JISBD 2008

13th Conference on Software Engineering and Databases

XIII Jornadas de Ingeniería del Software y Bases de Datos

Gijón (Spain), October 7-10 2008

EDITORS: Ana Moreira
María José Suárez-Cabal
Claudio de la Riva
Javier Tuya

13th Conference on Software Engineering and Databases

XIII Jornadas de Ingeniería del Software y Bases de Datos

Gijón (Spain), October 7-10 2008

EDITORS: Ana Moreira
María José Suárez-Cabal
Claudio de la Riva
Javier Tuya

Edita:
Ana Moreira
María José Suárez-Cabal
Claudio de la Riva
Javier Tuya

Filmación e impresión:
Gráficas Rigel

Depósito Legal:
AS - 5.236 - 08

ISBN:
978-84-612-5820-8

Volume Editors Details

Ana Moreira
Departamento de Informática
Faculdade de Ciências e tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
E-mail: amm@di.fct.unl.pt
URL: http://ctp.di.fct.unl.pt/~amm/

María José Suárez-Cabal
Departamento de Informática
Universidad de Oviedo
33204 Gijón, Spain
E-mail: cabal@uniovi.es

Claudio de la Riva
Departamento de Informática
Universidad de Oviedo
33204 Gijón, Spain
E-mail: claudio@uniovi.es
URL: http://www.di.uniovi.es/~claudio/

Javier Tuya
Departamento de Informática
Universidad de Oviedo
33204 Gijón, Spain
E-mail: tuya@uniovi.es
URL: http://www.di.uniovi.es/~tuya/

Preface

Celebrating 13 Years of JISBD

With the 2008 edition in Gijón (October 7 to 10), the Conference on Software Engineering and Databases (JISBD) celebrates 13 years of existence. Born as a forum where the Spanish community would publish their work, meet to discuss potential research collaborations and evaluate the progress of research projects funded by the Spanish Ministery of Science and Technology, JISBD has long since moved beyond its initial boundaries and crossed several oceans.

Presently, the conference has become an important reference for younger researchers, as well as a forum which the more experienced do not wish to miss. In recent years, JISBD has broadened its radius, accepting papers also in English and Portuguese, in addition to Spanish. This change, not only brought more conference participants, but also significantly increased the number of submissions and, principally, the quality of the submissions accepted.

The JISBD community is now self-sustained and continues to expand. The quality of work accepted is equivalent to that of other relevant international events. In recent years, it has been possible to edit a special volume of IEEE LA with extended versions of the best conference papers and this also is happening with the current edition. This special issue, together with the conference proceedings with ISBN, is a showcase of the quality of the work of JISBD.

One of the highlights of this conference has been the excellence of its keynote speakers. Many of the most admired international researchers and professionals have already been invited to address the JISBD participants.

Within this rich framework for scientific and technological interchange, the conference includes several satellite events. In addition to the presentation of high quality original papers in the main conference, the program includes tutorials, tool demonstrations and workshops for the discussion of innovative ideas and work in progress, as well as a forum to bring to a wider audience research work already published in prestigious journals or conference proceedings (with an acceptance rate below 25% and an impact factor above 0.5).

It is no exaggeration to claim that JISBD has been consolidating its position as a reference event where researchers and professionals of Software Engineering and Databases can get together to discuss results and share ideas. JISBD has become an important forum for collaboration between different strands and research groups, while continuing to offer its participants a well organized event with exceptional hospitality.

About this Edition

The increased global reach of JISBD is evident in the origin of papers received. This year, in addition to the two Iberian and ten Latin-American countries, submissions arrived also from China, France, Germany, India, Iran and Pakistan.

Of a total of 115 abstracts, 112 papers were submitted for review. Most papers were reviewed by three PC members, and several were reviewed by four. The program Committee accepted 30 full papers and selected 12 for presentation as short papers. The acceptance rate for full papers was approximately 25%.

The increasing success of the conference implies greater responsibilities in terms of guaranteeing independent judgement and ensuring compliance with international standards of ethics. For this reason, a greater effort has been made in recent years to avoid double submissions, a task made

more difficult by the fact that the conference accepts submissions in three languages. This year, three good papers were rejected due to double submission, in different languages to different events.

In addition to the accepted papers, the conference includes five workshops, one tutorial, nine tool demos, an industrial panel and also a forum to discuss important relevant work already published elsewhere.

A highlight of the conference is, without doubt, the excellence of the invited keynote speakers. This year is no exception and we are honoured indeed to receive Bashar Nuseibeh and Bran Selic.

Bashar Nuseibeh is an academic and researcher at the Open University in the UK and invited professor in various other universities, including Japan's National Institute of Informatics. Bashar chairs several international committees and is recognized also for industry work, including organizations such as the UK's National Air Traffic Services (NATS), Texas Instruments, Praxis Critical Systems, Philips Research Labs, and NASA.

Bran Selic was, for many years, a distinguished engineer and researcher at IBM, and currently heads a global consultancy based in Canada. He is internationally known for his work in large-scale industrial systems, and for his pioneering work in Model-Driven Development and Real-Time Embedded Systems.

Bahsar's keynote is entitled "*The five W's (and one "H") of Security: ... Software Engineering of Secure Systems"* while Bran's is on "*Model-Based Software Engineering: Expected and Unexpected Challenges*".

Acknowledgements

A very special word of thanks is due to Bashar and Bran for having accepted my invitation and for sharing all the participants their knowledge, experience and refined wit. I sincerely hope JISBD was also for them a gratifying and unique experience.

Acknowledgements are due to a multitude of collaborators without whom the conference could not have been a success. Firstly, the paper authors for the trust placed in the quality of JISBD as a conference that merited their submissions. Secondly, to the PC members, whose diligent review work ensured that the authors' trust continues to be justified.

For managing the submission and review process, I was fortunate to have the constant help of Juan Hernández and José Javier Berrocal; they were my guardian angels, constantly alert to deadlines and ready to help as necessary. A special thanks for the contribution of my "Executive Program Committee", Antonio Vallecillo, Juan Hernández, Miguel Toro, Vicente Pelechano and Xavier Franch.

Acknowledgement is due to the main conference organizers, especially to Javier Tuya and his co-chair, Claudio de la Riva, for their efficient handling of the numerous tasks that a conference of this size and quality entails. Thanks also to those responsible for the satellite events (in alphabetical order), António Rito Silva, Antonio Vallecillo, Gustavo Rossi, João Araújo, João Falcão e Cunha, José Berrocal, José Corrales, José García-Fanjul, João Miguel Fernandes, Lidia Fuentes and María José Suárez-Cabal.

Finally, a special word of thanks to the sponsors of this conference, without whose contribution the event would have been somewhat less charming (not to mention gastronomically less satisfying).

Ana Moreira
Program Committee Chair

Prefácio

Celebrando 13 Anos de JISBD

Com a edição de 2008 em Gijón (7-10 Outubro), a Conferência em Engenharia de Software e Bases de Dados (JISBD) celebra 13 anos de existência. Apesar de ter nascido como um fórum onde a comunidade espanhola publicava os seus trabalhos e se reunia para discutir potenciais colaborações futuras de investigação, e até avaliar o estado de andamento dos projectos de investigação financiados pelo Ministério da Ciência e Tecnologia espanhola, há muito que extravasou essas fronteiras e cruzou oceanos.

Actualmente o JISBD é um marco importante na investigação dos mais jovens, mas também um fórum que os mais seniores não querem perder. Nos últimos anos a conferência abriu-se para o mundo inteiro, aceitando artigos escritos em Inglês, Espanhol e Português. Esta viragem trouxe não só mais participantes à conferência, mas também um aumento significativo do número de trabalhos submetidos e, principalmente, um aumento na qualidade desses trabalhos.

A comunidade do JISBD é agora auto-sustentada e em contínua expansão. A qualidade dos trabalhos aceites é equiparada à de muitos outros eventos internacionais de relevo. Por este motivo, nos últimos anos, foi-nos possível editar um volume especial no IEEE LA com uma versão estendida dos melhores trabalhos da conferência, o que acontecerá também nesta edição. Este volume, em conjunto com as actas formais da conferência com ISBN, é uma montra da qualidade do trabalho que aqui se discute.

Uma das características de excelência desta conferência tem sido, desde sempre, o gabarito dos seus palestrantes convidados. É um prazer ver que muitos dos mais admirados investigadores e profissionais internacionais já foram convidados a falar para os participantes do JISBD.

Neste enquadramento fecundo para divulgação científica e tecnológica, a conferência inclui vários eventos satélite. Além dos artigos seleccionados para apresentação na conferência, o programa inclui ainda tutoriais, demonstrações de ferramentas, workshops para discussão de ideias inovadores e trabalhos em andamento, assim como um evento para a disseminação de trabalho de investigação já publicado em revistas e actas de conferências de grande prestígio (onde o índice de aceitação é inferior a 25% e o factor de impacto superior a 0.5).

Assim, não é excessivo afirmar que o JISBD se tem vindo a consolidar como um evento de referência onde investigadores e profissionais em Engenharia de Software e Bases de Dados se encontram para discutir, disseminar e trocar ideias, partilhar experiências e resultados entre diversos sectores e grupos de investigação, num contexto de excelente organização e invulgar hospitalidade.

Sobre esta Edição

A atestar o crescimento e internacionalização do JISBD está a origem dos artigos que nos chegaram. Este ano, a nacionalidade dos autores foi surpreendentemente diversificada, pois para além dois países Ibéricos e de dez países Latino-Americanos, recebemos trabalhos também da Alemanha, China, França, Índia, Irão e Paquistão.

