- Las peculiaridades del proceso productivo en muchas universidades cubanas, y especialmente en la UCI, implican la vinculación real de la mayoría de los estudiantes al desarrollo de proyectos informáticos, por tanto, las habilidades de GR no pueden ser solo adquiridas por aquellos que deban desempañar solo los roles relacionados con el tratamiento del riesgo.

Por estas, entre otras razones, la enseñanza de la GR no puede quedar circunscrita a la estrategia de capacitación de cualquier modelo que se adopte. Partiendo del análisis anterior, se propone un conjunto de principios que deben ser considerados para lograr dimensionar la GR en el plano pedagógico:

5.1.1. Educación temprana en la GR

Educar a los estudiantes en la GR no es fácil. Asimilar las habilidades necesarias generalmente toma varios años [22]. Teniendo en cuenta la experiencia de estos autores, y analizando que la aparición de los riesgos no está limitada por los tipos de actividades del ser humano y tampoco sujeta a delimitaciones temporales, es ineludible la necesidad de la formación temprana de los futuros ingenieros en el tratamiento de los riesgos.

La práctica profesional propicia las condiciones pedagógicas para iniciar este proceso de enseñanza aprendizaje. En ella los estudiantes deben planificar el trabajo para la realización de proyectos de curso sencillos, lo cual debe completarse con la identificación de los riesgos que pueden afectar su desarrollo exitoso. Al concluir, resulta provecho

valorar los riesgos que resultaron problemas y analizar la efectividad de la identificación inicial.

5.1.2. Integración interdisciplinaria

Las etapas de GR, en especial la identificación y el análisis, exigen el empleo de técnicas que afortunadamente son tratadas en otras disciplinas y adaptadas al entorno de esta área. En las asignaturas relacionadas con la disciplina de las Ciencias Empresariales se adquieren habilidades para la recopilación de información y análisis multicriterio a través de la utilización de la técnica DELPHI [23], diagramas de espina de pescado, diagramas causa efecto, análisis de Debilidades, Amenazas, Fortalezas y Oportunidades (DAFO). Por otra parte, la Ingeniería y Gestión del Software facilita el dominio de los procesos de desarrollo de software.

5.1.3. Fusión GR-desarrollo de software

Existen múltiples propuestas para la realización de la GR, variadas en número como en su enfoque del proceso. Sin embargo uno de los retos actuales para los líderes, radica en cómo insertar las actividades de GR como parte del desarrollo del proyecto, sin violentar sus fases, minimizando los costos y logrando la simbiosis con la gestión del proyecto. La GR no puede aislarse del desarrollo de ese software en riesgo y por tanto, la enseñanza de estos dos elementos no puede tampoco separarse. Es muy importante que durante su formación, el ingeniero comprenda el carácter indisoluble de este binomio.

Los riesgos solo podrán ser gestionados con las prácticas, herramientas y técnicas que se logren asimilar durante la formación; pero solo son gestionables en la producción, puesto que es en este ámbito donde surgen los riesgos, desde el punto de vista del desarrollo del software.

Solo con su aplicación armónica y estructurada en el propio desarrollo del software se logrará que la GR sea una práctica real y beneficiosa y no un mito.

6. Experiencias en la Universidad de las Ciencias Informáticas

Además de los principios explicados, en el caso de la UCI, se ejecutan otras acciones integradas al propio proceso docente de pregrado y además a la formación posgraduada.

En el primer caso se concibió un curso optativo de GR como parte del segundo perfil Calidad de Software. Las asignaturas de este perfil están organizadas en los niveles básico, especializado y de aplicación. El curso optativo corresponde al nivel especializado y fue previsto inicialmente para estudiantes de tercer año en adelante. Hasta el momento, ha sido recibido por estudiantes de los grupos de calidad de las 10 facultades de la institución, que tienen bajo su responsabilidad la revisión de los expedientes del proyecto, artefacto donde se exige la lista de riesgos y el plan de mitigación como parte de la gestión del proyecto.

En el caso del posgrado, se imparte el Curso de Gestión de Riesgos en Proyectos Informáticos a los profesores universitarios. No debe olvidarse la superación de los profesionales en esta área pues hasta el momento, los responsables del cumplimiento de la GR en los polos productivos de las facultades, han sido docentes en su mayoría, debido a la poca experiencia de trabajo con los riesgos de forma general.

Otro de los elementos que ha ayudado a la implantación del modelo y a la formación ha sido el análisis de riesgos post-mortem. Una de las dificultades a las se que enfrenta hoy la GR, es la carencia de información histórica que facilite planificar la GR e identificar los riesgos. El análisis de los problemas (riegos hechos realidad) que han enfrentado los proyectos es una de las mejores vías para prevenir que ocurran nuevamente en los de actual desarrollo.

El modelo comenzó su implantación en el año 2006, lo cual permitió identificar sus bondades, debilidades y las acciones necesarias para generalizarlo en el proceso productivo UCI. El primer ciclo de aplicación incluyó proyectos con diferentes características y en diferentes fases del ciclo de vida:

- Proyecto “Atención Primaria de Salud”, encargado de la informatización de la gestión de la información relacionada con este proceso (APS).
- Proyectos del Programa Nacional de Informatización del Conocimiento Geológico en Cuba (PNICG).
- Proyecto “A Jugar”: Software Educativo para la enseñanza preescolar (AJ).
- Proyecto “Sistema de información geográfica de la Universidad de las Ciencias Informáticas (SIG-UCI)”
- Proyecto “Sistema de captura y catalogación de medias (SCCM)”.

- Proyecto “Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado (MANUGAS)”

De manera general los proyectos incluidos en el estudio presentaron problemas en la planificación y gestión, lo que implicó que el proceso de planificación de la GR no se realizara con la madurez requerida. En todos los casos, los equipos de GR presentaron cuestionarios ajustados a los proyectos, a partir de la taxonomía del SEI fundamentalmente, para facilitar la identificación de los riesgos, lográndose muy buena comunicación con el equipo. A continuación se relacionan algunos de los riesgos más comunes y en la Tabla 2 se muestra su porcentaje de aparición en los proyectos analizados:

R1. Insuficiente implicación de los usuarios.

R2. Estimaciones de productividad y calidad que no tienen en cuenta los datos históricos.

R3. Cambios significativos en la estructura organizacional del proyecto.

R4. Las políticas y estándares no se encuentran definidos o no son seguidos.

R5. Inestabilidad en los requerimientos.

R6. Las características del producto dificultan la realización de pruebas.

R7. Programa de formación inadecuado.

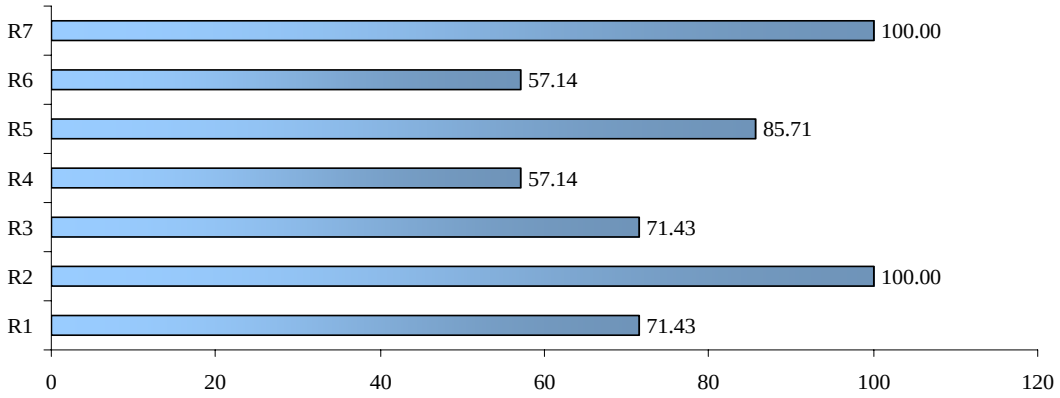

Figura 2. Riesgos más comunes identificados en los proyectos.

El proceso de análisis de los riesgos en ninguno de los proyectos se realizó utilizando las técnicas basadas en la valoración de los activos. La práctica demuestra que preparar a los equipos en este sentido implica el empleo de recursos y por tanto para facilitar su aplicación, se recomienda la participación de un experto que pueda dirigir las actividades.

