CibSE2010

XIII Congreso Iberoamericano en "Software Engineering"

El evento de referencia
en el ámbito iberoamericano
de la Ingeniería del Software

Organizan

UNIVERSIDAD DEL AZUAY

UNIVERSIDAD DE CUENCA

UNIVERSIDAD DEL PACÍFICO
Escuela de Negocios

Con el auspicio de

CUENCA ¡todo un mundo!

Microsoft

ISBN: 978-9978-325-10-0

http://www.uazuay.edu.ec/cibse/

CibSE2010 XIII Congreso Iberoamericano en "Software Engineering"

CibSE2010

XIII Congreso Iberoamericano
en "Software Engineering"

Universidad del Azuay

12 - 16 de abril

Cuenca - Ecuador

CIbSE 2010

Actas

XIII Conferencia Iberoamericana en "Software Engineering"

Editores

Xavier Franch, Itana Maria de Souza Gimenes, Juan Pablo Carvallo.

Cuenca, Ecuador

12 a 16 de abril de 2010

Ficha Técnica
Actas de la XIII Conferencia Iberoamericana en "Software Engineering"
Editores: Xavier Franch, Itana Maria de Souza Gimenes, Juan Pablo Carvallo
Abril 2010 – Cuenca, Ecuador

Editorial: Universidad del Azuay
ISBN: 978-9978-325-10-0

PRESIDENCIA DEL COMITÉ ORGANIZADOR

Juan Pablo Carvallo
Universidad del Azuay / Universidad del Pacifico, Cuenca, Ecuador

PRESIDENCIA DEL COMITÉ DE PROGRAMA

Xavier Franch
Universitat Politècnica de Catalunya, España

Itana Maria de Souza Gimenes
Universidade Estadual de Maringá, Brasil

PRESIDENCIA DEL SIMPOSIO DOCTORAL

Claudia P. Ayala
Universitat Politècnica de Catalunya, España

PRESIDENCIA DE PUBLICIDAD

Carlos Cares
Universidad de la Frontera, Temuco, Chile

COMITÉ LOCAL

Oswaldo Merchan Universidad de Azuay	Marlene Albarracín Universidad de Azuay	Nathalia Rincón Universidad de Azuay
Juan Carlos Lazo Universidad de Azuay	Jorge Espinoza Universidad de Azuay	Lucía Mendez Univ. de Cuenca
Katherine Ortiz Universidad de Azuay	Manuel Guamán Universidad de Azuay	Javier Valdiviezo Univ. del Pacífico

MIEMBROS DEL COMITÉ DE PROGRAMA

Alejandra Cechich, *Universidad Nacional de Comahue,* Argentina
Alessandro García, *Lancaster University,* UK
Alvaro Arenas, *CCLRC Rutherford Appleton Lab.,* UK
Amador Durán, *Universidad de Sevilla,* España
Antonio Brogi, *Università di Pisa,* Italia
Antonio Vallecillo, *Universidad de Málaga,* España
Carla Silva, *Universidade Federal de Pernambuco,* Brasil
Carme Quer, *Universitat Politècnica de Catalunya,*
Claudia Ayala, *Norwegian University of Science and Technology,* Norway
Claudia Pons, *Universidad Nacional de La Plata,* Argentina
Coral Calero, *Universidad de Castilla-La Mancha,* España
Cristina Gómez, *Universitat Politècnica de Catalunya,* España
Dan Hirsch, *Intel,* Argentina
Daniel Riesco, *Universidad de San Luis,* Argentina
Daniela Godoy, *Universidad Nacional del Centro,* Argentina
Elena Navarro, *Universidad de Castilla-La Mancha,* España
Ernest Teniente, *Universitat Politècnica de Catalunya,* España
Ernesto Pimentel, *Universidad de Málaga,* España
Falcão e Cunha João, *Universidade do Porto,* Portugal
Fernanda Alencar, *Universidade Federal de Pernambuco,* Brasil
Francisco Pinheiro, *Universidade de Brasilia,* Brasil
Francisco Ruíz, *Universidad de Castilla-La Mancha,* España
Gaston Mousques, *Universidad ORT,* Uruguay
Guilherme Travassos, *Universidade Federal Rio de Janeiro,* Brasil
Gustavo Rossi, *Universidad Nacional de La Plata,* Argentina
Hernan Melgratti, *Universidad de Buenos Aires,* Argentina
Isabel Brito, *Instituto Politécnico de Beja,* Portugal
Isabel Díaz, *Universidad Central de Venezuela,* Venezuela
Jaelson Castro, *Universidade Federal de Pernambuco,* Brasil
Jesús García-Molina, *Universidad de Murcia,* España
João Araújo, *Universidade Nova de Lisboa,* Portugal
Jonás Montilva, *Universidad de Los Andes,* Venezuela
José Maldonado, *Universidade de São Paulo,* Brasil
José Pow-Sang, *Pontificia Universidad Católica del Perú,* Perú
Juan Carlos Trujillo, *Universidad de Alicante,* España
Juan Hernández, *Universidad de Extremadura,* España
Júlio Leite, *Pontificia Universidad Católica,* Brasil
Luca Cernuzzi, *Universidad Católica Nuestra Señora de Asunción,* Paraguay
Luís Olsina, *Universidad Nacional de La Pampa,* Argentina
Lyrene Silva, *Universidade Estado do Rio Grande do Norte,* Brasil
Marcello Visconti, *Universidad Técnica Federal Santa María,* Chile
Márcio Barros, *Universidade Federal Rio de Janeiro,* Brasil
Maria Cecilia Bastarrica, *Universidad de Chile,* Chile
María Lencastre, *Universidade Federal de Pernambuco,* Brasil
Miguel Katrib, *Universidad de la Habana,* Cuba
Oscar Dieste, *Universidad Politécnica de Madrid,* España
Oscar Pastor, *Universidad Politécnica de Valencia,* España
Rafael Calvo, *University of Sydney,* Australia
Raquel Anaya, *Universidad EAFIT,* Colombia
Regina Braga, *Universidade Federal de Juiz de Fora,* Brasil
Renata Guizzardi, *Universidade Federal do Espírito Santo,* Brasil
Ricardo Falbo, *Universidade Federal do Espírito Santo,* Brasil
Roxana Giandini, *Universidad Nacional de La Plata,* Argentina
Sandra Fabri, *Universidade Federal de São Carlos,* Brasil
Silvia Gordillo, *Universidad Nacional de La Plata,* Argentina
Vicente Pelechano, *Universidad Politécnica de Valencia,* España
Victor Santander, *Universidade Estadual Oeste do Paraná,* Brasil