O número total de resumos foi de 115, sendo que destes, 112 artigos foram submetidos para avaliação. Cada artigo foi avaliado por pelo menos três revisores, sendo que vários foram avaliados por quatro. O Comité de Programa aceitou 30 artigos longos e escolheu 12 para apresentação como artigos curtos. Assim, o índice de aceitação de artigos longos foi de cerca de 25%.

Este sucesso acarreta responsabilidades acrescidas em garantir a independência de julgamentos e em fazer cumprir a ética e as normas internacionais. É por este motivo que, nos últimos anos, se

tem feito um esforço muito grande para evitar submissões duplicadas, tarefa nem sempre fácil para os membros do Comité de Programa, já que a conferência aceita três línguas de escrita. Este ano foram rejeitados três bons artigos avaliados como de submissão duplicada, em duas línguas, para eventos diferentes.

Para além dos artigos seleccionados, a conferência conta também com a organização de cinco *workshops*, um *tutorial*, nove demonstrações de ferramentas, um painel industrial e ainda um fórum onde se discutem trabalhos de relevo já publicados em revistas ou outras conferências.

Mas sem dúvida que os momentos mais altos da conferência são sempre marcados pelo admirável conjunto de palestrantes convidados. Este ano tivemos a sorte de receber Bashar Nuseibeh e de Bran Selic.

Bashar Nuseibeh é um académico e investigador da Open University, na Inglaterra, e professor convidado em várias outras universidades, incluindo o Instituto Japonês de Informática. Bashar preside vários comités internacionais e é admirado também pelo seu trabalho para a indústria, que inclui organizações como o National Air Traffic Services (NATS) do Reino Unido, Texas Instruments, Praxis Critical Systems, Philips Research Labs, e a NASA.

Bran Selic foi durante umas dezenas de anos engenheiro e investigador distinguido da IBM e actualmente preside uma empresa de consultoria internacional sediada no Canadá. É conhecido mundialmente pelos seus trabalhos em sistemas de larga escala industrial e também pelo seu pioneirismo nas áreas de desenvolvimento orientado a modelos e sistemas embutidos de tempo real.

A palestra do Bashar é intitulada "*The five W's (and one "H") of Security: ... Software Engineering of Secure Systems"*, enquanto que a do Bran é sobre "*Model-Based Software Engineering: Expected and Unexpected Challenges*".

Agradecimentos

Uma palavra especial de agradecimento ao Bashar e ao Bran por terem aceite o meu convite e por brindarem todos os participantes com a sua experiência, conhecimento e refinado sentido de humor. Espero que o JISBD tenha sido também para eles uma experiência agradável e diferente.

Agradecimentos são justamente devidos ao grande número de colaboradores, sem o contributo dos quais, a conferência não poderia ter tido êxito. Aos autores, claro, por confiarem na qualidade do JISBD e submeterem, por isso, os seus trabalhos. Aos membros do Comité de Programa cujas revisões asseguram que essa confiança continua a justificar-se.

Para gerir o sistema de submissão, contei com o apoio incondicional do Juan Hernández e do José Javier Berrocal. Eles foram os meus "anjos da guarda", sempre atentos a todos os prazos e prontos a dar todas as explicações. Um agradecimento particular ao contributo meu "Comité Executivo de Programa", Antonio Vallecillo, Juan Hernández, Miguel Toro, Vicente Pelechano e Xavier Franch. Obrigada pelo vosso apoio e sugestões.

Obrigada aos organizadores principais da conferência, em especial ao Javier Tuya, e ao seu vice-presidente, Claudio de la Riva, pela gestão eficaz das inúmeras tarefas que uma conferência desta dimensão exige. Um agradecimento é ainda devido, e por ordem alfabética, aos responsáveis dos eventos satélite, António Rito Silva, Antonio Vallecillo, Gustavo Rossi, João Araújo, João Falcão e Cunha, José Berrocal, José Corrales, José García-Fanjul, João Miguel Fernandes, Lidia Fuentes e María José Suárez-Cabal.

Finalmente, um agradecimento aos patrocinadores da conferência, sem o contributo de quem o evento teria tido menos charme (e uma gastronomia muito menos requintada).

Ana Moreira
Presidente do Comité de Programa

Prefacio

Con esta edición 2008 en Gijón (7 al 10 de Octubre), las Jornadas de Ingeniería del Software y Bases de Datos (JISBD) celebra 13 años de existencia. JISBD nació como un foro donde la comunidad española publicaba su trabajo, discutía potenciales colaboraciones en investigación y evaluaba el progreso de los proyectos de investigación financiados por el Ministerio de Ciencia y Tecnología, y en la actualidad ha traspasado fronteras y cruzado varios océanos.

Actualmente, la conferencia es una referencia importante para jóvenes investigadores, así como un foro de cita obligada para investigadores más experimentados. Durante los últimos años, JISBD se ha abierto al mundo, aceptando artículos en Inglés y Portugués, además de Castellano. Este cambio no solamente se ha traducido en más participantes, sino que ha incrementado significantemente el número de artículos enviados, y principalmente, la calidad de los artículos aceptados.

La comunidad JISBD está actualmente auto sustentada y continúa expandiéndose. La calidad de los trabajos aceptados es equivalente al de otros eventos internacionales relevantes. Durante los últimos años, ha sido posible editar un volumen especial de IEEE LA con versiones ampliadas de los mejores trabajos presentados en la conferencia, lo que sucederá también en la presente edición. Este volumen especial, junto con las actas de la conferencia con ISBN, es una muestra de la calidad de los trabajos de JISBD.

Una de las características más sobresalientes de la conferencia ha sido la calidad de los ponentes invitados. Varios investigadores y profesionales de reconocido prestigio internacional han sido invitados a participar como ponentes en JISBD.

Dentro de este marco científico y tecnológico, la conferencia incluye varios eventos relacionados. Además de la presentación de artículos originales de alta calidad en la conferencia principal, el programa incluye tutoriales, demostraciones de herramientas, talleres para la discusión de ideas innovadoras y trabajos en curso, así como la divulgación de trabajos de investigación publicados en revistas y conferencias de prestigio (con un ratio de aceptación por debajo del 25% y un factor de impacto por encima de 0,5).

No es una exageración afirmar que JISBD ha consolidado su posición como un evento de referencia donde investigadores y profesionales de la Ingeniería del Software y las Bases de Datos se reúnen para discutir resultados y compartir ideas. JISBD se ha convertido en un foro importante para la colaboración entre diferentes sectores y grupos de investigación, en un contexto de excelente organización y excepcional hospitalidad.

Sobre la presente edición

El crecimiento e internacionalización de JISBD se hace evidente analizando el origen de los artículos recibidos. En la presente edición, además de los artículos recibidos de los dos países de la Península Ibérica y los diez países Latinoamericanos, se han recibido artículos de China, Francia, Alemania, India, Irán y Pakistán.

De un total de 115 resúmenes previamente recibidos, finalmente se recibieron 112 artículos para su revisión. La mayoría de los artículos fueros revisados por tres miembros del Comité de Programa y varios por cuatro. El Comité de Programa aceptó 30 artículos largos y seleccionó 12 para su presentación como artículos cortos. El ratio de aceptación para los artículos largos fue de aproximadamente el 25%.

El éxito de la conferencia implica grandes responsabilidades en términos de garantizar la independencia de las revisiones y el cumplimiento de los estándares internacionales de ética. Por esta razón, durante los últimos años se ha realizado un mayor esfuerzo en aras de evitar envíos duplicados, una tarea especialmente dificultosa, ya que la conferencia acepta envíos en tres idiomas. En la

presente edición tres artículos fueron rechazados debido al doble envío en diferentes idiomas para diferentes eventos.

Además de los artículos aceptados, la conferencia incluye cinco talleres, un tutorial, nueve demostraciones de herramientas y foro para la discusión y divulgación de trabajos relevantes previamente publicados, así como una mesa redonda de carácter industrial.

Una característica importante de la conferencia es, sin ninguna duda, la excelencia de los ponentes invitados. La presente edición no es una excepción y estamos orgullosos de contar con la presencia de Bashar Nuseibeh y Bran Selic.

Bashar Nuseibeh es académico e investigador en la Open Unversity del Reino Unido y profesor invitado en otras muchas universidades, incluyendo el Instituto Nacional Japonés de Informática. Bashar preside varios comités internacionales y está reconocido igualmente por su trabajo industrial, incluyendo organizaciones tales como el Servicio Nacional de Tráfico Aéreo del Reino Unido (NATS), Texas Instruments, Praxis Critical Systems, Philips Research Labs y la NASA.

Bran Selic fué durante varios años un destacado ingeniero e investigador en IBM y actualmente lidera un consultora internacional con sede en Canadá. Es internacionalmente conocido por su trabajo en sistemas industriales a gran escala y por su trabajo pionero en Desarrollo Dirigido por Modelos y Sistemas Empotrados en Tiempo Real.

La conferencia de Bashar se titula "*The five W's (and one "H") of Security: ... Software Engineering of Secure Systems"* y la de Bran "*Model-Based Software Engineering: Expected and Unexpected Challenges*".

Agradecimientos

Un agradecimiento especial es para Bashar y Bran por haber aceptado mi invitación y por compartir con todos los participantes sus conocimientos, experiencia y refinado sentido del humor.

Agradecimientos también para la multitud de colaboradores sin los cuales el éxito de la conferencia no habría sido posible. En primer lugar, para los autores de los artículos por confiar en la calidad de JISBD y enviar sus trabajos. En segundo lugar, para los miembros del Comité de Programa, cuyas revisiones aseguran la calidad de los trabajos.