Además en ninguno de los proyectos se realizan estimaciones de costo basadas en activos.

La planificación de las respuestas se realizó adecuadamente y las acciones se incluyeron en el plan del proyecto. Sin embargo su ejecución no fue totalmente seguida pues no se logró el compromiso necesario de la dirección del proyecto. Los planes de mitigación relacionados con la definición de las políticas y estándares y los programas de formación del equipo, fueron los ejecutados con mayor rigor; en el resto de los casos no se aplicaron con la severidad solicitada. A pesar de estas limitaciones la exposición al riesgo se redujo en más de un 50% en todos los proyectos.

Se considera que no se logró una total integración de las responsabilidades de los roles propuestos en MoGeRi con las responsabilidades de los roles ya definidos en el proyecto y esto también influyó en la disminución del alcance del proceso de seguimiento y control de los riesgos. En todos los casos el rol de gestor de riesgos estuvo desempeñado por estudiantes de 5to año de la carrera Ingeniería en Ciencias Informáticas. Los equipos de gestión de riesgos estuvieron integrados por entre 4 y 5 miembros del proyecto con participación activa de los líderes. No existían experiencias en la aplicación de métricas, este aspecto dificultó también la recolección de datos para la incorporación de las métricas propuestas por MoGeRi.

MoGeRi fomenta la capacidad de la GR que es incluida como área de proceso en la representación del Modelo Integrado de Capacidad y Madurez (CMMI). En la Tabla 2 se describe en qué proceso de MoGeRi se garantiza el cumplimiento de las prácticas específicas propuestas por CMMI.

CMMI	**MoGeRi**
Determinar los orígenes y categorías de los riesgos.	Identificación de los riesgos.
Definir los parámetros de los riesgos.	Identificación de los riesgos.
Establecer una estrategia de GR.	Respuestas a los riesgos.
Identificar los riesgos.	Identificación de los riesgos.
Evaluar las categorías de los riesgos.	Identificación de los riesgos.
Desarrollar planes para la reducir los riesgos.	Respuestas a los riesgos.
Implementar los planes de reducción de riesgos.	Respuestas a los riesgos. Seguimiento y control de los riesgos.

Tabla 2. Prácticas específicas propuestas por CMMI en MoGeRi.

7. Conclusiones

La introducción temprana y didácticamente organizada de la GR, permite a los egresados de las carreras de informática orientarse hacia la minimización y/o evitación de riesgos identificados y analizados, para tomar las decisiones correctas en el momento correcto acerca del rumbo de un proyecto de desarrollo de software. Los innegables beneficios de la GR hacen necesaria su aplicación en el entorno de la producción de software, área que actualmente se ve renovada por la fuerza y creatividad con que las universidades se integran a las empresas. Como solución a esta problemática surge MoGeRi, un Modelo para la Gestión de Riesgos en Proyectos de Desarrollo de Software.

MoGeRi fomenta la comunicación del equipo del proyecto pues en cada proceso definido se emplean las reuniones de análisis que promueven el debate sobre los resultados obtenidos. El registro de riesgos y el plan de GR promueven la reutilización y registro de datos, no solo de los riesgos sino como información histórica del proyecto. Su aplicación en proyectos de diferentes características demuestra sus beneficios a favor del producto final, de la gestión de proyecto y de la formación del equipo involucrado. En los proyectos pilotados se realizó un análisis de la capacidad de la GR según CMMI y en todos los casos se cumplieron las prácticas específicas propuestas.

La implantación del MoGeRi, como la de cualquier otro marco de trabajo, no puede hacerse sin un diagnóstico inicial en la institución y en los propios proyectos, serán estos resultados los que permitan aplicar en mayor o menor medida los principios aquí descritos y apoyarlos además con la ejecución de otras acciones de formación desde el pregrado, el posgrado o la producción. Esta estrategia facilitará la capacitación en las técnicas de GR que propicia el propio modelo, es por esto que no se separa una de la otra.

Referencias

[1] Castro Díaz-Balart, F., *Ciencia, Tecnología y Sociedad. Hacia un desarrollo sostenible en la Era de la Globalización,* Editorial Científico-Técnica, 2004.

[2] Charette, R., *Software Engineering Risk Analysis and Management,* McGraw–Hill/Intertext, 1989.

[3] Ministerio de Administraciones Públicas, *Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información. Método. (v 1.1)*, Catálogo general de publicaciones oficiales, 2006.

[4] Consejo Superior de Informática, *Eurométodo v1*, Ministerio de Administraciones Públicas, Madrid, 1997.

[5] SEI. *Risk Management*. 2000 [Consultado: enero de 2008] http://www.sei.cmu.edu/news-at-sei/columns/the_cots_spot/2000/march/cots-mar00.htm]

[6] ISO/IEC, *Information technology. Security techniques. Management of information and communications technology security*, 2004.

[7] Marcelo, J. y Fernández, M., "Risks and Project Management", *UPGRADE*, vol VIII, nº 5, pp. 36-41, 2007.

[8] PMI, *A guide to the Project Management Body of Knowledge (PMBoK).*,PMI Communications. ISBN 1930699506, 2004.

[9] Kahkonen, K. "Integration of Risk and Opportunity Thinking in Projects". En: *Fourth European Project Management Conference*, 2001.

[10] Mochal, T., *Factor Positive Risk Into Project Planning*, Tech Republic, 2002.

[11] Data and Analysis Center for Software, ACS *Software Acquisition Gold Practices, Formal Risk Management, 2008*. [Consultado: enero de 2008] www.goldpractices.com/practices/frm/index.php

[12] Proyecto Eurométodo, *EUROMÉTODO v1. Diccionario*, 1996.

[13] Marcelo, J., Rodenes, M. y Torralba, J., "Estudio exploratorio sobre los métodos de gestión de proyectos de alto riesgo". En: *Primer Congreso SOporte del COnocimiento con la TEcnología, SOCOTE*. Valencia (España), 2003.

[14] ESG-UQAM, *Results Summary*. 2007, University of Quebec at Montreal. School of Bussines Administration: Montreal. p. 24 [Consultado: enero de 2008] http://www.practices-survey.esg.uqam.ca/

[15] Economist Intelligence Unit, "Coming to grips with IT risk". *The Economist Intelligence Unit*, 2007. [Consultado: enero de 2008] www.eiu.com/site_info.asp?info_name=eiu_SAP_Coming_to_grips_with_IT_risk&rf=0

[16] Kulik P. y Weber C., *Software Risk Management Practices*, 2001. [Consultado: enero de 2008].

www.visualbasic.ittoolbox.com/browse.asp?c=VBPeerPublishing&r=%2Fpub%2FPK083001.pdf

[17] Zulueta, Y., "Modelo de Gestión de Riesgos en Proyectos de Desarrollo de Software", En: Memorias de *III Conferencia Científica UCIENCIA*, Ciudad de La Habana, 2007.

[18] Universidad de Lima. *Plan de estudios. Facultad de Ingeniería de Sistemas.* 2003 [Consultado: noviembre de 2008] http://fresno.ulima.edu.pe/sf/sf_bd6500.nsf/default/8FA58232C4B1827F05256F8E00751165/$file/plan2.pdf]

[19] Universidad Politécnica de Madrid., *Plan de Estudios. Titulación: Ingeniero en Informática.* 2006 www.upm.es/estudios/oficiales/titulaciones/plan_estudio/finform_96.pdf

[20] Universidad de Oviedo. *Proyectos de informática*. 2007 [Consultado: Noviembre de 2008] http://euitio178.ccu.uniovi.es/wiki/index.php/Proyectos_de_inform%C3%A1tica

[21] Universidad Latinoamericana de Ciencia y Tecnología. *Ingeniería Informática. Plan de estudios.* 2007 http://www.ulacit.ac.cr/doc/maya/300-014.pdf

[22] Boehm, B. y Port, D., *Educating Software Engineering Students to Manage Risk*, CS510, University of Southern California, 2004.