REVISORES ADICIONALES

Agustina Buccella
Andrea Delgado
Andrés Flores
Cesar Mtz. Spessot
Cláudio Sant'Anna
Dante Carrizo
Elisa Y. Nakagawa
Evellin Cardoso
Fernando Brunetti
Francisco Gutiérrez
Hélio Costa
Hernán Molina
Irene Garrigós
Javier Cámara
Javier Troya
Jorge Villalón
José E. Rivera
José Fco. Aldana
Jose-Norberto Mazón
Manuel F. Bertoa
María de los A. Martín
Mario Beron
Martín Solari
Pablo Garralda
Pablo Michelis
Patricio Maller
Santiago Matalonga
Thaizel Fuentes
Vander Alves
Victoria Torres

PREFACIO

Bienvenidos a la decimotercera edición de la Conferencia Iberoamericana de "Software Engineering" (CIbSE 2010) celebrada en Cuenca, Ecuador, organizada conjuntamente por la Universidad del Azuay, la Universidad del Pacífico y la Universidad de Cuenca, del 12 al 16 de abril del 2010.

Con el objetivo de convertirse en un evento de referencia en el ámbito iberoamericano de la Ingeniería del Software, CIbSE recoge el testigo de la Conferencia de Ingeniería de Requisitos y Ambientes Software (IDEAS) para promover la investigación de calidad en dicho ámbito. CIbSE provee un foro que permite que investigadores, educadores y profesionales en general, presenten y discutan los desarrollos más recientes en ingeniería del software.

La primera edición de IDEAS se celebró en 1998 en Torres, Brasil, como un workshop. Desde entonces, el evento se ha realizado de manera exitosa en diversos países de Latinoamérica: San José-Costa Rica (IDEAS'99), Cancún-México (IDEAS'00), Heredia-Costa Rica (IDEAS'01), La Habana-Cuba (IDEAS'02), Asunción-Paraguay (IDEAS'03), Arequipa-Perú (IDEAS'04), Valparaíso-Chile (IDEAS'05), La Plata-Argentina (IDEAS'06), Isla de Margarita-Venezuela (IDEAS'07), Pernambuco-Brasil (IDEAS'08) y Medellín-Colombia (IDEAS'09), siendo en esta última edición que se aprobó oficialmente el cambio de nombre del evento de Workshop a Conferencia, teniendo en cuenta su evolución en número de trabajos presentados y participantes inscritos.

La agenda académica de CIbSE 2010 cuenta con tres conferencias plenarias, una mesa redonda, cuatro tutoriales, un simposio doctoral y seis sesiones técnicas en las que se presentan los trabajos científicos aceptados por el Comité de Programa. Los tres ponentes convidados son Sebastián Uchitel (Imperial College London, UK; Universidad de Buenos Aires, Argentina), Julio Cesar Sampaio do Prado Leite (Pontifícia Universidade Católica do Rio de Janeiro, Brasil) y Lalo Steinman (Microsoft Latin America, Ecuador).

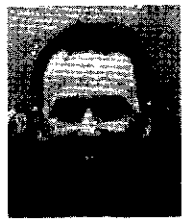
CIbSE 2010 ha recibido un total de 82 artículos provenientes de 16 países diferentes escritos en los tres idiomas oficiales de la conferencia. Los trabajos fueron normalmente revisados por tres miembros del Comité de Programa de la conferencia, formado por investigadores de primer orden de la comunidad. Después de un minucioso proceso de selección, fueron aceptados 16 artículos completos y 8 artículos cortos, resultando pues en un porcentaje de aceptación del 19.5%, lo que muestra el rigor de dicho proceso. El esfuerzo del Comité

de Programa y de los revisores adicionales que colaboraron en el proceso de revisión fue sobresaliente y digno de mención.

Desde este prefacio queremos agradecer profundamente a todos aquellos que han hecho posible la organización de CIbSE 2010. A los autores que enviaron sus trabajos y sus propuestas de tutoriales a la conferencia. A los miembros del Comité de Programa y revisores adicionales que posibilitaron la confección del programa. A la Presidenta del Simposio Doctoral. A los miembros del Comité Local y Presidente de Publicidad que lidiaron con toda la problemática organizativa propia de un evento de la magnitud de CIbSE 2010.

Finalmente, extendemos una cordial bienvenida a conferenciantes, autores, estudiantes y profesionales que asisten a CIbSE 2010. Esperamos que puedan disfrutar del evento y además tengan la oportunidad de disfrutar de la cultura de Cuenca y la amabilidad de su gente.

Cuenca, Ecuador, Abril 2010



Xavier Franch

Itana Maria de Souza Gimenes

Presidentes del Comité de Programa

Juan Pablo Carvallo

Presidente General

ÍNDICE / ÍNDICE / INDEX

Conferencias Invitadas / Palestras Convidadas / Invited Talks

Tutoriales / Tutoriais / Tutorials

Sesiónes Tecnicas / Sessões Técnicas / Technical Sessions

SESIÓN 1: REQUISITOS

SESIÓN 2: ESPECIFICACIÓN Y DISEÑO DE SOFTWARE

SESIÓN 3: INGENIERÍA DEL SOFTWARE EXPERIMENTAL

SESIÓN 4: MDD. METAMODELOS

SESIÓN 5: APLICACIONES WEB E INTERFAZ HOMBRE-MÁQUINA

SESIÓN 6: PROCESOS DE NEGOCIO Y DE SOFTWARE. MEDICIÓN

Artículos Cortos / Artigos Curtos / Short Papers

Simposium Doctoral / Simpósio de Doutorado / Doctoral Symposium

Presentaciones Cortas / Apresentações Curtas / Short Presentations

Generación automática de casos de prueba en Ingeniería del Producto en el contexto de las Líneas de Producto Software

Beatriz Pérez Lamancha[1], Macario Polo Usaola[2], Mario Piattini Velthius[2],
[1] Centro de Ensayos de Software, Universidad de la República,
Montevideo, Uruguay, bperez@fing.edu.uy
[2] Grupo de Investigación ALARCOS, Universidad de Castilla-La Mancha,
Ciudad Real, España, {macario.polo, mario.piattini}@uclm.es

Abstract. In previous works, an automated testing framework was defined. The framework follows Model Driven Engineering principles applied to testing and can be used both in Software Product Line development and in tradicional development projects. The test models are generated from UML design models using the model transformation languge Query / View / Transformation (QVT), an OMG standard for model transformation languages. The test models are consistent with the UML Testing Profile. For Software Product Line development, the variability is traced to test models. This paper describes the way in that test model are derivated from line test models: the variability is resolved by means of a UML profile, and some of the QVT transformations are explained.