Para el proceso de gestión y revisión de los trabajos recibidos, fui afortunada por tener la ayuda constante de Juan Hernández y José Javier Berrocal. Ellos fueron mis ángeles guardianes, alertándome constantemente de las fechas límite y siempre preparados para ayudarme cuando lo necesitaba. Agradecimientos especiales por la contribución de mi "Comité de Programa Ejecutivo", Antonio Vallecillo, Juan Hernández, Miguel Toro, Vicente Pelechano y Xavier Franch.

Agradecimientos también para los organizadores de la conferencia principal, especialmente al presidente del comité organizador Javier Tuya y su vicepresidente Claudio de la Riva, por su manejo eficiente de las numerosas tareas que una conferencia de este tamaño y calidad conllevan. Agradecimientos también para los responsables de los eventos relacionados (en orden alfabético) António Rito Silva, Antonio Vallecillo, Gustavo Rossi, João Araújo, João Falcão e Cunha, José Berrocal, José Corrales, José García-Fanjul, João Miguel Fernandes, Lidia Fuentes y María José Suárez-Cabal.

Finalmente, palabras especiales de agradecimiento para los patrocinadores de la conferencia, sin cuya contribución el evento habría sido menos encantador (y con una gastronomía menos refinada).

Ana Moreira
Presidenta del Comité de Programa

Conference Committee

Program Committee Chair
Ana Moreira (Univ. Nova de Lisboa, Portugal)

Organizing Chair
Javier Tuya (Univ. Oviedo, Spain)

Organizing Co-Chair
Claudio de la Riva (Univ. Oviedo, Spain)

Permanent Committee Secretary
Mario Piattini (Univ. Castilla-La Mancha, Spain)

Tutorial Chair
António Rito Silva (Univ. Técnica Lisboa, Portugal)

Workshop Chair
João Araújo (Univ. Nova de Lisboa, Portugal)

Tool Demonstrations Chair
Lidia Fuentes (Univ. Málaga, Spain)

Relevant Papers Dissemination Chairs
Antonio Vallecillo (Univ. Málaga, Spain)
João Falcão Cunha (Univ. Porto, Portugal)

Proceedings Chair
María José Suárez-Cabal (Univ. Oviedo, Spain)

Cyber Chair
Jose Javier Berrocal (Univ. Extremadura, Spain)

Web Chair
José A. Corrales (Univ. Oviedo, Spain)

Publicity Chairs
Gustavo Rossi (Univ. La Plata, Argentina)
José García-Fanjul (Univ. Oviedo, Spain)
João Miguel Fernandes (Univ. Minho, Portugal)

Organizing Committee (Univ. Oviedo, Spain)

Javier Tuya
Claudio de la Riva
José García-Fanjul
Isabel Sevilla
María José Suárez-Cabal
José Ramón de Diego
Raquel Blanco
Eugenia Díaz Fernández
José A. Corrales
Marta Fernández de Arriba

SISTEDES Executive Board

President
Miguel Toro (Univ. Sevilla, Spain)

Vice President
Juan José Moreno (Univ. Polit. Madrid, Spain)

Secretary
Nieves R. Brisaboa (Univ. Coruña, Spain)

Treasurer
Javier Tuya (Univ. Oviedo, Spain)

Members
Pere Botella (Univ. Polit. Catalunya, Spain)
Ricardo Peña (Univ. Complutense Madrid, Spain)
Coral Calero (Univ. Castilla-La Mancha, Spain)
Manuel Hermenegildo (Univ. Polit. Madrid, Spain)
Ernesto Pimentel (Univ. Málaga, Spain)
María Ribera Sancho (Univ. Polit. Catalunya, Spain)
Natalia Juristo (Univ. Polit. Madrid, Spain)
Salvador Lucas (Univ. Polit. Valencia, Spain)

Submission and Review Support System
(Quercus Software Engineering Group)

Javier Berrocal (Univ. Extremadura, Spain)
Juan Hernández (Univ. Extremadura, Spain)

Secretariat

Fundación Universidad de Oviedo
C/ Principado 3, 4ª Planta
33007 Oviedo, Spain.
Tel: 34-985104927
Fax: 34-985104928

Executive Program Committee

Xavier Franch (Univ. Polit. Catalunya, Spain)
Juan Hernández (Univ. Extremadura, Spain)
Vicente Pelechano (Univ. Polit. Valencia, Spain)
Antonio Vallecillo (Univ. Málaga, Spain)
Miguel Toro (Univ. Sevilla, Spain)
Javier Tuya (Univ. Oviedo, Spain)

Program Committee

Albert Abelló (Univ. Polit. Catalunya, Spain)
Ana Paula Afonso (Univ. Lisboa, Portugal)
Ademar Aguiar (Univ. Porto, Portugal)
Jesús Aguilar (Univ. Sevilla, Spain)
José Aldana (Univ. Málaga, Spain)
Mauricio Alférez (U. Nova de Lisboa, Portugal)
Bárbara Álvarez (Univ. Polit. Cartagena, Spain)
Raquel Anaya (Univ. EAFIT, Colombia)
María José Aramburu (Univ. Jaume I, Spain)
Hernán Astudillo (U. T. Federico Santa María, Chile)
Orlando Belo (Univ. do Minho, Portugal)
Rafael Berlanga (Univ. Jaume I, Spain)
Paulo Borba (Univ. Federal Pernambuco, Brazil)
Pere Botella (Univ. Polit. Catalunya, Spain)
Rosana Braga (Univ. São Paulo, Brazil)
Nieves Brisaboa (Univ. Coruña, Spain)
Isabel Brito (Inst. Polit. Beja, Portugal)
Fernando Brito e Abreu (U. Nova de Lisboa, Portugal)
Coral Calero (Univ. Castilla-La Mancha, Spain)
Marcelo Campo (UNICEN, Argentina)
Carlos Canal (Univ. Málaga, Spain)
Valeria de Castro (Univ. Rey Juan Carlos, Spain)
Matilde Celma (Univ. Polit. Valencia, Spain)
Christina Chávez (Univ. Bahia, Brazil)
Rafael Corchuelo (Univ. Sevilla, Spain)
Dolors Costal (Univ. Polit. Catalunya, Spain)
Yania Crespo (Univ. Valladolid, Spain)
Carlos Delgado (Univ. Carlos III, Spain)
Oscar Díaz (Univ. País Vasco, Spain)
Javier Dolado (Univ. País Vasco, Spain)
Xavier Franch (Univ. Polit. Catalunya, Spain)
Pablo de la Fuente (Univ. Valladolid, Spain)
Mario Gaspar da Silva (Univ. Lisboa, Portugal)
Alessando García (Univ. Lancaster, UK)
Marcela Genero (Univ. Castilla-La Mancha, Spain)
Cristina Gómez (Univ. Polit. Catalunya, Spain)
Jaime Gómez (Univ. Alicante, Spain)
Alfredo Goñi (Univ. País Vasco, Spain)
Silvia Gordillo (UNLP, Argentina)
Pedro Guerreiro (Univ. Algarbe, Portugal)
Juan Hernández (Univ. Extremadura, Spain)
Jon Iturrioz (Univ. País Vasco, Spain)
Elena Jurado (Univ. Extremadura, Spain)
Natalia Juristo (Univ. Polit. Madrid, Spain)
Miguel Katrib (Grupo WEBOO, Cuba)
María Lencastre (Univ. Pernambuco, Brazil)
Antonia Lopes (Univ. Lisboa, Portugal)
Adolfo Lozano (Univ. Extremadura, Spain)
Esperanza Marcos (Univ. Rey Juan Carlos, Spain)
Henrique Madeira (Univ. Coimbra, Portugal)
Eduardo Mena (Univ. Zaragoza.Spain)
Ana María Moreno (Univ. Polit. Madrid, Spain)
Juan José Moreno (Univ. Polit. Madrid, Spain)
Juan Manual Murillo (Univ. Extremadura, Spain)
Oscar Pastor (Univ. Polit. Valencia, Spain)
Vicente Pelechano (Univ. Polit. Valencia, Spain)
Marcelo Pimenta(Univ. F. Rio Grande do Sul, Brazil)
Ernesto Pimentel (Univ. Málaga, Spain)
Mónica Pinto (Univ. Málaga, Spain)
Ángeles Places (Univ. Coruña, Spain)
Antonio Polo (Univ. Extremadura, Spain)
Claudia Pons (UNICEN, Argentina)
Tom Price (Univ. F. Rio Grande do Sul, Brazil)
Carme Quer (Univ. Polit. Catalunya, Spain)
Celia Ramos (Univ. Algarbe, Portugal)
Isabel Ramos (Univ. Sevilla, Spain)
Isidro Ramos (Univ. Polit. Valencia, Spain)
Claudio de la Riva (Univ. Oviedo, Spain)
José Riquelme (Univ. Sevilla, Spain)
José Luis Roda (Univ. La Laguna, Spain)
María José Rodríguez Fortis (Univ. Granada, Spain)
José Raúl Romero (Univ. Córdoba, Spain)
Antonio Ruiz (Univ. Sevilla, Spain)
Francisco Ruiz (Univ. Castilla-La Mancha, Spain)
José Samos (Univ. Granada, Spain)
Fernando Sánchez (Univ. Extremadura, Spain)
Juan Sánchez (Univ. Polit. Valencia, Spain)
Carla Silva (Univ. F. Pernambuco, Brazil)
Ernest Teniente (Univ. Polit. Catalunya, Spain)
Miguel Toro (Univ. Sevilla, Spain)
Ambrosio Toval (Univ. Murcia, Spain)
Juan Carlos Trujillo (Univ. Alicante, Spain)
Toni Urpi (Univ. Polit. Catalunya, Spain)
Antonio Vallecillo (Univ. Málaga, Spain)
Belén Vela (Univ. Rey Juan Carlos, Spain)