[23] Hernández, R. *Curso básico de gestión de proyectos*. Universidad de las Ciencias Informáticas, Dirección de Investigaciones, 2

Una herramienta para la reducción de conjuntos de casos de prueba

Pedro Reales
Escuela Superior de Informática – Universidad de Castilla-La Mancha – España
pedro.reales@uclm.es

Macario Polo
Escuela Superior de Informática – Universidad de Castilla-La Mancha – España
macario.polo@uclm.es

Resumen

Este artículo presenta un algoritmo de reducción para conjuntos de casos de prueba en formato JUnit, así como una herramienta (en forma de plugin para Eclipse) que lo implementa, y que puede descargarse desde http://geclipsetesting.sourceforge.net/. El objetivo de la reducción de conjuntos de casos de prueba es la obtención de una nueva versión del conjunto, pero con menos casos de prueba, mientras que se mantiene la cobertura alcanzada se mantiene. El algoritmo, y la implementación que se la ha dado en el plugin, consiguen estos resultados.

Palabras clave: Generación de casos de prueba, reducción de conjuntos de casos de prueba, JUnit, cobertura de código.

A tool for minimizing sets of test cases

Abstract

This article presents an algorithm for reducing JUnit test suites, as well as a tool (implemented as an Eclipse plugin) that realizes it. The tool can be downloaded from http://geclipsetesting.sourceforge.net/. With test suite reduction, the test engineer may get smaller test suites that, however, keep the same coverage that the original one. The algorithm and its implementation fulfil these goals.

Key words: Test case generation, test suite reduction, JUnit, code coverage.

Zulueta,Y., Despaigne,E. y Hernández A., "Una herramienta para la reducción de conjuntos de casos de prueba", REICIS, vol. 5, no.3, 2009, pp.21-37. Recibido: 15-3-2009; revisado: 10-4-2009; aceptado: 19-9-2009.

1. Introducción

La priorización de los casos de prueba es una práctica recomendada para reducir los costes en las pruebas de regresión: básicamente, la idea consiste en ejecutar aquellos casos de prueba que son más importantes de acuerdo a diferentes criterios de calidad. Una manera de priorizar los casos de prueba consiste en reducir el tamaño del *test suite* (conjunto de casos de prueba) sin perder calidad. Más formalmente, se transforma el conjunto de casos de prueba T en un nuevo conjunto $T' \subseteq T$, siendo $|T'| \leq |T|$ y preservando T' la misma cobertura alcanzada que con el conjunto T. Este problema ha sido analizado en muchos contextos por diferentes investigadores. Jones y Harrold [1] plantean el problema de "reducción al conjunto óptimo de casos de prueba" como se muestra en la Figura 1.

Dados: Un conjunto de casos de prueba *T*, un conjunto de requisitos de prueba $r_1, r_2, ..., r_n$, que deben ser satisfechos para alcanzar la cobertura deseada en el programa bajo prueba.
Problema: Encontrar $T' \subset T$ *de manera que T' satisface todos los* r_i *y* $(\forall T'' \subset T, T''$ *satisface todos los* $r_i \Rightarrow |T'| \leq |T''|)$

Figura 1. Problema de reducción al conjunto óptimo de casos de prueba [1]

El problema de la reducción óptima (es decir, la obtención de un *test suite* nuevo cuyo cardinal sea el mínimo posible) es NP-completo [2], por lo que no es un problema resoluble en tiempo polinomial. Por esta razón, todos los algoritmos que tratan este problema obtienen soluciones próximas a la óptima, pero no ofrecen garantías de que la solución ofrecida sea, desde el punto de vista del cardinal del *test suite*, la mejor posible. La mayoría de los algoritmos desarrollados son voraces, y obtienen *test suites* reducidos que cumplen los mismos requisitos de calidad (normalmente, algún criterio de cobertura) que el original.

Aunque durante años las actividades de prueba se han llevado a cabo de una manera relativamente descuidada, la introducción, hace algunos años, de los *frameworks* automatizados tipo X-Unit (JUnit, NUnit, etc…), ha permitido a las organizaciones desarrolladoras de software ir introduciendo, progresivamente, buenas prácticas de *testing* en sus proyectos de desarrollo [3]. En torno a estos *frameworks*, que supusieron realmente un avance importante para la "democratización" el *testing*, diferentes investigadores y

compañías han desarrollado extensiones que permiten algunas funcionalidades adicionales, como pruebas de interfaces de usuario (mediante UISpec, por ejemplo), pruebas de caja blanca (con herramientas o plugins como Coverlipse o Eclemma Code Coverage [4], descargable éste desde http://www.eclemma.org/), y pruebas de mutación (como Muclipse, un plugin para Eclipse basado en MuJava [5]).

El uso conjunto de JUnit y Eclemma permite detectar fallos en el sistema bajo prueba, a la vez que se obtiene una estimación del código recorrido por los casos de prueba, en forma de un valor de la cobertura de sentencias de código fuente o de instrucciones de *bytecode*. Para disminuir el coste derivado de la reejecución de los casos de prueba hemos desarrollado e implementado un algoritmo que reduce el tamaño de un conjunto de casos de prueba en formato JUnit, mientras que preserva la cobertura alcanzada en el sistema bajo prueba.

Este artículo se organiza de la siguiente manera: en la sección 2 se hace una introducción al problema de la reducción de los casos de prueba y se describen algunos algoritmos que lo resuelven. En la sección 3 se presenta el algoritmo que hemos desarrollado, cuya implementación se describe en al sección 4. En la sección 5 se presenta un "ejemplo motivacional" extraído de la literatura de referencia. En la sección 6 se describen algunos experimentos llevados a cabo. Por último, la sección 7 muestra las conclusiones a las que se han llegado.

2. Algoritmos para reducir conjuntos de casos de prueba.

Supongamos que tenemos una versión orientada a objetos del clásico problema de la determinación del tipo de un Triangulo [6]: el lado izquierdo de la Figura 2 representa la posible estructura de esta clase: dispone de tres métodos de configuración (*setI*, *setJ* y *setK*, que asignan valor a los tres lados del triángulo), y de un método *getType* (que devuelve un valor numérico que representa si la figura es un triangulo o no, y el tipo de triángulo en caso afirmativo: equilátero, isósceles o escaleno).

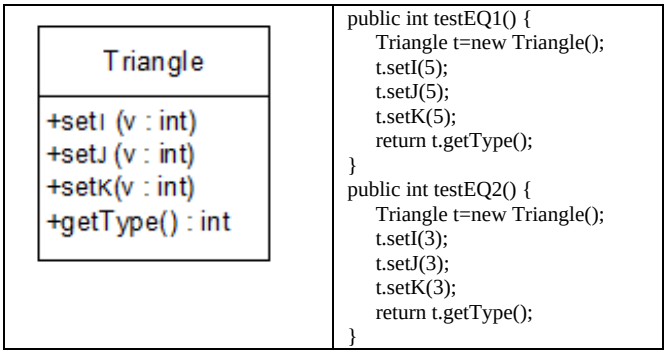

Figura 2. Representación del clásico problema del triangulo mediante una clase

En el lado derecho de la figura anterior aparecen dos posibles casos de prueba para la clase: ambos corresponden a un triángulo equilátero y, probablemente, ambos ejecutarán el mismo conjunto de instrucciones (o cualquier otro criterio de cobertura) de la clase bajo prueba. Desde el punto de vista de las pruebas, el segundo caso de prueba es redundante con respecto al primero, por lo que, si se fuese a reducir el tamaño de este *test suite*, el ingeniero de pruebas podría quedarse solamente con uno de ellos.

El problema de la redundancia de casos de prueba puede ser serio cuando existen múltiples casos de prueba y su redundancia no es tan evidente como ocurre en este sencillo ejemplo. En estas situaciones, el ingeniero de pruebas debería aplicar algún tipo de estrategia de reducción al conjunto de casos de prueba para obtener un conjunto más pequeño, pero que alcance la misma cobertura que el conjunto original en el sistema bajo prueba. En esta sección se hace una revisión de algunos algoritmos desarrollados por diferentes investigadores para reducir conjuntos de casos de prueba.

2.1 El algoritmo HGS

Harrold, Gupta y Sofa [7] proponen un algoritmo voraz (normalmente referenciado como HGS) para reducir un conjunto de casos de prueba preservando los requisitos de las pruebas cubiertos por el conjunto original. Los “requisitos de las pruebas” (*test requirement*) son, normalmente, uno o más criterios de cobertura, de manera que este algoritmo permite seleccionar un subconjunto de casos que verifique más de un criterio de cobertura.