Keywords: Testing, Software Product Lines, Model-Driven Engineering, Model-Driven Testing, UML Testing Profile, QVT.

1 Introducción

La Ingeniería Dirigida por Modelos (*Model-Driven Engineering*) considera los modelos como elementos principales para el desarrollo, mantenimiento y evolución del software a través de transformaciones entre modelos [1]. Sumado a la independencia entre modelos, la Arquitectura Dirigida por Modelos (*Model-Driven Architecture, MDA*) [2] separa la complejidad del negocio de los detalles de su implementación, definiendo distintos modelos de software en diferentes niveles de abstracción. Se definen tres puntos de vista del sistema en MDA: (i) Modelo Independiente de la Computación (*Computation Independent Model, CIM*), que describe el contexto y los requisitos del sistema sin considerar su estructura o procesos, (ii) el Modelo Independiente de la Plataforma (*Platform Independent Model, PIM*), que describe las capacidades operacionales del sistema sin tener en cuenta una plataforma específica, y (iii) Modelo Específico de la Plataforma (*Platform Specific Model, PSM*), que incluye detalles del sistema para una plataforma concreta [3]. Una transformación entre modelos "es el proceso de convertir un modelo en otro para el mismo sistema" [2]. Un lenguaje de transformación de modelos es un

lenguaje que toma un modelo como entrada y, de acuerdo a un conjunto de reglas, produce un modelo de salida. Si bien existen muchos lenguajes de transformación, en este trabajo se utiliza el lenguaje *Query/View/Transformation* (QVT) [4], el cual ha sido definido por *Object Management Group (OMG)* como estándar para las transformaciones en el contexto de MDA.

Las Líneas de Producto de Software son propicias para ser desarrolladas siguiendo los principios de la Ingeniería Dirigida por Modelos. Una línea de productos software se define como "un conjunto de sistemas software, que comparten un conjunto común de características (*features*), las cuales satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base (core assets) de una manera preestablecida" [5]. Uno de los aspectos distintivos de las LPS frente al desarrollo tradicional es la importancia de la variabilidad a lo largo de todo el proceso de desarrollo: los productos de la línea comparten un conjunto de características (*commonalities*) y difieren en determinados puntos de variación (*variation points*), que representan la variabilidad entre los productos. Un aspecto central en el desarrollo de LPS es la división de los procesos de ingeniería: la Ingeniería de Dominio, responsable de desarrollar los elementos comunes a todos los integrantes de la línea y su mecanismo de variabilidad, y la Ingeniería del Producto (o Ingeniería de la Aplicación), cuyos cometidos incluyen desarrollar los productos para clientes concretos, reutilizando los recursos creados en la Ingeniería del Dominio [6].

En trabajos anteriores [7-9] se ha descrito un marco automatizado para la gestión de las pruebas en el contexto de LPS, el cual se resume en la Figura 1. Las pruebas dirigidas por modelos (*Model-driven testing*) requieren la derivación sistemática y en lo posible automatizada de las pruebas a partir de modelos [10]. El marco propuesto se basa en Pruebas Dirigidas por Modelos siguiendo un enfoque MDA aplicado a las Líneas de Productos Software. Los modelos de prueba se generan automáticamente a partir de modelos UML, el enfoque utiliza metamodelos estándares del OMG para realizar dicha transformación, tanto para el metamodelo fuente (UML), el de destino (Perfil de Pruebas UML) como para el lenguaje de transformación (QVT).

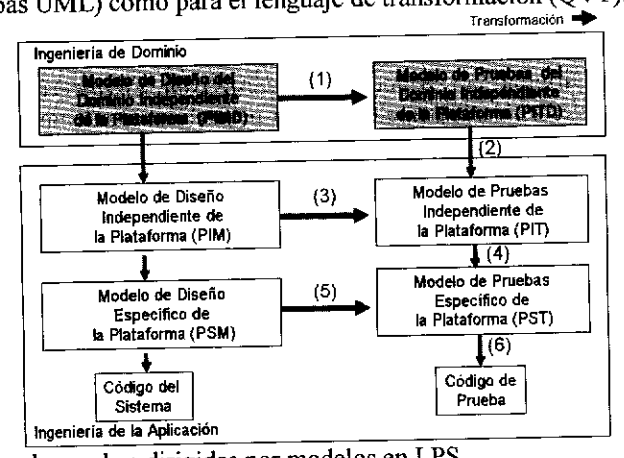


Figura 1. Marco de pruebas dirigidas por modelos en LPS

Dicho marco aún no ha sido completamente desarrollado, en [8] se describen las transformaciones QVT que generan automáticamente el modelo de pruebas para la LPS, lo cual se corresponde con la flecha 1 de la Figura 1.

La principal aportación de este artículo es definir la forma en que el modelo de prueba para cada producto es derivado en forma automática a partir de los modelos de prueba de la línea, esto se corresponde con la flecha 2 de la Figura 1. Este trabajo constituye una contribución importante en el camino de contar con el marco automatizado completamente desarrollado.

El artículo se organiza de la siguiente manera: la sección 2 resume los trabajos relacionados, la sección 3 describe el marco automatizado para pruebas definido en trabajos anteriores, la sección 4 describe la propuesta para generar el modelo de prueba de cada producto de la LPS. Finalmente, en la sección 5, se presentan las conclusiones y el trabajo futuro.

2 Trabajos relacionados

Las pruebas en el contexto de las LPS incluyen la derivación de casos de prueba para la línea y para cada producto específico. Para reducir el costo de crear los casos de prueba, deben aprovecharse las posibilidades que brinda la variabilidad en LPS. Esta sección revisa los trabajos existentes y se estructura en dos secciones: la primera resume los trabajos relacionados con las pruebas en LPS y en MDE, la segunda resume brevemente el Perfil de Pruebas de UML.