Referees

Álvaro E. Prieto Ramos
Amador Durán Toro
André L. Santos
Andrea Delgado
Ángel Herranz
Angélica Caro
Anna Grimán Padua
Antonio Jesús Roa Valverde
Antônio Oliveira Filho
Antonio Ruiz-Cortés
Arturo Zambrano
Carlos Bobed
Carlos D. Barranco González
Carlos Enrique Cuesta Quintero
Carlos Neil
Cecilia Delgado Negrete
César J. Acuña
Claudio Sant`Anna
Cristina Vicente Chicote
Daniel Rodríguez
Dante Carrizo
Diana Marcela Sánchez
Diego Alonso Cáceres
Diego Seco Naveiras
Domingo Savio Rodríguez Baena
Eduardo Rodríguez López
Elisa Yumi Nakagawa
Ellen Francine Barbosa
Encarna Sosa Sánchez
Fernando Molina Molina
Fran J. Ruiz Bertol
Francisco Javier Lucas Martínez
Francisco Luís Gutiérrez Vela
Francisco Martínez Álvarez
Ignacio García Rodríguez de Guzmán
Ismael Caballero
Ismael Navas Delgado
Ismael Sanz Blasco
Javier Pérez García
Joaquín Lasheras
Joaquín Nicolás
Jorge Gracia
Jorge Martínez Gil
José María Cavero Barca
Juan Ángel Pastor Franco
Juan M. Vara
Juan Manuel Pérez Martínez
Manuel Ángel Serrano Martín
Manuel Resinas
Márcio de Medeiros Ribeiro
Marcirio Chaves
Marcos López Sanz
Mari Carmen Otero
María Esperanza Manso Martínez
María Luisa Rodríguez Almendros
María Teresa Gómez López
María Visitación Hurtado Torres
Martin Solari
Miguel Ángel Laguna Serrano
Miguel Ángel Martínez
Miguel Rodríguez Luaces
Mª Ángeles Moraga de la Rubia
Nuno Cardoso
Orlando Avila-García
Oscar Dieste
Óscar Pedreira Fernández
Othmane Chniber
Pablo Inostroza
Pablo Trinidad
Paloma Cáceres García de Marina
Pedro Sánchez Palma
Raquel M. Crespo García
Raquel Trillo Lado
Roberto Almeida Bittencourt
Roberto Rodríguez Echeverría
Roberto Ruiz
Rui Lopes
Sascha Ossowski
Sergio Ilarri Artigas
Vicente Luque Centeno

Sponsors


Universidad de Oviedo
400
cuarto centenario


PCTI
ASTURIAS


Ayuntamiento de Gijón


GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN


GOBIERNO DEL PRINCIPADO DE ASTURIAS


TIC
CLUSTER ASTURIAS


csc


InterSystems


cajAstur


DEPARTAMENTO DE INFORMATICA
UNIVERSIDAD DE OVIEDO


Sistedes
Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software


gijon.info

Table of Contents[1]

[1] The section headings below correspond to the conference program, but do not include all the presentations in each conference session (where short papers and dissemination papers on the same topic also were included). Thus, the sections here all contain fewer papers than the corresponding conference session; the short papers are listed separated in this volume, followed by a chapter with an overview of the dissemination papers.

Information Engineering

Model Engineering

Formal Methods

Maintenance and Testing

Data Mining, Data Streaming and Datawarehouses

Reengineering and Software Modernization

Quality, Measurement & Estimation of Products & Processes

Requirements Engineering

Keynote Address 2

Short Papers

Workshops, Tutorials, Demos and Dissemination

Generación automática de casos de prueba en líneas de producto

Pedro Reales Mateo, Beatriz Pérez Lamancha, Macario Polo Usaola

Escuela Superior de Informática
Universidad de Castilla-La Mancha
Paseo de la Universidad, 4
13071-Ciudad Real (Spain)
{pedro.reales, beatriz.plamancha,macario.polo}@uclm.es

Resumen. Este artículo describe un método para la generación de casos de prueba en contextos de líneas de producto. El trabajo hace un énfasis especial en la automatización de la obtención del oráculo, que es una línea de investigación candente en el área del *testing*.

Palabras Clave: Líneas de producto, oráculos, pruebas.

1. Introducción

En el contexto de la Ingeniería del Software, una línea de producto representa "un conjunto de sistemas software que comparten un conjunto amplio de características que satisfacen las necesidades específicas de un dominio o segmento del mercado y que se desarrollan de acuerdo a un conjunto común de "activos base" (*core assets*) de una manera predeterminada" [1]. De acuerdo con McGregor et al., los productos de una línea se caracterizan, por un lado, por su similitud con respecto a las características comunes y, por otro, por la diversidad que introduce cada producto con respecto a la línea [2]. Este doble caracterización permite llevar la reutilización a todos los niveles: mientras que la reutilización en sentido clásico tiene un sentido más "oportunista" [3], en el sentido de que se refiere a la construcción de piezas encapsuladas de software que tal vez puedan ser incorporadas en proyectos futuros, la idea de la reutilización en líneas de producto es un concepto más amplio, que engloba todo el proceso de construcción del software, dándole así a la reutilización un enfoque tanto proactivo [2] como predictivo [4].

Cada producto se distingue de la línea en una serie de puntos de variabilidad, con lo que un producto representa una variante de la línea. Así, para construir un producto a partir de una línea, se determinan sus requisitos particulares y, en función de éstos y de los requisitos de la línea, se determinan los puntos de variabilidad, que serán analizados y que podrán dar lugar a una cuantificación de la reutilización; entonces, se está en condiciones de pasar al diseño del producto, en donde se integran los elementos procedentes de la línea y los del propio producto; posteriormente suele identificarse una última etapa de de ensamblaje, en la que se implementa el producto final.

En el método de construcción sucintamente descrito en el párrafo anterior se pueden distinguir dos procesos de desarrollo, que suelen representarse de forma paralela:

- La Ingeniería de Dominio, referida al desarrollo de los elementos comunes [1].
- La Ingeniería de Producto, dedicada a la implementación de los productos concretos.

En el nivel de Ingeniería de Dominio se describen los requisitos de la línea, que serán reutilizados en la Ingeniería de Producto.

La variabilidad puede afectar a todos los artefactos software utilizados en la línea, llegando desde los requisitos hasta el código y afectando, por tanto, no sólo al producto final, sino a todos los intermedios [5]. De este modo, las líneas de producto permiten la reutilización de la mayor parte de los artefactos utilizados durante la Ingeniería de Dominio.

En este artículo se presenta un método para la generación automática de casos de prueba en líneas de producto. La idea parte de las posibilidades de reutilización a nivel de Ingeniería de Dominio, para generar casos de prueba que puedan ser utilizados en los diferentes productos.

Para esto, resulta fundamental mantener la coherencia y la trazabilidad entre los diferentes artefactos que se generan durante todo el desarrollo: así, en el caso ideal, los requisitos, casos de uso y resto de artefactos diseñados para la línea de producto servirían para construir casos de prueba para todos los productos. Si bien

existen diferentes técnicas y propuestas para automatizar la generación de casos en el contexto de las líneas de producto, en todas ellas queda pendiente el hecho de la manipulación del oráculo, que es una línea de investigación fundamental en el área del *testing*, de cara a conseguir la automatización de varias actividades del proceso de pruebas [6]. En este trabajo se presenta una contribución importante en este sentido. La sección 2 plantea algunos retos pendientes respecto del oráculo, así como un breve repaso a algunos trabajos relativos a las pruebas, y a las pruebas en líneas de producto. Las secciones 3 y 4 presentan las principales aportaciones de este trabajo. La sección 5 presenta algunas conclusiones, lecciones aprendidas y trabajos futuros.

2. Trabajos relacionados

Una de las principales dificultades con las que se encuentra la investigación en el área del *testing*, y que, de acuerdo con Bertolino [6], representa un gran obstáculo para avanzar en su automatización, es la descripción del oráculo. El oráculo es el mecanismo de que se dota a cada caso de prueba para determinar, tras su ejecución, si el sistema que se está probando supera o no el caso. No existe un mecanismo que permita describirlo de manera genérica y, en la práctica, se implementa siempre manualmente [7].

Para Baresi y Young [7] el "oráculo ideal" debería verificar la satisfacción de las características del sistema, pero tratando de evitar la sobreespecificación, puesto que esto podría llevar a que la descripción de un oráculo genérico pudiese suponer tanto esfuerzo como la escritura de múltiples oráculos individuales.

Bertolino [6] también hace referencia a este oráculo ideal, que describe como "un método mágico [sic] que proporciona las salidas para cada caso de prueba, aunque en la práctica se implemente como un motor o heurística que emite un veredicto de paso o fallo sobre las salidas observadas".

Se debe, por tanto, buscar un punto de compromiso a partir del cual el esfuerzo de describir un oráculo generalizable y aplicable a múltiples casos de prueba, compense respecto de la escritura de un oráculo por cada caso.

El trabajo citado de Baresi y Young, del año 2001, presenta el estado del arte en cuanto al problema del oráculo. La mayor parte de las propuesta analizadas se refieren a la inserción de instrucciones en los programas que realizan algún tipo de comprobación, como las *assert* de Java, macros de C o extensiones para lenguajes que permiten, mediante algún tipo de preprocesamiento, incluir en el código la comprobación de restricciones.

Otras propuestas utilizan descripciones formales de los programas, en donde se hace necesario distinguir entre el espacio de estados de los objetos de la clase, y un conjunto de reglas de reescritura que muestran cómo evoluciona el estado conforme se ejecutan operaciones sobre la instancia. Enfoques como éste han sido muy estudiados [8, 9]. Estas especificaciones, sin embargo, requieren mucho esfuerzo, su creación y mantenimiento pueden resultar tan complejos como la escritura y mantenimiento del propio programa y son, además, propensas a error [10].