Los principales pasos de este algoritmo son:

1. Inicialmente, todos los requisitos están sin marcar.

2. Añadir los casos de prueba que ejecutan sólo un requisito al conjunto de casos de prueba reducido y marcar todos los requisitos cubiertos por los casos de prueba seleccionados.
3. Ordenar los requisitos sin marcar de acuerdo al número de casos de casos de prueba que ejecutan cada requisito. Si varios requisitos son cubiertos por el mismo número de casos de prueba, se marca el mayor número de estos requisitos. Si además hay múltiples casos de prueba "empatados", se selecciona el caso de prueba que marque más requisitos en total. Si aun así sigue habiendo un empate, se selecciona un caso de prueba arbitrariamente. Entonces, se marcan los requisitos ejecutados por el caso de prueba seleccionado, y se eliminan los casos de prueba redundantes con respecto a los casos de prueba ya seleccionados.
4. Repetir el paso 3 hasta que todos los requisitos de pruebas hayan sido marcados.

2.2 Mejoras de Gupta

Con diferentes colaboradores, Gupta ha propuesto una serie de mejoras para mejorar el algoritmo HGS:

- Con Jeffrey [8], Gupta añade al algoritmo "redundancia selectiva". "La redundancia selectiva" hace posible la selección de un caso de prueba que, por cualquier requisito de prueba dado, alcanza la misma cobertura que otro caso de prueba seleccionado anteriormente, pero que ahora añade cobertura de un requisito de prueba diferente. Por ejemplo, se puede dar la posibilidad de que T' alcance el criterio de todas las ramas, pero no el de *def-uses*; por lo tanto, se puede añadir un nuevo caso de prueba t a T' si éste incrementa la cobertura de los requisitos de *def-uses*: ahora, T' no incrementará el criterio de todas las ramas, pero sí que incrementa el criterio de *def-uses*.
- Con Tallam [9], la selección de los casos de prueba se basa en técnicas de análisis de conceptos. De acuerdo con los autores, esta versión alcanza mejores reducciones que las versiones previas, sin un empeoramiento en el tiempo de ejecución.

2.3 Algoritmo de Heimdahl y George

Heimdahl y George [10] también proponen un algoritmo voraz para reducir conjuntos de casos de prueba. Básicamente, lo que hacen es tomar un caso de prueba aleatoriamente, lo ejecutan y comprueban la cobertura alcanzada. Si esta cobertura es mayor que la cobertura alcanzada hasta ahora, se añade el caso de prueba al conjunto final.

Este algoritmo es repetido cinco veces para obtener cinco conjunto reducidos de casos de prueba diferentes. Puesto que el azar es un componente esencial, este algoritmo no puede garantizar una buena calidad de los resultados.

2.4 Algoritmo de McMaster y Memon

McMaster y Memon [11] también presentan un algoritmo voraz. El parámetro tomado en cuenta para incluir los casos de prueba en el conjunto reducido se basa en el número de llamadas hechas a la pila del programa bajo prueba. Como se puede ver, este criterio de selección no es un requisito de pruebas frecuente.

2.5 Resumen

Puesto que el problema de reducir un conjunto de casos de prueba es un problema NP-Completo, todas las propuestas discutidas presentan un algoritmo voraz para encontrar una buena solución en un tiempo polinomial. Los requisitos de las pruebas para seleccionar los casos de prueba pueden ser cualquiera: cobertura de bloques, de caminos, etc.

El algoritmo presentado en este trabajo es también un algoritmo voraz, garantizando que la cobertura alcanzada por T’ es la misma que la alcanzada por T. Las principales diferencias con los trabajos discutidos anteriormente son la implementación amigable de este algoritmo en una herramienta fácilmente usable, su aplicación con casos de prueba en formato JUnit (un *framework* ampliamente extendido) y la posibilidad de usar las ventajas de herramientas de terceros (como se verá más adelante, nuestra herramienta hace uso de las capacidades de Eclemma).

3. Un algoritmo para reducir conjuntos de casos de prueba basado en cualquier criterio de cobertura

El algoritmo voraz presentado aquí puede hacer uso de cualquier criterio de cobertura para incluir casos de prueba en el conjunto reducido de casos de prueba final.

Inicialmente, el algoritmo ejecuta todos los casos de prueba del conjunto original T y registra los elementos de código fuente (por ejemplo, sentencias) alcanzados por cada caso de prueba. Después, el algoritmo sigue los siguientes pasos: (1) añade al conjunto reducido T' el caso de prueba *t* que alcance el mayor valor de cobertura para el criterio seleccionado (por ejemplo, selecciona el caso de prueba que alcanza un mayor número de sentencias); (2) elimina aquellos casos de prueba que sólo ejecutan elementos que ya han sido visitados por los casos de prueba que están en T' (por ejemplo, la primera vez se borran los casos de prueba que solo ejecutan sentencias ejecutadas por *t*); (3) se repiten los pasos (1) y (2) hasta que todos los elementos alcanzados por los casos de prueba de T sean alcanzados por los casos de prueba de T'.

A modo de ejemplo, la Tabla 1 muestra la matriz de alcance que un conjunto de casos de prueba (compuesto por los casos *tc1* al *tc6*) obtiene en una supuesta clase (compuesta por las sentencias *s1* a *s7*). Esta matriz se construye en el primer paso del algoritmo. Después, el algoritmo toma el caso de prueba que ejecuta más sentencias, que en el ejemplo pueden ser *tc2* o *tc3*. Supongamos que se selecciona *tc2*, el cual se añade a T'. Puesto que las sentencias alcanzadas por *tc1* y *tc5* también son ejecutadas por *tc2*, se eliminan *tc1* y *tc5*, dejando la matriz como en la Tabla 2.

	tc1	tc2	tc3	tc4	tc5	tc6
s1	X	X				
s2	X	X			X	
s3		X				
s4			X			
s5			X			
s6			X	X		
s7				X		X

Tabla 1. Matriz de alcance (I) de un programa

	tc2	tc3	tc4	tc6
s1	X			
s2	X			
s3	X			
s4		X		
s5		X		
s6		X	X	
s7			X	X

Tabla 2. Matriz de alcance (II) de un programa

Después, el algoritmo selecciona el siguiente caso de prueba que ejecuta más sentencias (*tc3*) y lo añade al conjunto reducido de casos de prueba. Hecho esto, el algoritmo selecciona *tc4* y elimina *tc6*, de manera que el conjunto final queda como sigue:

$$T'=\{tc2, tc3, tc4\}.$$

Realizar una comparación de este algoritmo con los algoritmos comentados en la sección 2 presenta muchas dificultades: (1) en primer lugar, porque a veces es técnicamente complicado implementar los algoritmos, por ejemplo, el criterio usado por McMaster y Memon ("llamada únicas a la pila"). (2) El algoritmo de Heindahl y George selecciona el conjunto reducido de casos de prueba mediante azar, lo cual no garantiza la calidad y, además, la comparación sería diferente después de cada ejecución. (3) El agoritmo HGS y sus mejoras usan dos o más tipos de requisitos de las pruebas para realizar la selección de los casos de prueba: Nuestro algoritmo toma el criterio del plugin Eclemma Code Coverage (instrucciones de código byte, sentencias de código fuente, métodos, tipos y bloques), entre los cuales existen relaciones subsunción (por ejemplo, el criterio de sentencias subsume al criterio de métodos). Además, cuando reducimos por el criterio más estricto de los que soporta Eclemma, estamos también reduciendo por los otros criterios, por lo que la comparación con el algoritmo HGS no tiene sentido. Si el algoritmo HGS fuera aplicado con únicamente un criterio de los permitidos por Eclemma, los resultados serían similares a los nuestros.

4. Implementación del algoritmo en Geclipse

El algoritmo ha sido implementado en *Geclipse*, un plugin para Eclipse el cual, de forma amigable, ofrece algunas de las funcionalidades de *testooj* [12] dentro del entorno de desarrollo. Geclipse permite también la generación automática de casos de prueba para una clase determinada con diferentes estrategias de generación [13] (*each choice*, algunas variantes de *pair-wise* y *all combinations*).

Geclipse utiliza Eclemma para acceder a la cobertura alcanzada por cada caso de prueba y de esta manera poder construir la matriz de alcance. Hecho esto, Geclipse implementa el algoritmo descrito en la sección 3 para reducir la matriz.