2.1 Pruebas en LPS y MDE

En general, las propuestas para la derivación de casos de prueba en LPS utilizan como base para el modelado UML ó artefactos de UML. Todas ellas contemplan la trazabilidad entre la Ingeniería del Dominio y del Producto en LPS. Sin embargo, no toman en cuenta las capacidades de los marcos de modelado estándares específicamente diseñados para las pruebas tales como el Perfil de Pruebas de UML[11]. Además, dado que los enfoques basados en modelos son especialmente útiles para SPL, es especialmente aconsejable el uso de lenguajes de transformación estándares para la generación de los modelos en forma automática. Una descripción completa de los trabajos existentes sobre pruebas en LPS puede encontrarse en [12], a continuación se resumen brevemente los trabajos que definen metodologías para la derivación de casos de prueba en LPS.

Nebut et al. [13] obtienen casos de prueba a partir de diagramas de secuencia de alto nivel, que luego se utilizan para generar automáticamente casos de prueba para cada producto. Bertolino et al. [14] proponen la metodología PLUTO (Product Line Use Case Test Optimization) que extiende la descripción textual de los casos de uso con un conjunto de etiquetas de variabilidad que son usadas para luego derivar los casos de prueba de la LPS. Kang et al. [15] extiende la notación del diagrama de secuencia para representar los escenarios de los casos de uso con variabilidad. Reuys et al. [16] presentan ScenTED (Scenario-based Test case Derivation), donde el

modelo de pruebas (representado con diagramas de actividad) se construye a partir de las funcionalidades y usan diagramas de secuencia para representar el escenario de prueba. Olimpiew el al. [17] propone el método PLUS (Product Line UML-based Software engineering) de tres fases: creación del diagrama de actividad a partir de los casos de uso, creación de tablas de decisión a partir de los diagramas de actividad y creación de plantillas de prueba a partir de las tablas de decisión.

Existen muchas propuestas para las pruebas dirigidas por modelos, pero muy pocas se enfocan en la automatización de las transformaciones entre modelos para las pruebas. Dai [18] argumenta que la filosofía de MDA (CIM, PIM, PSM) puede ser aplicada tanto al modelado del sistema como a su prueba. Los modelos de prueba pueden transformarse a partir de los modelos del sistema, una vez que el PIM se ha definido, se deriva el modelo de pruebas independiente de la plataforma (PIT). Este modelo, a su vez, puede transformarse en un modelo de prueba específico de la plataforma (PST). Por último, el PST puede transformarse en código de prueba [18]. Las transformaciones entre modelos de prueba de la propuesta de Dai han sido descritas a nivel teórico y no se han implementado. Zander et al [19] han implementado las transformaciones desde el PIT representado con el Perfil de Pruebas de UML 2.0 a código de prueba usando TTCN-3. Dichas transformaciones han sido realizadas usando Java. Baker et al. [10] definen modelos de prueba usando UML-TP donde las transformaciones se realizan manualmente en lugar de con un lenguaje de transformación. Naslavsky et al. [20] adaptan un modelo basado en flujo de control y lo usan para generar casos de prueba a partir de diagramas de secuencia. De acuerdo con su trabajo, tienen planificado usar el metamodelo de trazabilidad de ATL, pero la propuesta no ha sido aún automatizada.

2.2 Perfil de Pruebas UML

El perfil de Pruebas de UML 2.0 (UML Testing profile, UML-TP) [11] extiende UML 2.0 con conceptos específicos para las pruebas, agrupándolos en: arquitectura de pruebas, datos de pruebas, comportamiento de pruebas y tiempos de prueba[11]. La arquitectura de las pruebas contiene la definición de todos los conceptos necesarios para realizar las pruebas. En ellas se definen el contexto de las pruebas y el resto de los elementos necesarios para definir las pruebas. El comportamiento de las pruebas especifica las acciones y evaluaciones necesarias para la prueba. El caso de prueba es el concepto principal en el modelo de prueba, y su comportamiento puede ser descrito usando el concepto *Behavior de* UML 2.0, diagramas de secuencia, máquinas de estado o diagramas de actividad. En el Perfil, un caso de prueba (*test case*) es una operación de un contexto de prueba que especifica cómo un conjunto de componentes cooperan con el sistema bajo prueba (system under test, SUT) para alcanzar el objetivo de prueba [10].

3 Marco automatizado para pruebas dirigidas por modelos

En trabajos anteriores hemos definido un marco para las pruebas dirigidas por modelos que puede ser utilizado tanto en el desarrollo de software convencional como

el desarrollo de Líneas de Producto Software. Dicho marco automatiza la generación de modelos de prueba, siguiendo el enfoque de la Ingeniería Dirigida por Modelos. La Figura 1 muestra un esquema de las transformaciones automáticas entre modelos definidas (los modelos de prueba aparecen en el lado derecho). Se definen tres niveles de modelos para las pruebas:

- Modelo de Pruebas del Dominio Independiente de la Plataforma (PITD): este modelo se utiliza en el desarrollo de LPS, definiendo las pruebas a nivel de Ingeniería de Dominio. Los modelos de diseño de la LPS (PIMD) son transformados en modelos de prueba para la LPS. Dicho modelo de pruebas traza la variabilidad de la LPS para las pruebas. En este modelo de prueba estarán los casos de prueba comunes de la LPS, que prueban los aspectos comunes para todos los productos de la LPS, y los casos de prueba variables, que prueban funcionalidades que existen en algunos productos de la LPS pero no en todos.
- Modelo de Pruebas Independiente de la Plataforma (PIT): este modelo se utiliza en desarrollo convencional y en LPS, definiendo los artefactos de prueba a nivel independiente de la plataforma para cada producto. En el caso de LPS, este modelo pertenece a la Ingeniería del Producto y para generarlo es necesario resolver la variabilidad definida en el PITD. En este modelo se encontrarán los casos de prueba extraídos del PITD al resolver la variabilidad y los casos de prueba específicos de las funcionalidades que están presentes en un solo producto.
- Modelo de Pruebas Específico de la Plataforma (PST): para definir este modelo de pruebas se refina el modelo PIT con aspectos específicos de la plataforma donde se ejecutará el producto. Este modelo agrega detalles específicos de la plataforma que enriquecen los casos de prueba generados en el PIT.

Los modelos de pruebas son conformes con el Perfil de Pruebas de UML y las transformaciones se realizan con el lenguaje de transformación entre modelos QVT. Los modelos de diseño de la LPS contienen la variabilidad que permite luego derivar cada producto. Es necesario definir cómo se trazará la variabilidad desde los modelos de diseño del dominio de la línea (PIMD) al modelo de pruebas (PITD), y cómo se describirá dicha variabilidad en el modelo de pruebas. La Figura 2 muestra los modelos utilizados en cada nivel de PIM para el diseño y las pruebas. A continuación se describe cada uno de los modelos.