En su *roadmap* sobre la dirección que debe tomar la investigación en el área del *testing,* Harrold [11] sugiere la utilización de artefactos de "precódigo", como documentos de diseño, requisitos, especificaciones arquitectónicas, etc., enfoque que resulta muy estimulante para el caso de las líneas de producto.

En este sentido, diferentes autores han propuesto estrategias para obtener casos de prueba a partir de varios tipos de diagramas. Basanieri et al. [12], por ejemplo, utilizan diagramas de casos de uso y de interacción para derivar casos de prueba desde las etapas iniciales del desarrollo de un sistema orientado a objetos: a partir de los escenarios mostrados en los diagramas de interacción construyen secuencias de mensajes que se envían a los objetos que intervienen en el escenario; posteriormente, asignan valores de prueba a los mensajes para construir casos de prueba que podrían traducirse a código para ser ejecutados. Del oráculo no se hace mención en el trabajo, por lo que probablemente haya de ser anotado manualmente para cada caso prueba.

Las máquinas de estados han sido, sin duda, los artefactos que con más éxito se han aplicado para la generación de casos de prueba [13-18]. Si se interpreta de forma parecida a un autómata finito, el lenguaje aceptado por una máquina de estados se corresponde con la secuencia de operaciones permitida por los objetos de la clase cuyo comportamiento está representando, lo que puede incluir las condiciones de guarda. Los estados, por otra parte, si pueden expresarse en función de los atributos de la clase, constituyen por sí mismos un medio muy adecuado para definir restricciones sobre la vida de la instancia, resultando muy útiles para definir los oráculos. El clásico ejemplo de una cuenta corriente que inicialmente está en el estado *Cero,* la transición con la operación *ingresar(x:double),* siendo *x>0*, que conduce al objeto al estado *Positivo,* puede traducirse a una postcondición sobre la operación:

```
context Cuenta :: ingresar(x:double)
```

```
pre : x>0
post: saldo@pre=0 implies saldo>0
```

Para máquinas de estados, además, se han definido criterios de cobertura para garantizar que los casos de prueba que se generan la ejecutan "completamente" [16] (transiciones, predicados, pares de transiciones, secuencia completa).

La investigación en pruebas en líneas de producto se apoya también en artefactos de precódigo y en técnicas de derivación de casos de prueba, con enfoques parecidos a los comentados arriba:

- Nebut et al. [19] obtienen casos de prueba para cada uno de los diferentes productos de una línea a partir de los requisitos funcionales, descritos mediante diagramas de secuencia de alto nivel. Con esto obtienen una serie de "activos reutilizables", a los que llaman "patrones de prueba de comportamiento" (*behavioural test patterns,* behTP), que se utilizan para generar casos de prueba específicos para cada producto. Estos diagramas de secuencia son genérico respecto al nombre y tipo de las instancias, y respecto a los nombres y tipos de los parámetros en las llamadas a los métodos. Cada producto de la línea debe encontrarse descrito en UML mediante: (1) un diagrama de clases; (2) máquinas de estados para las clases principales; (3) diagrama de objetos o de despliegue que represente el estado inicial del sistema. Cada *behTP* representa un conjunto de escenarios genéricos que muestra una vista de alto nivel de algunos escenarios que el sistema puede ejecutar.
- Bertolino et al. [20] derivan casos de prueba a partir de casos de uso adaptados para líneas de producto (PLUC: *Product Line Use Cases*), en los cuales se anotan las variaciones con etiquetas. Los PLUC son descritos con más detalle en la sección 3.1 de este trabajo.
- Kang et al. [21] derivan casos de prueba a partir de los diagramas de secuencia de la línea. Para esto, definen una extensión de los diagramas de secuencia que permite representar variabilidad en los escenarios de los casos de uso. Además, la línea debe disponer de un modelo de características (*Feature Model*), casos de uso con escenarios y componentes. A partir de una arquitectura de prueba, los diagramas de secuencia son la base para la derivación formal del escenario de prueba.
- Reuys et al. [22] obtienen los casos de prueba a partir de los casos de uso y los diagramas de actividad. Utilizan los diagramas de actividad como modelos de prueba de los que se derivan los escenarios de casos de prueba. Los escenarios de casos de prueba son especificados como diagramas de secuencia, describiendo las acciones de quien prueba y las respuestas del sistema, sin especificar los datos de prueba concretos. Luego, los escenarios del caso de prueba se refinan con los datos de entrada, resultados esperados, otra información que pueda hacer falta y scripts de prueba pertinentes para ese escenario. Se observa que el oráculo queda, una vez más, para una fase manual.
- Olimpiew el al.[23] obtienen modelos de prueba a partir de casos de uso. Los modelos de prueba personalizables se crean durante el desarrollo de la línea de producto de software en tres etapas. Primero, crean diagramas de actividad de los casos de uso; luego, tablas de decisión a partir de los diagramas de actividad; por último, plantillas de prueba a partir de las tablas de decisión y de los caminos desde los diagramas de actividad.

En nuestra propuesta, que se describe en las dos siguientes secciones, utilizamos la especificación genérica de la línea de producto en forma de diagramas de secuencia para generar casos de prueba.

3. Diseño de la línea

Siguiendo algunas las ideas de los autores mencionados en la sección anterior, el diseño de la línea de producto se realiza utilizando: casos de uso, para representar y describir las funcionalidades; diagramas de secuencia, para representar los escenarios de cada caso de uso; y las descripciones de los estados por los que pasan las instancias (que pueden proceder de una máquina de estados, si bien ésta no resulta imprescindible), para obtener el oráculo de los casos de prueba. A continuación se describen estos tres pasos con más detalle, incluyendo la forma en que se trata la variabilidad.

3.1. Diseño de casos de uso

El diseño de casos de uso comienza con el dibujo de su diagrama, lo que incluye los propios casos de uso, las relaciones entre ellos, los actores, etc. De cada caso de uso deben identificarse las clases de análisis que

intervienen en su ejecución (de interface, controladores y entidades). En el contexto de las líneas de producto es especialmente importante mantener la trazabilidad entre los diferentes artefactos que se van construyendo, puesto que describen, con mayor o menor nivel de abstracción, diferentes vistas de la línea y de cada producto. Respecto de los casos de uso, es conveniente que éstos sean tan aproximados a la realidad final de la línea y de los productos como sea posible: es decir, mejor si las clases que se identifican son realmente clases de diseño en lugar de clases de análisis.

Cada caso de uso se describe mediante una plantilla textual, en la que los flujos de eventos describen detalladamente los escenarios de ejecución. Estos flujos de eventos son la base para la construcción posterior de los diagramas de secuencia, por lo que los eventos se deben redactar, normalmente, en términos de *sujeto-verbo-predicado*, en donde el sujeto y el predicado se corresponden con actores o con instancias de las clases identificadas, y en donde el verbo se corresponde con el mensaje que más tarde se representará en el diagrama de secuencia. En algunas situaciones (parte de la funcionalidad se delega a otro caso de uso, p. ej.), es posible que no se indique el objeto receptor.

3.1.1. Variabilidad en los casos de uso.

Si bien los casos de uso no se utilizan directamente para generar casos de prueba, el trabajo en líneas de producto impone la gestión conjunta y rigurosa de la trazabilidad y de la variabilidad. En este sentido, existen diferentes propuestas para tratar la variabilidad durante el desarrollo, como las de Gomma [24] o Bertolino et al. [20], que aplican una serie de extensiones a los modelos UML.

Bertolino et al. [25] describen los requisitos funcionales de una línea de productos software mediante PLUC (*Product Line Use Cases*), que contienen la información tradicional más la variabilidad que soportará la funcionalidad en cuestión, según las siguientes etiquetas:

- Alternativas: definen diferentes alternativas de ejecución dependiendo del producto.
- Opcionales: definen ejecuciones que son opcionales dependiendo del producto
- Paramétricas: definen diferentes ejecuciones en función de los valores de otras etiquetas.

Para este trabajo se han redefinido las etiquetas *Alternativas* y *Opcionales* y se ha creado la etiqueta *Alcance*, que indica los productos de la línea a los que afecta la funcionalidad incluida en el caso de uso.

De este modo, a la plantilla que contiene la descripción textual de los casos de uso se le añaden dos nuevas secciones:

- *Alcance*, que permite saber qué productos van a incluir el caso de uso.
- *Variaciones*, que define los puntos de variación. Los flujos de eventos pueden anotarse con etiquetas que estarán definidas en el nuevo campo de Variaciones.

La principal diferencia que marca la adaptación realizada con las etiquetas definidas por Bertolino es que éstas van a estar siempre parametrizadas por la etiqueta que define el alcance del caso de uso: de esta manera, todos los puntos de variación están definidos en función de los productos que soportan el caso de uso, con lo que son fácilmente transportables a otros tipos de diagramas, simplemente transportando las etiquetas.

3.2. Diagramas de secuencia

En general, se construye un diagrama de secuencia por cada flujo de eventos identificado en la plantilla de descripción del caso de uso. Como se ha dicho, cada objeto corresponderá al sujeto o al predicado de un evento en el flujo de eventos, mientras que el mensaje entre los dos objetos procede del verbo que enlazaba a ambos en la descripción del evento.