El diseño arquitectónico de Geclipse, Eclemma y Eclipse se muestra en la Figura 3: el paquete Eclemma representa los elementos del plugin Eclemma invocados por Geclipse, el

cual esta compuesto por los paquetes presentación, dominio y persistencia. Como se muestra en la Figura 3. Arquitectura de Geclipse, *ControladorGCP* (el controlador de Geclipse) invoca a la clase *SessionAnalyzer* de Eclemma para obtener la cobertura alcanzada por cada caso de prueba. Después, estas coberturas se procesan mediante las clases del paquete *conjuntoMinimo* de Geclipse.

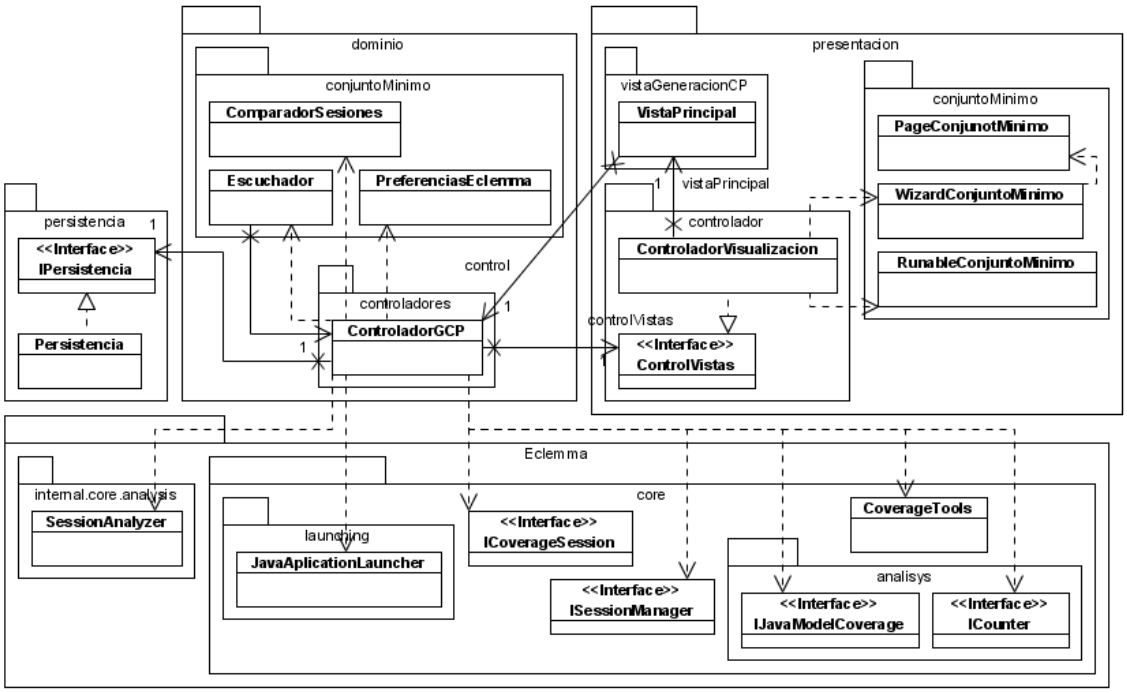

Figura 3. Arquitectura de Geclipse

Eclemma es capaz de medir cinco criterios de cobertura diferentes: instrucciones de código byte, sentencias de código fuente, métodos, tipos y bloques. En una sesión de pruebas, este plugin recoge todas las medidas y las guarda en un objeto de tipo *ICoverageSession* (Figura 3); después, Geclipse procesa esos resultados para realizar los análisis correspondientes.

Geclipse también ofrece la posibilidad de generar casos de prueba usando expresiones regulares (las cuales de diseñan usando el conjunto de operaciones públicas de la clase bajo prueba), asignar valores de prueba a los parámetros de las operaciones y ejecutar los casos de prueba. Esta técnica de generación de casos de prueba se inspira en la propuesta de Kirani y Tsai [14], realizada en 1994, y ya había sido implementada en la herramienta *testooj* [12].

En la Figura 4 (lado derecho), el ingeniero de pruebas está escribiendo una expresión regular para generar casos de prueba para la clase *TriangleType* (problema de la determinación del tipo de un triángulo antes mencionado). Esta expresión será procesada usando el paquete *java.util.regex* para, mediante su expansión, obtener "plantillas de prueba". Una plantilla de prueba consiste en una secuencia de expresiones que, más tarde, debe recibir valores de prueba para, mediante su combinación, pueden poder obtener casos de prueba ejecutables. Así, por ejemplo, a partir de la expresión regular *TriangleType().[setI(int)|setJ(int)|setK(int)]*getType(),* podrían obtenerse, entre otras muchas, las plantillas de prueba siguientes:

TriangleType().getType()

TriangleType().setI(int).getType()

TriangleType().setI(int).setJ(int).setK(int).getType()

TriangleType().setI(int).setJ(int).setK(int).setI(int).getType()

…

Para obtener casos de prueba ejecutables, se asignan valores de prueba (*test data*) a los parámetros de las operaciones que intervienen en la expresión regular. Asignando, por ejemplo, los valores 1 2 y 3 al único parámetro de los métodos *setI*, *setJ* y *setK*, de la tercera plantilla de prueba que se acaba de poner como ejemplo, podrían obtenerse los siguientes casos de prueba:

TriangleType().setI(1).setJ(1).setK(1).getType()

TriangleType().setI(1).setJ(1).setK(2).getType()

TriangleType().setI(1).setJ(1).setK(3).getType()

TriangleType().setI(1).setJ(2).setK(1).getType()

…

En el árbol de la Figura 4 (lado izquierda) se muestran algunas de las plantillas de prueba procedentes de la expresión regular de la Figura 4 (Derecha). Como ya se ha comentado, estas plantillas de prueba deben ser combinadas con datos de prueba reales.

Por otro lado, también es posible que el resultado esperado de alguno de los casos de prueba sea el lanzamiento de una excepción. En la ventana mostrada en la Figura 5, el ingeniero de pruebas está, por un lado, asignando valores de prueba a los parámetros de las operaciones; por otro, puede asociar excepciones a los casos de prueba que se generarán.

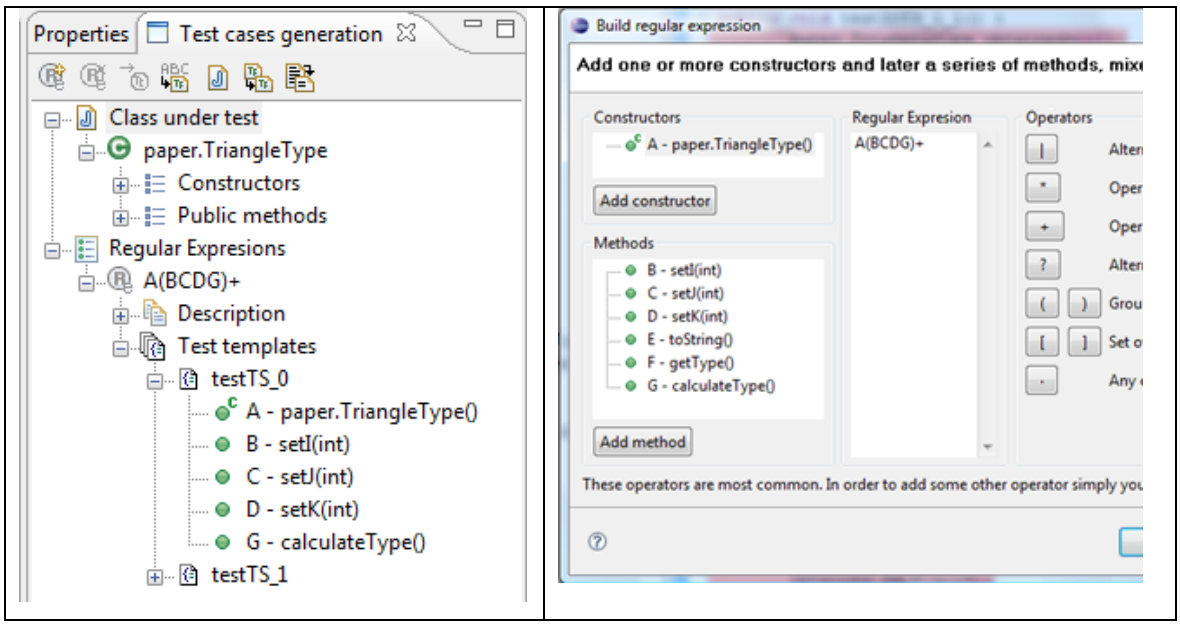

Figura 4. Vista de Geclipse con la estructura de una clase bajo prueba y las plantillas de pruebas generadas (izquierda). A la derecha, el *tester* escribe una expresión regular para generar casos de prueba