- Modelo de Variabilidad: Este modelo representa la variabilidad en la LPS y se representa mediante un Perfil UML del Modelo Ortogonal de Variabilidad (OVM) [21], dicho perfil puede ser consultado en [8].
- Diagrama de secuencia con variabilidad: Este modelo describe un escenario de un caso de uso. La variabilidad se representa utilizando el estereotipo *Variation Point* en el *CombinedFragment* del diagrama de secuencia UML, puede ser consultado en [7].
- Caso de prueba: Este modelo muestra el comportamiento para el caso de prueba derivado a partir del diagrama de secuencia del PIMD. El comportamiento del caso de prueba se representa mediante un diagrama de secuencia con variabilidad. El caso de prueba se transforma automáticamente utilizando QVT y traza la variabilidad al modelo de prueba. El modelo de prueba usa como

metamodelo el Perfil de Pruebas UML. Las transformaciones para generar el modelo de pruebas de la LPS pueden ser consultadas en [8, 9].

- Arquitectura de pruebas: Este modelo de clases muestra los elementos de la arquitectura de prueba, se transforma automáticamente utilizando QVT y traza la variabilidad al modelo de prueba. La arquitectura de pruebas usa como metamodelo el Perfil de Pruebas de UML. La generación automática de dicha arquitectura puede consultarse en [8, 9].

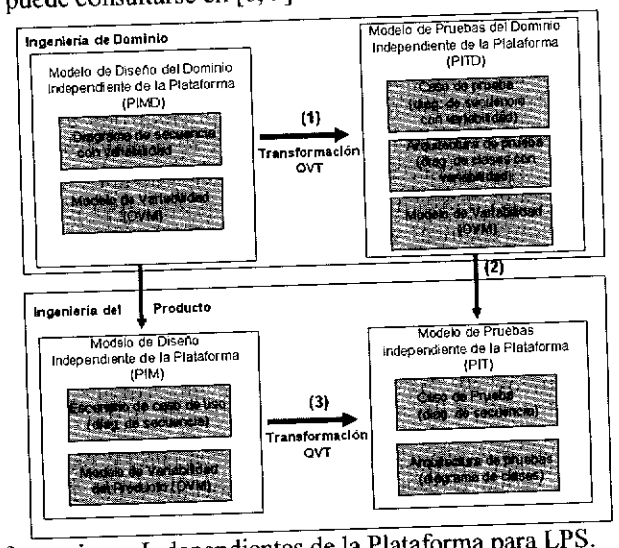


Figura 2. Transformaciones Independientes de la Plataforma para LPS.

A continuación se describe la aportación de este artículo, que consiste en generar automáticamente los casos de prueba para cada producto y se corresponde con la transformación indicada por la flecha 2 en la Figura 2.

4 Generación automática del modelo de prueba para cada producto

La generación de los casos de prueba para un producto específico de la línea requiere contemplar los siguientes tres escenarios:

- Casos de prueba comunes: Estos casos de prueba son comunes a todos los productos de la línea y por tanto, no contienen variabilidad. Se toman directamente del repositorio de casos de prueba de la línea.
- Casos de prueba con variabilidad: Estos casos de prueba requieren resolver la variabilidad para el producto a probar.
- Casos de prueba específicos: Estos casos de prueba probarán funcionalidad que existen solamente en este producto, dicha funcionalidad no se encuentra especificada en la LPS y debe especificarse en la Ingeniería del Producto.

Por tanto, el conjunto de casos de prueba para un producto estará compuesto por los casos de prueba de la LPS comunes a todos los productos, más los casos de prueba específicos generados para ese producto en particular, más los casos de prueba donde se resuelve la variabilidad.

Para la generación del modelo de prueba de cada producto en particular en forma automatizada se procede de la siguiente forma:

- Casos de prueba comunes: Estos casos de prueba se copian directamente al modelo de prueba del producto a partir del modelo de prueba de la línea.
- Casos de prueba con variabilidad: Estos casos de prueba requieren resolver la variabilidad para el producto a probar. Esta es la aportación principal de este artículo y es descrita en detalle en la sección 4.2.
- Casos de prueba específicos: Estos casos de prueba deben generarse de cero, ya que no existen en el modelo de prueba de la línea. Dicha generación se ha descrito en trabajos de investigación anteriores y puede ser consultado en [9], donde a partir de una diagrama de secuencia se genera automáticamente por medio de transformaciones QVT el caso de prueba y la arquitectura de prueba siguiendo el UML-TP.

4.1 Ejemplo: Línea de Producto de Juegos de Mesa

Esta sección describe un resumen del caso de estudio desarrollado para comprobar la aplicabilidad de nuestra propuesta. Se trata de un sistema distribuido cliente-servidor donde jugar a uno o más juegos de una familia de juegos de mesa. Este tipo de juegos comparten un amplio conjunto de características, tales como la existencia de un tablero, son jugados por uno o más jugadores, utilizan dados, existe la posibilidad de robar piezas, pueden incluir preguntas al jugador o políticas relacionadas con el paso del turno al siguiente jugador, etc. Aunque la mayoría de software desarrollado con LPS corresponde a aplicaciones empotradas [22, 23], este ejemplo es adecuado para ejemplificar cómo este paradigma relativamente nuevo puede ser aplicado también para el desarrollo de sistemas de software puro.

La LPS permite cuatro tipos de juegos de mesa: Ajedrez, Damas, Parchís y Trivial. Dado que no es posible mostrar la LPS completa, en la Figura 3 se muestran algunos de los puntos de variación y sus variantes siguiendo la notación OVM [21]. En la notación gráfica de OVM, los puntos de variación están representados por triángulos y sus variantes con un rectángulo. Las líneas punteadas representan las variantes opcionales (es decir, pueden ser omitidas en algunos productos), mientras que las líneas continuas representan variantes obligatorias (que están presentes en todos los productos). Las asociaciones entre las variantes pueden ser: *requires_V_V* y *excludes_V_V*, dependiendo de si una variante requiere o excluye a otra. Del mismo modo, las asociaciones entre una variación y un punto de variación puede ser: *requires_V_VP* y *excludes_V_VP*, donde se indica si una variación requiere o excluye un punto de variación.