3.2.1 Variabilidad en los diagramas de secuencia

A través de los PLUC se obtienen diagramas de secuencia, que pueden incorporar variabilidad en función de la presencia o ausencia de etiquetas de puntos de variación. En el caso de los diagramas de secuencia, puede haber variabilidad en los mensajes o en los objetos. Respecto de los mensajes, se definen tres tipos de variabilidad:

- *Opcionales*: son mensajes que podrán no aparecer en todos los productos.
- *Obligatorios*: son mensajes que aparecen en todos los productos, pero cuya implementación variará dependiendo del producto.
- *Fijos*: son mensajes que se comportan exactamente igual en todos productos de la línea.

Los mensajes variables se anotan con el estereotipo adecuado y, junto a éste, se incluye la etiqueta que hace variable a ese mensaje.

Respecto de los objetos, su variabilidad depende de que envíe o reciba mensajes variables en éste o en otro diagrama de secuencia. De esta manera, cuando un objeto contenga mensajes variables, éste se

estereotipará como variable y se añadirán, junto al estereotipo, las diferentes etiquetas que están anotadas en los mensajes.

3.3. Descripción de los estados

Los diagramas de secuencia serán el principal artefacto utilizado para derivar los escenarios de prueba que, en algún momento, han de traducirse a casos de prueba concretos en algún formato tipo X-Unit (JUnit, p.ej.). Un elemento esencial de los casos de prueba es el oráculo, que sirve al fin y al cabo para determinar si el sistema supera o no la prueba. Con objeto de lograr el máximo grado de automatización, deben describirse los estados de las instancias de las clases que: (1) intervienen en los diagramas de secuencia; y (2) son relevantes para determinar el éxito o el fracaso de la ejecución del caso de prueba.

Cada estado debe poder ser descrito en términos de los valores de los atributos de la clase cuyo comportamiento se está representando. Los estados completarán la descripción de los escenarios representados mediante diagramas de secuencia: cuando llega un mensaje a una instancia, se anota el estado en que se espera que quede la instancia en ese escenario. Así, el diagrama de secuencia dispone de toda la información necesaria para obtener los escenarios de prueba y los oráculos.

Se puede considerar la necesidad de describir el comportamiento de las instancias con máquinas de estados pero esto complica el diseño de la línea, pues habría que mantener la correspondencia entre mensajes (del diagrama de secuencia) y transiciones (de la máquina de estados). Se ha comprobado que es suficiente con mantener la descripción de los estados y los diagramas de secuencia convenientemente anotados.

3.3.1 Variabilidad en las descripciones de los estados

Del mismo modo que en los casos anteriores, las descripciones de los estados han de anotarse con las etiquetas que marcan sus puntos de variación. En este caso, además, es posible que haya que combinar puntos de variación definidos en diferentes casos de uso, ya que ahora se representan todos los estados de una clase, independientemente de los escenarios en que esté involucrada. Puesto que los estados estarán definidos en función de los atributos de la clase a la que representa, la variabilidad se encuentra en ellos. Así, se definen dos tipos de variabilidad:

- *Fijos*: son atributos que compondrán el estado en todos los productos.
- *Opcionales*: son atributos que van componer el estado solo en algunos productos.

A la descripción de los estados se añaden las etiquetas de los puntos de variación que hacen variables a los atributos.

3.4. Una línea de productos para juegos de mesa

Supongamos que se desea construir una línea de productos para juegos de mesa (ajedrez, trivial, damas, parchís…) por ordenador. Se dispondrá de un servidor del juego al que se conectan una serie de clientes, cada uno de los cuales es operado por una persona, que toma las decisiones del juego que le parezcan oportunas.

En esta línea coexisten, por tanto, dos sistemas que pueden describirse, resumidamente, como en la Figura 1: el lado izquierdo muestra la vista general de la aplicación cliente, que se comunica con dos actores (el jugador y el servidor); el derecho representa el sistema servidor, que se comunica únicamente con el sistema cliente.

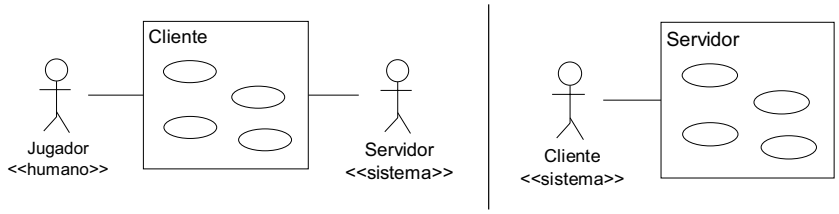

Figura 1. Vista general de la aplicación cliente/servidor

Uno de los casos de uso del servidor es *Mover pieza*, que se ejecuta cuando el cliente envía el movimiento de una pieza sobre el tablero. Cuando esto ocurre, y como se muestra en el lado izquierdo de la Figura 2, el servidor debe realizar otras operaciones, como comprobar la legalidad del movimiento, comer quizás otra ficha, pasar tal vez el turno y actualizar el estado de los clientes que juegan esa partida. Por mantener el

ejemplo simplificado, asumimos que son suficientes las dos clases mostradas en el lado derecho de la figura (*Controlador* y *Partida)* para gestionar este caso de uso.

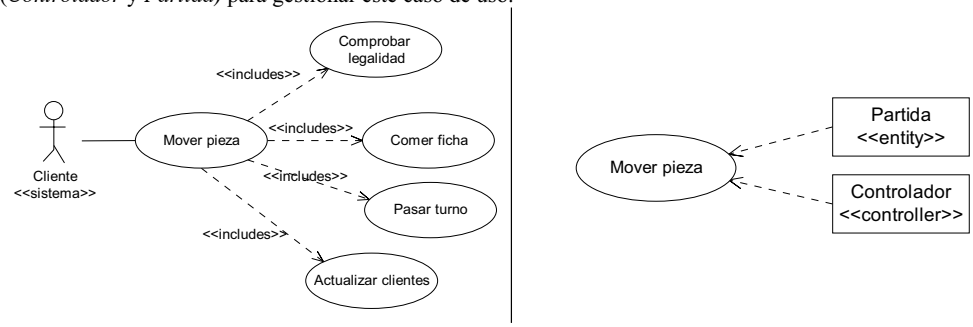

Figura 2. El caso de uso *Mover pieza*, perteneciente al *Servidor*

CASO DE USO	Mover pieza
OBJETIVO	Mover una pieza de una posición a otra.
ALCANCE	Cualquier juego [MP0]
PRECONDICIONES	Que sea el turno del cliente que realiza el movimiento
CONDICIÓN FINAL DE ÉXITO	Una pieza queda movida
CONDICIÓN FINAL DE FRACASO	No hay movimiento
ACTORES	Cliente
DISPARADOR	El cliente ejecuta un movimiento

FLUJO NORMAL DE EVENTOS

1. El **cliente** le dice al **controlador** que realice el movimiento.
2. El **controlador** le pasa el movimiento a la **partida**
3. {[MP1] La **partida** comprueba la legalidad del movimiento. *Ref*. *cruzada* Comprobar legalidad movimiento}
4. {[MP2] El **controlador** le dice a la **partida** que coma ficha (si procede). *Ref*. *cruzada* Comer ficha}
5. El **controlador** notifica la jugada los **clientes contrincantes**. *Ref*. *cruzada* Actualizar clientes.
6. {[MP3] El **controlador** le pregunta a la **partida** si hay que pasar el turno}
7. {[MP2] El **controlador** pasa el turno. *Ref*. *cruzada* Pasar Turno}

VARIABILIDAD

MP0: [1 de n]. Alcance
  0 - Ajedrez, 1 – Damas, 2 – Parchís, 3 – Trivial

MP1:[1 de n]. Alternativa
  Si MP0=0
    Comprobar que el movimiento es correcto en función de la pieza movida
  Si MP0=1
    Comprobar que el movimiento es correcto en diagonal
  Si MP0= 3 || MP0=2
    Comprobar que la posición final es coherente con el resultado de los dados obtenido antes de mover

MP2: [0..1 de 1]. Opcional
  Cuando MP0= 0 || MP0=1 || MP0=2

MP3: [0..1 de 1]. Opcional
  Cuando MP0= 2

Figura 3. Descripción textual del caso de uso *Mover pieza*

La Figura 3 muestra la descripción textual del caso de uso, en la que se describe únicamente el flujo normal de eventos. Obsérvese que, respecto de otras plantillas habituales, se han añadido dos secciones:

Alcance, en la que se indican los productos a los que es aplicable el caso de uso (un supuesto caso de uso *Tirar dados* no sería aplicable, por ejemplo, al juego del ajedrez); y *Variabilidad*, que describe los puntos de variación del caso de uso, según se indicó en la Sección 3. En el ejemplo de la figura se muestra un ejemplo de cada tipo de etiqueta definida: con la etiqueta *MP0* indicamos el alcance del caso de uso; con *MP1*, una alternativa (se ejecutará siempre, pero de forma diferente según el producto); con *MP2* y *MP3* se representan dos etiquetas opcionales (pueden ejecutarse o no, según el producto). Nótese que a estas etiquetas se les hace referencia en la descripción del flujo normal de eventos.

Una vez descrito el caso de uso en la plantilla, se construyen los diagramas de secuencia procedentes de los flujos de eventos que se hayan indicado. La Figura 4 muestra el diagrama correspondiente al flujo normal de eventos del caso de uso *Mover pieza* (descrito en la Figura 3). Obsérvense los estereotipos y etiquetas de los mensajes y objetos variables, en las que se hace referencia a los productos a los que son aplicables.