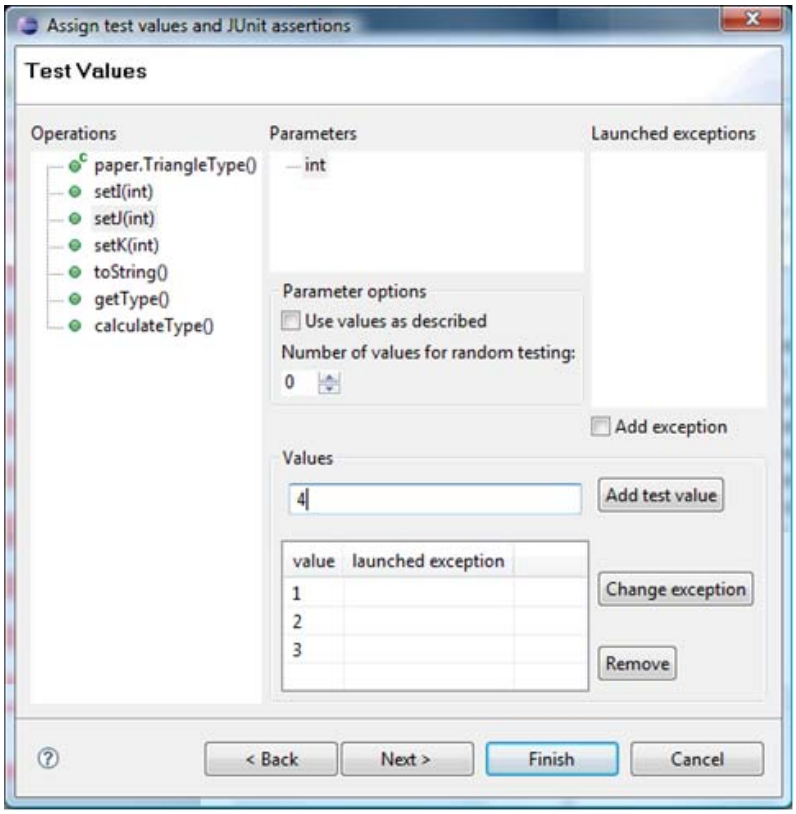

Figura 5. Asignación de valores de prueba

Cuando el conjunto de casos de prueba está disponible (ya sea mediante su confección manual, o mediante su generación automática a partir de la expansión de expresiones regulares), se pueden llevar a cavo las tareas de ejecución y reducción del *test suite.*

El lado izquierdo de la Figura 6 muestra las diferentes opciones de cobertura para reducir el conjunto de casos de prueba: Eclemma evalúa la cobertura que alcanzan los casos de prueba en todo el sistema que se está probando. Geclipse permite reducir el conjunto de casos bien fijándose en la cobertura del sistema completo, bien fijándose sólo en una clase. Igualmente, el usuario puede elegir cuál de los cinco criterios de cobertura medidos por Eclemma se utilizará para realizar la reducción. Durante la ejecución de los casos (lado derecho de la Figura 6), Geclipse utiliza la información de cobertura suministrada por Eclemma para realizar la reducción del *test suite.*

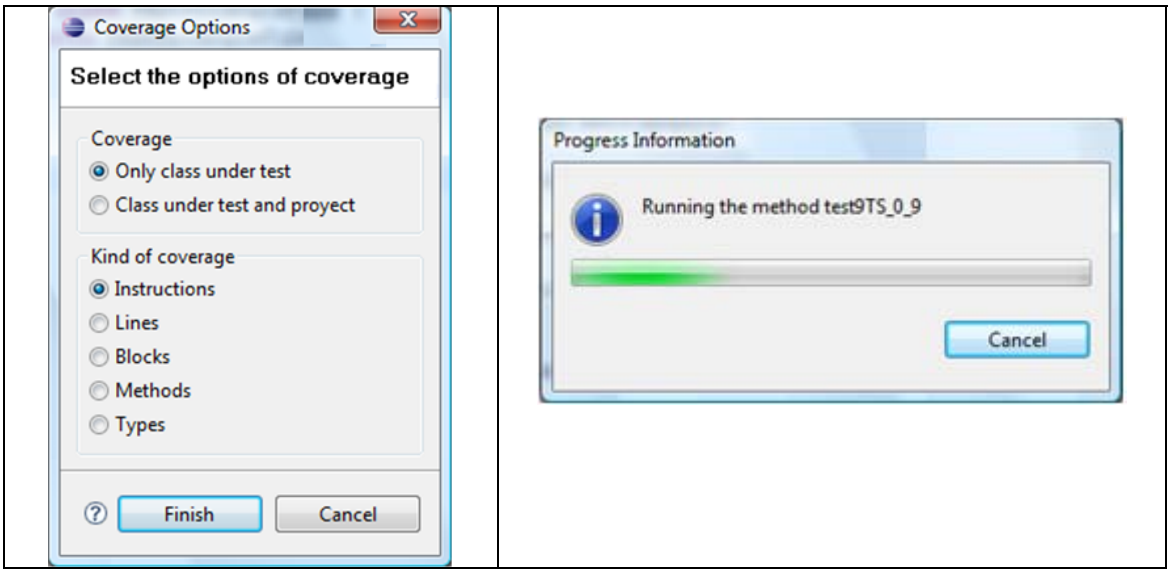

Figura 6. Selección del tipo de cobertura (Izquierda) y proceso de reducción del conjunto de casos de prueba (Derecha)

Finalmente, cuando la reducción se ha completado, la herramienta construye un segundo conjunto de casos de prueba que sólo contiene los casos de prueba seleccionados mediante el algoritmo (Figura 7), y el cual obtiene la misma cobertura que el conjunto original. De este modo, los costes de reejecución de casos (sobre todo, en situaciones de realización de pruebas de regresión) pueden verse disminuidos de forma importante, ya que sólo se ejecutan los ejemplares verdaderamente significativos.

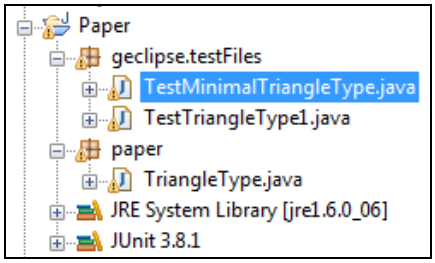

Figura 7. Conjunto de casos de prueba reducido

5. "Un ejemplo motivador"

En el trabajo [8], Jeffrey y Gupta muestran, con el mismo título que esta sección (*A motivational example*), un pequeño programa para ejemplificar su algoritmo con redundancia selectiva. Este programa ha sido traducido al código Java en la Figura 8.

```
public class JGExample {
    float returnValue;
    public float f(float a, float b, float c, float d) {
        float x=0, y=0;
        if (a>0)
            x=2;
        else
            x=5;
        if (b>0)
            y=1+x;
        if (c>0)
            if (d>0)
                returnValue=x;
            else
                returnValue=10;
        else
            returnValue=(1/(y-6));
        return returnValue;
    }
    public String toString() {
        return "" + returnValue;
    }
}
```

Figura 8. Una versión en Java del "ejemplo motivador" de Jeffrey y Gupta

Usando tres valores de prueba (-1.0, 0.0 y 1.0) para cada uno de los cuatro parámetros de la función *f*, y generando casos de prueba con el algoritmo *all-combinations*, Geclipse genera un archivo en formato JUnit con 3x3x3x3=81 casos de prueba, que alcanzan todas las sentencias de la clase bajo prueba.

Después de aplicar nuestro algoritmo, el conjunto de 81 casos de prueba se reduce a 3 casos de prueba (un 3,7% del conjunto original).