La LPS de juegos de mesa tiene cuatro puntos de variación (ver Figura 3):

- Juego (*Game*): Este punto de variación tiene cuatro variantes: Parchís (Ludo), Damas (*Checkers*), Ajedrez (*Chess*) y Trivial.
- Oponente (*Opponent*): El jugador puede jugar contra el ordenador (*computer*), o contra otro jugador que se encuentre en línea (*Player On Line*).
- Jugadores (*Players*): El número mínimo de jugadores para todos los juegos es 2, pero, opcionalmente, algunos juegos se puede jugar de 3 a 4 o más de 4. El

Ajedrez excluye estas dos opciones, mientras que el Ludo excluye la opción de más de 4 jugadores.

- Tipo (*Type*): Algunos juegos utilizan dados (Dice) o realizan preguntas al jugador (*Quiz*). El Trivial utiliza ambas opciones y el Ajedrez y las Damas excluyen este punto de variación. El Ludo requiere los dados pero excluye las preguntas.

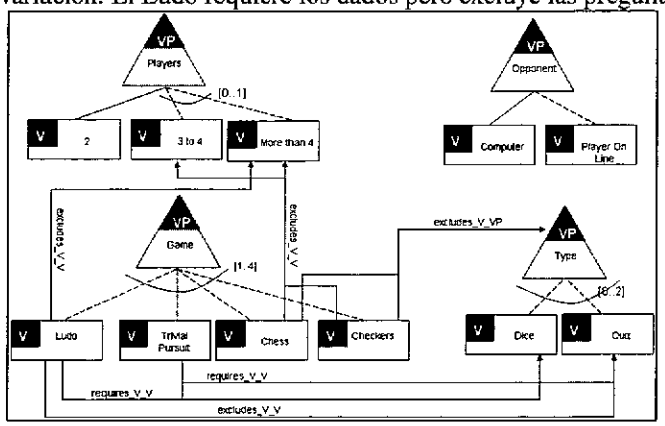


Figura 3. Puntos de variación y sus variantes para la LPS de Juegos de Mesa

La Figura 4 muestra dos modelos de la LPS de Juegos de Mesa, el modelo de arriba es un Modelo de Variabilidad conforme con el Perfil de OVM que representa los puntos de variación Juego (*Game*) y Tipo (*Type*), los cuales están estereotipados como *Variation Point*. Las variantes están estereotipadas como *Variant*. Las asociaciones entre los puntos de variación y sus variantes están estereotipadas como opcional (*optional*) y las restricciones entre los elementos también se muestran como asociaciones estereotipadas como *requires* o *excludes*. La ortogonalidad del modelo puede verse en las asociaciones entre el modelo de variabilidad (parte de arriba) y el diagrama de secuencia (parte de abajo), dicha asociación está estereotipada como realizad by. El modelo de variabilidad representado como Perfil de UML contiene la misma información que la notación gráfica de OVM de la Figura 3, la diferencia es que el Perfil UML para OVM nos permite asociar los puntos de variación y las variantes a cualquier artefacto de UML. En el caso del ejemplo podemos ver que las variantes Ludo, Trivial y Dice están relacionadas con el *CombinedFragment* del diagrama de secuencia.

El segundo modelo representado en la Figura 4 es un diagrama de secuencia, que representa la funcionalidad de Mover en un juego de mesa. Para esto, el jugador expresa su intención de mover la pieza y el sistema comprueba que tenga el turno. El *CombinedFragment* estereotipado como *Variation Point* es quien comprueba si el juego es Ludo o Trivial, en dicho caso el jugador debe tirar los dados antes de mover. Para los demás juegos (Ajedrez y Damas), esta funcionalidad se ignora y no forma parte del producto. Luego, el jugador mueve la pieza y se actualiza el tablero.

Para probar la funcionalidad "Mover Pieza" de la Figura 4, el comportamiento del caso de prueba conforme con el Perfil de Pruebas UML cumple los siguientes pasos (ver Figura 5):

1. Obtener los datos de prueba: El *testComponent Player_TComp* invoca el *DataSelector* en el *dataPool* que retorna los datos necesarios para probar la funcionalidad.
2. Ejecutar el caso de prueba en el SUT: El *testComponent Player_TComp* simula al actor y éste a su vez invoca la funcionalidad a probar en el SUT. El SUT es *Interface_SUT*.
3. Obtener el resultado del caso de prueba: El *testComponent* es el responsable de comprobar si el resultado retornado por el SUT es correcto, para esto utiliza las acciones de validación (*ValidationActions*) que son las encargadas de informar al árbitro (*arbiter*) el resultado de la prueba.