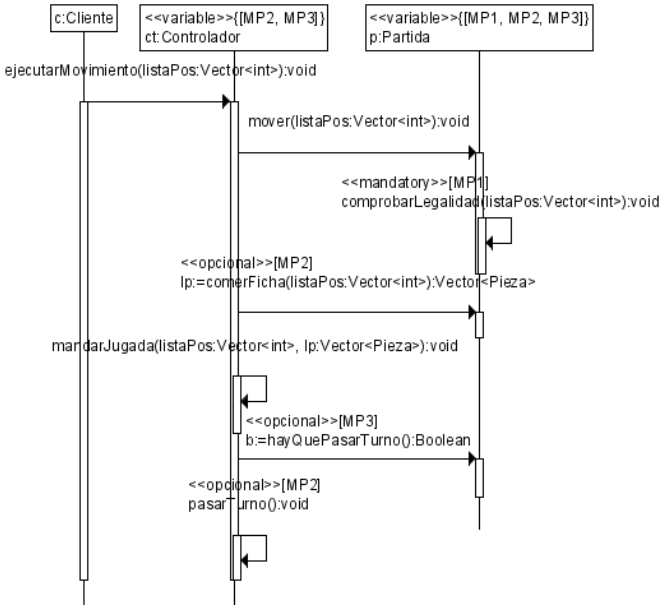

Figura 4. Diagrama de secuencia con variabilidad

Después de construir los diagramas de secuencia se describen los estados de las diferentes clases, usando las notaciones de variabilidad expuestas anteriormente (sección 3.3.1). Como se ha comentado en la sección 3.3, los estados deben poderse describir en función de los valores de los campos de la clase, ya que contribuyen a determinar el estado en el que debe quedar la instancia después de ejecutar la operación. Por ello, la Tabla 1, que contiene la descripción de los estados de *Partida*, es un complemento esencial para la posterior generación de oráculos y de casos de prueba, y debe apoyarse en la estructura de clases.

Tabla 1. Descripción de los estados de la de la clase Partida.

Creada	this.tablero == null this.clientes.size() == 0 this.nJugadores == 0 <<opcional>>{CP1}
Iniciando	this.tablero != null this.clientes.size() < nJugadores this.jConTurno == null
Jugando	this.clientes.size() == nJugadores this.jConTurno != null this.jugada == null this.piezaAMover ==null this.posicionesAComer == -1<<opcional>>{MP2} this.fichasComidas == -1<<opcional>>{MP2}
Comiendo	this.posicionesAComer > 0 <<opcional>>{MP2} this.fichasComidas < this.posicionesAComer <<opcional>>{MP2}
anotandoDados	this.seisesSeguidos != -1 <<opcional>>{TD0} this.puntuacionDados !=0 <<opcional>>{TD0}

Por último hay que relacionar los diagramas de secuencia construidos con los estados definidos. La Figura 5 muestra el diagrama de secuencia de la Figura 4, pero relacionado con los estados que establecen los diferentes mensajes. Obsérvese que estos estados están anotados antes de cada mensaje entre comillas (*mandarJugada* y *preparadoParaMover* son estados del controlador, por lo que no aparecen en la Tabla 1).

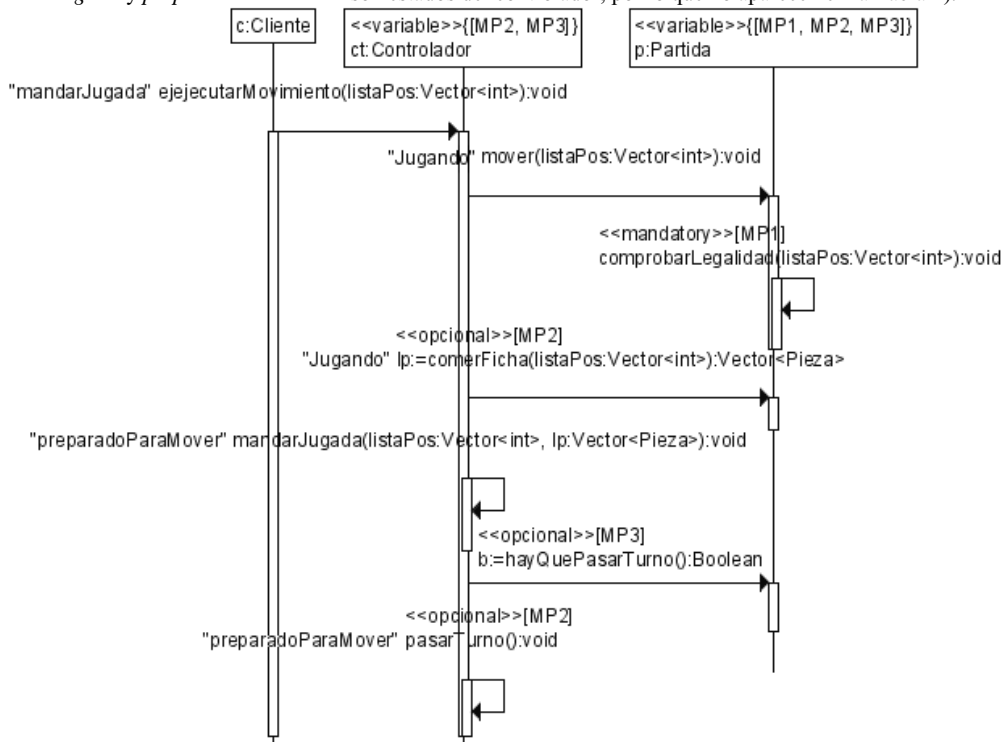

Figura 5. Diagrama de secuencia con variabilidad que contiene la información de los estados finales

4 Obtención de los casos de prueba

Cuando la línea se encuentra diseñada conforme a la sección anterior, se dispone de la información necesaria para generar los casos de prueba. Para ello, se generan en primer lugar los escenarios de prueba a partir de los diagramas de secuencia y, a continuación, se obtienen los oráculos a partir de los estados anotados en el diagrama de secuencia.

4.1 Generación de los escenarios

El primer paso es obtener los escenarios de prueba. Siguiendo las ideas de Basanieri [12] y Bertolino [6], los escenarios de prueba de una clase se obtienen a partir de los diagramas de secuencia en los que interviene esa clase.

El escenario de pruebas está compuesto por los métodos correspondientes a los mensajes que llegan a la instancia en el diagrama de secuencia: se excluyen, por tanto, los mensajes lanzados por la propia instancia.

A partir del ejemplo de la Figura 4, se obtiene un escenario de prueba para la clase *Partida,* en el cual se incluyen las operaciones *mover* y *comerFicha,* que son las dos operaciones que llegan a la instancia de *Partida* desde otros objetos (se excluye el mensaje *hayQuePasarTurno,* pues no modifica el estado de la instancia). La Tabla 2 muestra las operaciones que intervienen en el escenario de prueba generado para la clase Partida. Cada mensaje se anota con la variabilidad y el estado esperado.

Tabla 2. Mensajes involucrados en el escenario de prueba

Mensaje	Variabilidad	Estado
mover		Jugando
comerFicha	MP2	Jugando

4.2 Generación de los oráculos

Para generar los oráculos se utilizan las condiciones de los estados a los que hacen referencia los mensajes de los escenarios. Para cada mensaje del escenario de prueba, se obtienen de la Tabla 1 las condiciones del estado que le corresponde. En este ejemplo, ya que hay que ejecutar dos métodos, habrá que generar dos oráculos.

Las condiciones asociadas a los estados se indican a continuación. Obsérvese que, puesto que en el escenario de prueba se ejecuta el método "*comerFicha*", pero no se come ninguna ficha, el estado final de dicho método será *Jugando* (Figura 5): por tanto, el oráculo para ambos métodos estará compuesto por las siguientes condiciones:

- this.clientes.size() == nJugadores
- this.jConTurno != null
- this.jugada == null
- this.piezaAMover ==nul
- this.posicionesAComer == -1 <<opcional>>{MP2}
- this.fichasComidas == -1 <<opcional>>{MP2}

Puesto que estas condiciones están expresadas de forma ejecutable, simplemente habrá que añadirlas a la condición del oráculo mediante conjunciones.

En la Tabla 3 se muestran las condiciones que deben comprobar los oráculos que, posteriormente, serán combinados con los escenarios de prueba para generar los casos.

Tabla 3. Condiciones para los oráculos de los dos mensajes del escenario de prueba

Método	Condición
mover	this.clientes.size() == nJugadores this.jConTurno != null this.jugada == null this.piezaAMover ==null this.posicionesAComer == -1<<opcional>>{MP2} this.fichasComidas == -1<<opcional>>{MP2}
comerFicha	this.clientes.size() == nJugadores this.jConTurno != null this.jugada == null this.piezaAMover ==nul this.posicionesAComer == -1<<opcional>>{MP2} this.fichasComidas == -1<<opcional>>{MP2}

4.3 Obtención de los casos de prueba

Una vez se dispone de los escenarios de prueba y de los oráculos, se está en condiciones de obtener casos de prueba para la línea. Además del código ejecutable, cada caso contendrá la variabilidad que le afecta, de modo que posteriormente puedan obtenerse casos de prueba específicos para cada producto.

Un caso de prueba estará compuesto por una declaración de valores de prueba (algunos son comunes para todos los productos de la línea; mientras que otros, particulares, se crearán mediante métodos abstractos que serán implementados en especializaciones), la construcción de la instancia de la clase bajo prueba y la ejecución de los métodos que comprende el escenario, junto con el oráculo generado.

Cada operación del escenario se traduce a una llama a un método, anotándose su variabilidad mediante un comentario dentro del código. A continuación, se añade el oráculo. Los oráculos también tendrán anotada su variabilidad mediante un comentario antes de la instrucción.