6. Validaciones adicionales

Además del ejemplo ilustrativo de Jeffrey y Gupta, la implementación del algoritmo ha sido aplicado a un conjunto de programas *benchmark*, que de acuerdo con [15], se pueden clasificar en dos categorías:

- *Toy programs* (programas "de juguete"): hemos usado algunos de los programas que Pargas y Harrold [16] usaron para validar sus algoritmos de generación de datos de prueba. Para estos programas generamos un conjunto de casos de prueba con el algoritmo *all-combinations* de *testooj*.
- *Industrial programs* (programas "industriales"): la clase *PluginTokenizer* de la aplicación *jtopas* incluida en el Software Infrastructure Repository (SIR) [17], una propuesta de Do, Elbaum y Rothermel que persigue la puesta a disposición de la comunidad científica de un conjunto de programas relativamente complejos, que puedan ser utilizados como referencia para la realización y replicación de experimentos de *testing*. Para nuestro experimento se usaron los 14 casos de prueba que se proporcionan en esta *infraestructura* para la prueba de esa clase. *jtopas* implementa un párser de archivos XML y está compuesto por 22 clases.

Los resultados obtenidos después de aplicar el algoritmo son los que se muestran en la Tabla 3. Como se puede ver, se consiguen reducciones próximas al 90% del tamaño del *test suite* original. En este ejemplo, el criterio de cobertura seleccionado es el más estricto que permite el plugin de "Eclemma code coverage", sentencias, que es uno de los más usados en el desarrollo de software no crítico.

Los resultados con las clases *TriTyp* y *PluginTokenizer* son especialmente ilustrativos: el primer ejemplo es el ejemplo más usado en el campo de la investigación sobre las pruebas del software; en el segundo caso, los desarrolladores de la clase podrían haber escrito únicamente uno de los casos de prueba para alcanzar la misma cobertura que con el conjunto de casos de prueba completo.

Obviamente, la aplicación de cualquiera de los algoritmos revisados en la sección 2 o de nuestro algoritmo sobre un conjunto de de casos de prueba que ya es mínimo, no producirá ningún tipo de reducción: es un conjunto reducido, y no puede volver a ser reducido usando el mismo criterio de cobertura y el mismo algoritmo.

			# of test cases		
	Programa	**Nº de sentencias**	**Original**	**Reducido**	**Reducción**
Toy	Bisect	19	8	1	87,5%
	Bub	27	320	1	99,7%
	TriTyp	49	216	7	96,7%
Industrial	PluginTokenizer	157	14	1	92,8%

Tabla 3. Resultados al aplicar el algoritmo de reducción al conjunto de programas benchmark

7. Conclusiones y trabajos futuros

En este artículo se ha presentado un algoritmo eficiente para reducir el tamaño de un conjunto de casos de prueba en formato JUnit. JUnit (y en general todos los entornos X-Unit) son los entornos de pruebas más utilizados actualmente en las compañías de desarrollo de software. La combinación de JUnit con otras herramientas para realizar pruebas de caja blanca (como pueden ser Eclemma o Coverlipse) permite a los desarrolladores conocer la cobertura alcanza por sus casos de prueba. A su vez, esto conduce a la adición de nuevos casos de prueba, que permitan el recorrido de las zonas de código que permanecían inexploradas. Sin embargo, muchos de estos casos de prueba son probablemente redundantes, lo que finalmente incrementa el coste de la re-ejecución de los casos de prueba. Por lo tanto, las habilidades de herramientas como Geclipse proporcionan una buena ayuda para disminuir los costes de las pruebas de regresión.

Según nuestras noticias, Geclipse es la primera herramienta que incluye la implementación de un algoritmo para reducir conjuntos de casos de prueba aplicable a una herramienta de pruebas ampliamente usada como es JUnit.

Como trabajos futuros, sería interesante la posibilidad de incluir en la herramienta más algoritmos de reducción de conjuntos de casos de prueba, como los revisados en la sección 2. Por ejemplo, implementando el algoritmo HGS se daría la posibilidad de usar más de un criterio de cobertura al mismo tiempo. Relacionado con los criterios de cobertura, la integración de nuestro plugin con otras herramientas, que soporten más criterios de cobertura, sería también interesante.

Por último, indicar que Geclipse puede descargarse e instalarse desde la siguiente URL: http://geclipsetesting.sourceforge.net/

Agradecimientos

Este trabajo está parcialmente soportado por el proyecto PRALÍN (Pruebas en Líneas de Producto): Junta de Comunidades de Castilla-La Mancha/Fondo Europeo de Desarrollo Regional (FEDER), PAC08-0121-1374.

Referencias

[1] Jones JA and Harrold MJ. "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage". IEEE Transactions on Software Engineering, vol. 29, nº 3, pp. 195-209, 2003.

[2] Garey MR and Johnson DS. *Computers and Intractability*. New York: W.H. Freeman, 1979.

[3] Do H, Rothermel G and Kinner A. "Prioritizing JUnit test cases: An empirical assessment and cost-benefit analysis". Empirical Software Engineering, vol. 11, nº 1, pp. 33-70, 2006.

[4] Hoffmann M. "Code Coverage Analysis for Eclipse". Eclipse Summit Europe 2007. Ludwigsburg, 2007.

[5] Ma Y-S, Offutt J and Kwon YR. "MuJava: an automated class mutation system". Software Testing, Verification and Reliability, vol. 15, nº 2, pp. 97-133, 2005.

[6] Myers B. *The Art of Software Testing*: John Wiley & Sons, 1979.

[7] Harrold M, Gupta R and Soffa M. "A methodology for controlling the size of a test suite". ACM Transactions on Software Engineering and Methodology, vol. 2, nº 3, pp. 270-285, 1993.

[8] Jeffrey D and Gupta N. "Test suite reduction with selective redundancy". International Conference on Software Maintenance. Budapest (Hungary), pp. 549-558. IEEE Computer Society, 2005.

[9] Tallam S and Gupta N. "A concept analysis inspired greedy algorithm for test suite minimization". 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, pp. 35-42. 2005.

[10] Heimdahl M and George D. "Test-Suite Reduction for Model Based Tests: Effects on Test Quality and Implications for Testing". 19th IEEE International Conference on Automated Software Engineering, pp. 176-185. 2004.

[11] McMaster S and Memon A. "Call Stack Coverage for Test Suite Reduction". 21st IEEE International Conference on Software Maintenance. Budapest (Hungary), pp. 539-548, 2005.

[12] Polo M, Piattini M and Tendero S. "Integrating techniques and tools for testing automation". Software Testing, Verification and Reliability, vol. 17, nº 1, pp. 3-39, 2007.

[13] Grindal M, Offutt AJ and Andler SF. "Combination testing strategies: a survey". Software Testing, Verification and Reliability, vol. 15, nº, pp. 167-199, 2005.

[14] Kirani S and Tsai WT. "Method sequence specification and verification of classes". Journal of Object-Oriented Programming, vol. 7, nº 6, pp. 28-38, 1994.

[15] Juristo N, Moreno AM and Vegas S. "A Survey on Testing Technique Empirical Studies: How Limited is our Knowledge". International Symposium on Empirical Software Engineering (ISESE'02). Nara, Japan, pp. 161-172, 2002.

[16] Pargas RP, Harrold MJ and Peck RR. "Test-Data Generation Using Genetic Algorithms". Software Testing, Verification and Reliability, vol., nº 9, pp. 263-282, 1999.

[17] Do H, Elbaum SG and Rothermel G. "Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact". Empirical Software Engineering: An International Journal, vol. 10, nº 4, pp. 405-435, 2005.

Reseña sobre el taller ATSE'09 (Workshop on Automating Test Case Design, Selection and Evaluation)

Tanja Vos
Departmento de Sistemas de Información y Computación
Universidad Politécnica de Valencia
tvos@dsic.upv.es

Más de 50 asistentes acudieron a la I edición del taller sobre el Diseño, la Selección y la Evaluación Automatizada de Casos de Testeo (ATSE2009) organizados por la Universidad Politécnica de Valencia y la Universidad de Utrecht, y patrocinados por el proyecto EU EvoTest (FP6-2006-IST-33742), Berner&Mattner (http://www.berner-mattner.com) and Parasoft (www.parasoft.com). El evento se celebro el día 19 de Junio de 2009. Póvoa de Varzim, Portugal, junto con la 4ª Conferencia Ibérica de Sistemas y Tecnologías de la Información, CISTI 2009 (http://www.aisti.eu/cisti2009).