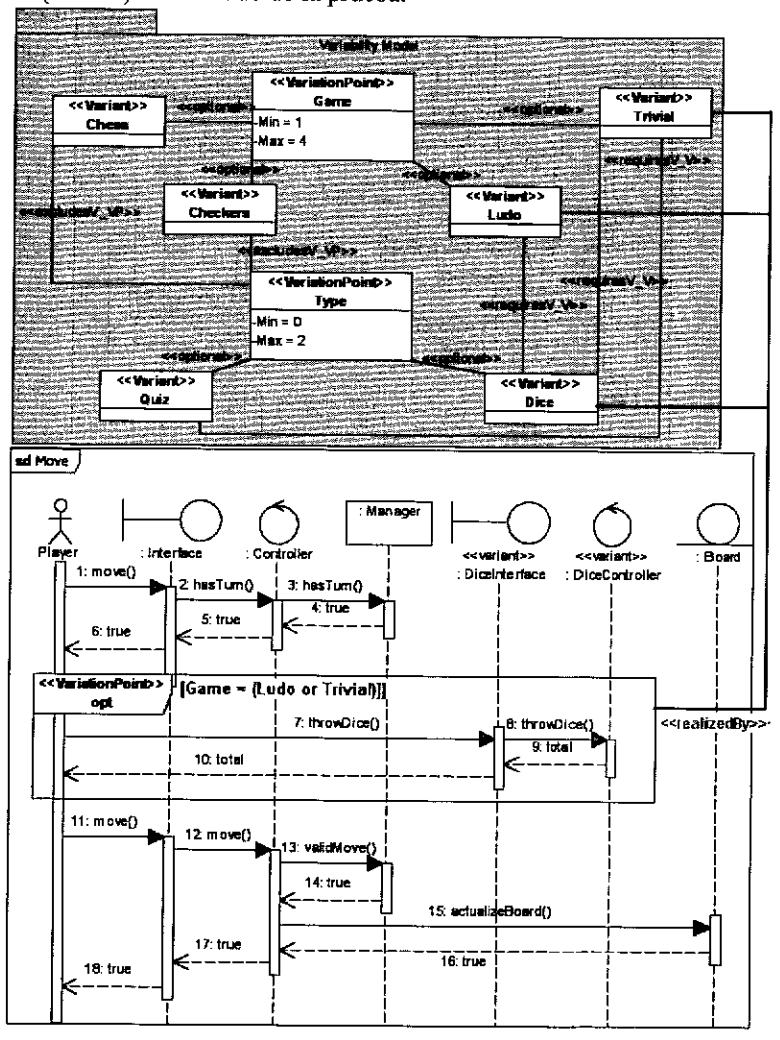


Figura 4. Diagrama de secuencia de Mover Pieza y su relación con el Modelo OVM

La Figura 5 muestra el caso de prueba resultante, dado que el diagrama de secuencia que vamos a probar tiene un *CombinedFragment* etiquetado *Variation Point* (ver

Figura 4), la prueba de esa porción de la funcionalidad también se realiza dentro de un *CombinedFragment*. Para cada *InteractionOperand* dentro del *CombinedFragment*, se realizan los mismos pasos descritos arriba: se obtienen los datos, se ejecuta la funcionalidad en el SUT y se obtiene el veredicto de la prueba. En el ejemplo se tiene un solo *InteractionOperand* con la guarda [Game=Ludo or Trivial], los datos de prueba son obtenidos invocando a *SelectData3()* que retorna el valor *totalE* que corresponde al resultado esperado para el caso de prueba. Luego, la operación *throwDice()* es invocada en el SUT llamado *DiceInterface_SUT*. El SUT retorna el resultado real *totalR*. Por ultimo, un *validationAction* compara el resultado esperado (*totalE*) con el resultado real (*totalR*) para obtener el veredicto.

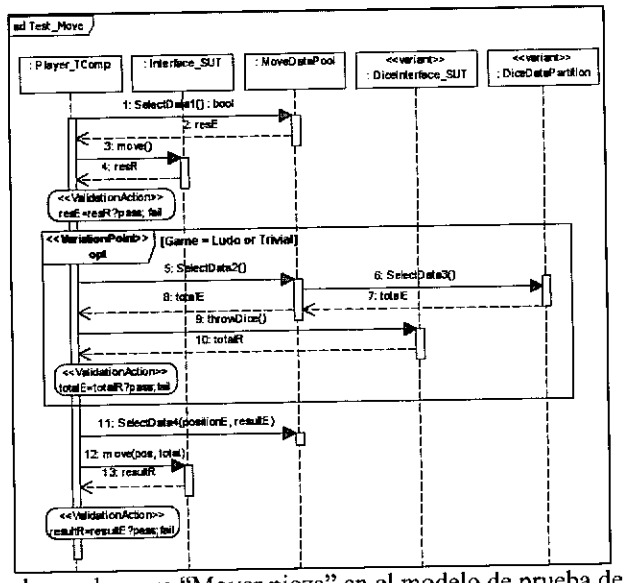


Figura 5. Caso de prueba para "Mover pieza" en el modelo de prueba de la LPS.

4.2 Derivación de los casos de prueba para cada producto resolviendo la variabilidad

En esta sección se describe la forma en que el modelo de pruebas de cada producto es generado a partir del modelo de pruebas de la LPS y se corresponde con la flecha 2 de la Figura 2. Para saber qué casos de prueba corresponden a cada producto primero es necesario conocer qué puntos de variación y variantes incluye cada producto. Esta información se encuentra en el Modelo de Variabilidad Ortogonal de cada producto (MVOP). El Modelo de Variabilidad de la línea es el que se muestra en la Figura 4, en la Figura 6 se describe el modelo de variabilidad para el producto Ajedrez y en la Figura 7 el modelo de variabilidad para el Trivial. En este último caso, el punto de variación *Type* es incluido junto con las variantes *Dice* (dados) y Quiz (pregunta).

Para derivar los casos de prueba de cada producto, tomando como ejemplo el caso de prueba Mover de la Figura 5, se debe resolver la variabilidad contenida en el *CombinedFragment*. En el caso del producto Ajedrez, se toma como entrada el modelo de variabilidad del producto de la Figura 6 y el caso de prueba de la Figura 5

y se obtiene el caso de prueba para el producto Ajedrez de la Figura 8. En este caso, el *CombinedFragment* entero es quitado del caso de prueba final.