Por último se realiza una conversión del texto de los oráculos para que hagan referencia al objeto que se está probando (es decir, los *this* se sustituyen por el identificador del objeto que se ha creado en el caso de

prueba). En la Figura 6 se muestra el caso de prueba que se ha obtenido a partir de los mensajes de la Tabla 2 y los oráculos de la Tabla 3. Como se aprecia, la líneas de comentario identifican los puntos de variación que afectan a las diferentes instrucciones.

```
public void test1(){

    Vector<int> arg1 = crearArg1();

    Partida o = new Partida();

    o.mover(arg1);
    //{VP} MP2
    assertTrue(o.clientes.size() == nJugadores && o.jConTurno != null &&
               o.jugada == null && o.piezaAMover ==nul &&
               o.posicionesAComer == -1 && o.fichasComidas == -1);

    //{VP} MP2
    res=o.comerFicha(arg1);
    //{VP} MP2
    assertTrue(o.clientes.size() == nJugadores && o.jConTurno != null &&
               o.jugada == null && o.piezaAMover ==nul &&
               o.posicionesAComer == -1 && o.fichasComidas == -1);

}
```

Figura 6. Caso de prueba genérico para la línea de juegos de mesa

4.4 Obtención de casos de prueba para los productos

Cuando los casos de prueba para la línea están construidos, se generan los casos de prueba específicos para cada producto, gracias a que se ha ido arrastrando y anotando la variabilidad de cada método.

Los puntos de variación asociados a cada llamada a un método pueden ser *Opcionales* (el caso de prueba varía dependiendo del producto, pudiendo estar la llamada presente o no) y *Alternativos* (el caso de prueba no varía). Además, los oráculos generados también pueden contener variabilidad, por lo que habrá que revisarlos junto con la Tabla 3.

La Figura 7 muestra un caso de prueba específico generado para un producto concreto (el *Trivial*) y otro para el *Ajedrez*. Puesto que el punto de variación indicado en la Figura 6 (denotado con *{VP} MP2*) no afecta al primero, la llamada a *comerFicha* desparece, junto con su oráculo y el oráculo generado para el método mover se ve modificado, ya que contienen condiciones que no han de estar presentes en este producto (ver Tabla 3). En el segundo caso, sí se incluye la llamada a *comerFicha* y los oráculos mantienen todas sus condiciones, ya que se ve afectado por el punto de variación. Como se ve, se mantiene el método y los oráculos afectados por el punto de variación, pero desaparece su notación de variabilidad.

```
public void test1(){
  Vector<int> arg1 = crearArg1();

  Partida o = new Partida();

  o.mover(arg1);
  assertTrue(
    o.clientes.size() == nJugadores &&
    o.jConTurno != null &&
    o.jugada == null &&
    o.piezaAMover ==nul &&);
}
```

```
public void test1(){

  Vector<int> arg1 = crearArg1();

  Partida o = new Partida();

  o.mover(arg1);
  assertTrue(
    o.clientes.size() == nJugadores &&
    o.jConTurno != null &&
    o.jugada == null && o.piezaAMover ==nul &&
    o.posicionesAComer == -1 &&
    o.fichasComidas == -1);

  res=o.comerFicha(arg1);
  assertTrue(o.clientes.size() == nJugadores &&
    o.jConTurno != null &&
    o.jugada == null && o.piezaAMover ==nul &&
    o.posicionesAComer == -1 &&
    o.fichasComidas == -1);

}
```

Figura 7. Un caso de prueba para el *Trivial* (izquierda) y otro para el *Ajedrez*

5. Conclusiones, trabajos futuros y lecciones aprendidas

En este artículo se ha presentado un enfoque para generar casos de prueba en líneas de producto. Una contribución significativa respecto de otros trabajos es el grado de automatización que se consigue en cuanto a la generación del oráculo, que es una línea de investigación que permanece abierta en el área del *testing*.

El enfoque requiere, como ya plantearon algunos investigadores, la completa descripción de la línea con artefactos de precódigo (especialmente diagramas de secuencia y máquinas de estados), que deben desarrollarse conjuntamente. En este contexto, se aprecia, más que en desarrollo de software tradicional, la necesidad de especificar con calidad y formalidad el sistema. Debido a este requisito, los costes del análisis y diseño del sistema resultan necesariamente más altos, pero previsiblemente han de compensar en las etapas posteriores de desarrollo (las especificaciones son susceptibles de ser procesadas mediante herramientas MDA), mantenimiento y pruebas (como se ha explicado en este trabajo). Queda pendiente la realización de estudios empíricos para constatar esta hipótesis económica, quizás mediante Ingeniería del Software Basada en Valor [26]. Por otro lado, se está desarrollando una herramienta para integrar el diseño de líneas de producto y la generación de casos de prueba, que utiliza las funcionalidades incluidas en *testooj,* una herramienta de generación, ejecución y optimización de casos de prueba [27]

6. Referencias

1. Clements P and Northrop L (2002). *Software Product Lines: Practices and Patterns*. Boston, USA: Addison-Wesley.
2. McGregor J, Northrop L, Jarrad S and Pohl K (2002). *Initiating Software Product Lines*. IEEE Software, **19**(4), p. 24-27.
3. Díaz Ó and Trujillo. S (2008). *Líneas de producto software*. En *Fábricas de Software: Experiencias, tecnologías y organización*, Piattini and Garzás (eds.). Ra-Ma: Madrid, Spain.
4. Krueger C (2006). *The emerging practice of software product line development*. Military Embedded Systems, (2nd semester), p. 34-36.
5. Thiel S and Hein A (2002). *Modeling and Using Product Line Variability in Automotive Systems*. IEEE Software, **19**(4), p. 66-72.
6. Bertolino A (2007). *Software Testing Research: Achievements, Challenges, Dreams*. Internation Conference on Software Engineering: IEEE Computer Society.
7. Baresi L and Young M (2001). Test oracles. Technical Report CIS-TR01 -02, Dept. of Computer and Information Science, Univ. of Oregon.
8. Tse T and Xu Z (1996). *Test Case Generation for Class-Level Object-Oriented Testing*. 9th International Software Quality Week. San Francisco, CA.
9. Doong RK and Frankl PG (1994). *The ASTOOT approach to testing object-oriented programs*. ACM Transactions on Software Engineering and Methodology, **3**(2), p. 101-130.
10. Hoffman D, Strooper P and Wilkin S (2005). *Tool support for executable documentation of Java class hierarchies*. Software Testing, Verification and Reliability, **15**(4), p. 235-256.
11. Harrold MJ (2000). *Testing: a Roadmap*. International Conference on Software Engineering. Limerick, Ireland: ACM.
12. Basanieri F, Bertolino A and Marchetti E (2002). *The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects*. 5th International Conference on The Unified Modeling Language: Springer-Verlag. LNCS.
13. Kirani S and Tsai WT (1994). *Method sequence specification and verification of classes*. Journal of Object-Oriented Programming, **7**(6), p. 28-38.
14. Grieskamp W, Gurevich Y, Schulte W and Veanes M (2001). *Testing with abstract state machines*. Formal Methods and Tools for Computer Science. Canary Islands, Spain.
15. Ball T, Hoffman D, Ruskey F, Webber R and White L (2000). *State generation and automated class testing*. Software Testing, Verification and Reliability, **10**, p. 149-170.
16. Offutt AJ, Liu S, Abdurazik A and Amman P (2003). *Generating test data from state-based specifications*. Software Testing, Verification and Reliability, (13), p. 25-53.
17. Hong HS, Lee I and Sokolsky O (2001). *Automatic test generation from statecharts using model checking*. Formal Approaches to Testing of Software (FATES'01). Aalborg, Denmark: BRICS: Basic Research in Computer Science.

18. Burton S, Clark J and McDermid J (2001). *Automatic generation of tests from statechart specifications*. Formal Approaches to Testing of Software (FATES'01). Aalborg, Denmark: BRICS: Basic Research in Computer Science.
19. Nebut C, Pickin S, Le Traon Y and Jezequel J (2003). *Automated requirements-based generation of test cases for product families*. Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on, p. 263-266.
20. Bertolino A, Gnesi S and di Pisa A (2004). *PLUTO: A Test Methodology for Product Families*. Software Product-family Engineering: 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003: Revised Papers.
21. Kang S, Lee J, Kim M and Lee W (2007). *Towards a Formal Framework for Product Line Test Development*. Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on, p. 921-926.
22. Reuys A, Kamsties E, Pohl K and Reis S (2005). *Model-based System Testing of Software Product Families*. Pastor, O.; Falcao e Cunha, J.(Eds.): Advanced Information Systems Engineering, CAiSE, p. 519–534.
23. Olimpiew E and Gomaa H (2006). *Customizable Requirements-based Test Models for Software Product Lines*. International Workshop on Software Product Line Testing.
24. Gomaa H (2005). *Designing Software Product Lines with UML.*
25. Bertolino A, Fantechi, A., Gnesi, S., Lami, G. *Product Line Use Case*.
26. Cabrero D, Garzás J and Piattini M (2007). *Maintenance cost of a software design: A value-based approach*. ICEIS: International Conference on Enterprise Information Systems.
27. Polo M, Piattini M and Tendero S (2007). *Integrating techniques and tools for testing automation*. Software Testing, Verification and Reliability, **17**(1), p. 3-39.


Universidad
de Oviedo

400
cuarto
centenario

PCTI
ASTURIAS

Ayuntamiento de Gijón

GOBIERNO
DE ESPAÑA
MINISTERIO
DE CIENCIA
E INNOVACIÓN

Gobierno del
Principado de Asturias

TIC
CLUSTER ASTURIAS

CSC

INTERSYSTEMS

cajAstur

DEPARTAMENTO DE INFORMÁTICA
UNIVERSIDAD DE OVIEDO

Sistedes
Sociedad de Ingeniería del Software y
Tecnologías de Desarrollo de Software

gijon.info