El objetivo de estas jornadas, eran proporcionar a los investigadores y a los profesionales un foro para el intercambio de ideas, experiencias, la comprensión de los problemas, las visiones para el futuro, y prometer soluciones para los problemas de generación automática de casos de prueba, la selección y la evaluación. El taller también ha proporcionado una plataforma para los investigadores y desarrolladores de herramientas de prueba para trabajar juntos e identificar los problemas en la teoría y la práctica del software de automatización de pruebas y establecer una agenda y las bases para el futuro desarrollo.

La temática de las jornadas, como indica su titulo, eran la automatización del diseño de casos de testeo. A pesar de que actualmente están disponibles muchas herramientas de automatización de pruebas para ayudar a la planificación y control de las mismas, como la ejecución de casos de prueba y la monitorización, todas estas herramientas comparten una filosofía similar dirigida hacia el diseño de los casos de prueba, la selección de los datos de prueba y la evaluación de la prueba. Esto deja algo crucial para el *tester* humano. Hay una razón; el diseño y la evaluación de las pruebas son difíciles de automatizar con las técnicas disponibles en la industria actual. El dominio de las posibles entradas (potenciales casos de test), incluso para un programa trivial, suele ser demasiado grande para ser estudiado exhaustivamente. En consecuencia, uno de los principales retos asociados con el diseño de

los casos de prueba es el de la selección de casos de prueba que sean eficaces en la búsqueda de fallos sin la necesidad de llevar a cabo un excesivo número de pruebas.

Imagénes de la celebración del taller ATSE'09

El programa de las jornadas consistía de una *keynote* de Joachim Wegener (Berner & Mattner, Germany) sobre las técnicas y herramientas que se utiliza en el sector de automoción para la automatización del diseño de las pruebas. Después, Willen Coppes (Parasoft, Los Paises Bajos) habló de las herramientas que ofrece Parasoft para la automatización de testeo. A continuación se incluyeron las presentaciones de Pedro Reales, explicando una herramienta para reducir el tamaño de los *suites* de casos de prueba, y de Christian Ekiza Lujua hablando de las ventajas y desventajas del testeo dirigido por *keywords*.

REICIS

ACTUALIDAD INVITADA

Las metodologías ágiles como garantía de calidad del software

José Ramón Díaz
Grupo de coordinación de Agile-Spain
jrramon@gmail.com

Introducción

En la actualidad, las metodologías ágiles de desarrollo de software están posicionándose fuertemente en las empresas más importantes del sector. Microsoft, IBM o Nokia por poner un ejemplo. Lamentablemente en España parece que arrancamos con retraso, aun cuando desde algunos sectores creemos que pueden ser un valor añadido clave en la creación de valor para el negocio y fomento de la innovación en el software.

Las metodologías ágiles contemplan el desarrollo de software de manera integral, con un énfasis especial en la entrega de valor al cliente, en la generación de negocio y el retorno de la inversión (ROI). Sólo hay una manera efectiva de crear software que funcione, y es de manera colaborativa. La colaboración entre cliente y desarrolladores es indispensable: se debe fomentar y apoyar. El software puede ser visto como un juego colaborativo (http://alistair.cockburn.us/Software+development+as+a+cooperative+game), y ahí hacen especial énfasis las metodologías ágiles, promoviendo procesos y métodos que faciliten esta colaboración. Se permite a los desarrolladores expandir su aportación de valor a los proyectos, y se ofrece a los clientes transparencia sobre los mismos.

La literatura clásica sobre desarrollo de software habla del triángulo de hierro formado por el alcance, costo y duración de un proyecto. Este triángulo siempre ha presupuesto que la calidad era inherente a los desarrollos, sin embargo, ha resultado ser la variable que más sufre en numerosos proyectos. La gestión tradicional de proyectos fija un alcance a desarrollar en un determinado tiempo con un costo establecido. Con esas tres

variables fijas, ante cualquier problema, la respuesta más probable será que la calidad disminuye: las prisas por terminar en plazo, por no superar el gasto permitido o cumplir el contrato firmado no dan muchas más opciones.

La calidad no es negociable

En estos momentos de mayor competencia la calidad juega un papel muy importante como ventaja competitiva ante competidores y clientes. Las metodologías ágiles nos están proporcionando un marco en el que lograr una calidad satisfactoria es parte integral del proceso de desarrollo.

Las herramientas que nos están poniendo a disposición de los desarrolladores son el punto de entrada a un cambio de paradigma que aumentará la calidad de los desarrollos. Técnicas como el desarrollo guiado por las pruebas (TDD), y aún más otro concepto que se está instaurando, el desarrollo guiado por las pruebas de aceptación (ATDD), serán las piezas fundamentales sobre las que se pueda elaborar un producto garantizando su integridad y calidad durante todo su ciclo de vida.

Se busca integrar el control de la calidad en el propio proceso de desarrollo. Es más, se busca que la única posibilidad de desarrollo sea creando cosas que funcionen correctamente, que cumplan con una definición de producto acabado en la que participan, colaboratívamente, el equipo de desarrollo y el cliente o dueño de producto. Se integra en el equipo a cualquier persona involucrada, como pueden ser personas responsables del testeo de software, pues comparten el mismo objetivo que el resto del equipo.

Otra cuestión fundamental es el cumplimiento de las expectativas del cliente. Por ello la creación del software en iteraciones y de manera incremental, base de todas las metodologías ágiles, permite alinear esas expectativas con el avance del proyecto. Uno de los principios básicos define que el grado de progreso de un proyecto únicamente se mide por el software creado que funciona. Es decir, software que ya proporciona valor al cliente, y que es potencialmente utilizable por él.

Los retos actuales

Actualmente la puerta de entrada a las metodologías ágiles de muchas empresas es Scrum. Se trata de una colección de procesos pensada para la gestión de proyectos que permite

centrarse en la entrega de valor al cliente y la potenciación del equipo para lograr su máxima eficiencia, dentro de un esquema de mejora continua. Personalmente creo que la verdadera revolución vendrá de la asimilación de los principios del Lean (www.poppendieck.com/papers/LeanThinking.pdf). Lean trata de aplicar los principios que revolucionaron la industria, provenientes de Toyota, y que se han trasladado al desarrollo de software.

Es una colección de seis principios que busca eliminar los trabajos que no generen valor para el cliente, que minimiza la deuda técnica, y que favorece una organización en busca de la mejora continua y facilite la labor de los equipos. No se trata de unas recetas a aplicar, si no de conceptos cuya implementación puede ser muy diferente según la casuística de las organizaciones. Es por eso por lo que son válidos universalmente.

Dónde estamos

Las metodologías ágiles cuentan ya con una amplia literatura y muchas experiencias reales de implantación (www.infoq.com/agile). Existen importantes conferencias a nivel internacional, donde se contrastan métodos y procesos relacionados con las mejoras en el desarrollo de software con aplicación de estas metodologías. Sin embargo, en España, da la impresión de que las únicas armas con las que se cuenta para mejorar la calidad del software son las certificaciones, impulsadas actualmente por la administración pública. Sin duda son un buen marco de referencia para numerosas empresas, pero también se debe conocer otros métodos que están ayudando a muchas empresas a mejorar la satisfacción del cliente, crear más negocio alrededor del desarrollo de software, y muy importante también, mejorar el trabajo de muchos desarrolladores, verdadero corazón de nuestra industria. Los conocimientos y las herramientas adquiridos por estos son el punto neurálgico de la calidad del software.

En España han empezado algunas iniciativas como ScrumManager (www.scrummanager.net/), compartiendo información, o Agile-Spain (www.agile-spain.com), que promueve el uso de las metodologías ágiles. La comunidad alrededor de Agile-Spain, por ejemplo, intenta dar a conocer y difundir estos conocimientos, con material en castellano, y empezando a promover eventos en España.

Perfil profesional

José Ramón Díaz es Ingeniero Informático, y MBA por la UNED. Tiene más de 10 años de experiencia en el desarrollo de software. Empezó a trabajar hace tres años con metodologías ágiles implantándolas en Biko2 en proyectos concretos. Hoy es responsable de la unidad de negocio "Software Factory 2.0" de dicha empresa. Es miembro del Grupo de Coordinación de la asociación de Agile-Spain y está involucrado en la organización del evento Open Space.