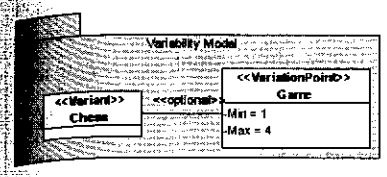


Figura 6 – Modelo de Variabilidad para el producto Ajedrez

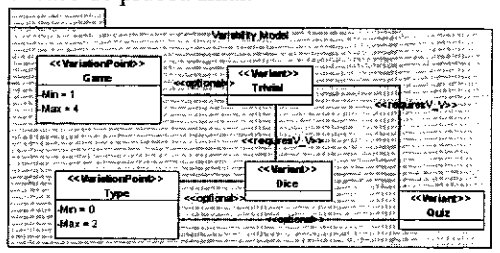


Figura 7 – Modelo de Variabilidad para el Producto Trivial

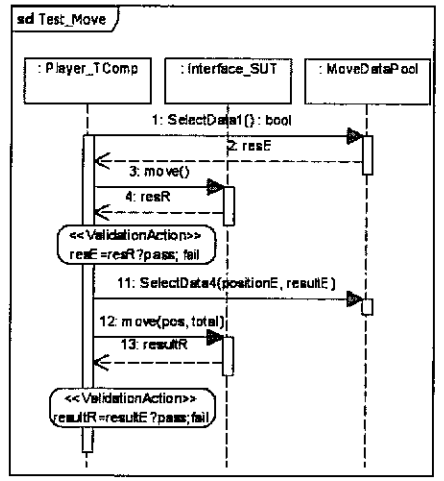


Figura 8. Caso de prueba para el producto Ajedrez

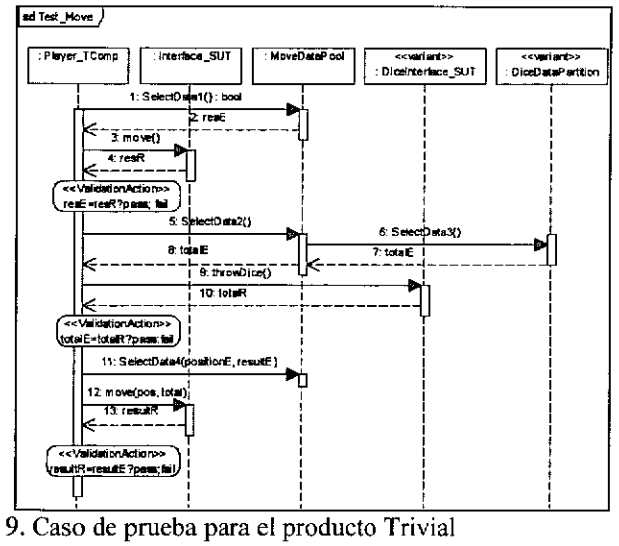


Figura 9. Caso de prueba para el producto Trivial

En el caso del producto Trivial, la Figura 9 muestra el caso de prueba, este caso de prueba tiene más mensajes que el anterior ya que se cumple la guarda de *CombinedFragment* y es por eso que los mensajes del punto de variación son incluidos para estos productos. Sin embargo, el *CombinedFragment* desaparece ya que no es necesario mantenerlo en el caso de prueba del producto final.

4.2 Automatización de las transformaciones mediante QVT

En esta sección se describe la forma en que las transformaciones QVT son desarrolladas. La Tabla 1 muestra el pseudocódigo de la transformación, los modelos de entrada son el Modelo de Variabilidad de la Línea (MVL), el Modelo de Variabilidad del Producto (MVP) y el modelo de prueba de la línea (MPL). Como salida se genera el modelo de prueba del producto (MPP).

```
1.  Crear el MPP
2.  Por cada caso de prueba (Interaction) en el MPL
3.      Si no tiene variabilidad
4.          Copiar el caso de prueba (Interaction) al MPP
5.      Sino
6.          Crear un nuevo caso de prueba en MPP
7.          Agregar las lifelines que no estén estereotipadas <<Variant>>
8.          Agregar los mensajes que no estén dentro de un CF estereotipado <<Variation Point>>
9.          Por cada CombinedFragment (CF) estereotipado <<Variation Point>>
10.             Buscar el Punto de Variación (PV) asociado en el MVL
11.             Si el PV asociado está incluido en el MVP
12.                 Por cada InteractionOperand del CombinedFragment
13.                     Buscar la Variante (V) asociada con el InteractionOperand en el MVL
14.                     Si la variante V está incluida en el MVP
15.                         Agregar las lifelines estereotipadas <<Variant>> asociadas con V
16.                         Agregar todos los mensajes incluidos en el InteractionOperand
17.                     FinSi
18.                 FinPor
19.             FinSi
20.         FinPor
21.     Fin Sino
22. Fin Por
```

Tabla 1 – Pseudocódigo para la generación del caso de prueba de cada producto

La línea 3 del pseudocódigo refiere a los casos de prueba comunes a todos los productos, en dicho caso, el caso de prueba se traslada al modelo de pruebas del producto. En caso contrario, para ese caso de prueba se crean las *lifelines* y mensajes que no tienen variabilidad y luego se resuelve la variabilidad existente en los *CombinedFragment*. En caso de que el *CombinedFragment* esté estereotipado como *<<Variation Point>>*, debe comprobarse que el punto de variación al que hace referencia se encuentre dentro del Modelo de Variabilidad del Producto. Si esto ocurre, para cada *InteractionOperand* dentro del *CombinedFragment*, se comprueba que la Variante a la cual hace referencia sea una Variante valida para el Producto. En dicho caso, se trasladan todos los mensajes contenidos en el *InteractionOperand* al caso de prueba del Producto. Cabe destacar que tanto el *CombinedFragment* como el *InteractionOperand* desaparecen del caso de prueba del Producto, como se mostró en las Figuras 8 y 9.

La Tabla 2 muestra el código QVT que transforma los mensajes dentro de un *InteractionOperand* cuando el *CombinedFragment* está estereotipado *Variation Point*. Se corresponde con las líneas 12, 13, 14 y 16 del pseudocódigo de la Tabla 1.

```
top relation translateInteractionOperandForProduct{
-- Por cada interaction
checkonly domain source i:uml::Interaction{
  -- Por cada CombinedFragment
    fragment = cf:uml::CombinedFragment{
    --Por cada InteractionOperand
    operand = io : uml::InteractionOperand {
    --Por cada mensaje del InteractionOperand,
    fragment = mes : uml::MessageOccurrenceSpecification{}
}}};
-- Busco asociacion entre el InteractionOperand y la Variante en el MVL
    checkonly domain source asoc:uml::Association{
      name = 'A_CF_' + io.name,
      ownedEnd = p1 : uml::Property {type = i },
      ownedEnd = p2:uml::Property{type = vp:uml::Class{} }};
-- Busco que exista la Variante en el modelo de Variab del Producto
    checkonly domain ovmp var:uml::Class{name = vp.name };
-- Creo el mensaje en el modelo de pruebas del Producto
    enforce domain target it:uml::Interaction{
    fragment = mes:uml::MessageOccurrenceSpecification{} };
when {  -- Comprueba que el estereotipo se aplicó al CombinedFragment
     st  = getStereotype ( 'Varition Point' );
     cf.isStereotypeApplied(st);      }
}
```

Tabla 2 – Transformación QVT para los mensajes incluidos en un InteractionOperand

5 Conclusiones

En el camino hacia el completo desarrollo del marco automatizado para la generación de casos de prueba en Líneas de Producto, este trabajo realiza la aportación de obtener los casos de prueba de la línea a nivel de PIM resolviendo la variabilidad para cada producto. Como trabajo futuro resta desarrollar las transformaciones a nivel de PSM y código para obtener el marco completo.

Agradecimientos

Esta investigación ha sido financiada por los proyectos: PRALIN (PAC08-0121-1374) y MECCA (PII2I09-00758394) del Cons. de Ciencia y Tecnol. de la Junta de Comunidades de Castilla-La Mancha y el proyecto PEGASO/MAGO (TIN2009-13718-C02-01) del MICINN y FEDER. Beatriz Pérez cuenta con una beca FPI de la Junta de Comunidades de Castilla-La Mancha (Orden de 13-11-2008).

Referencias

1. Mens, T. and P. Van Corp, A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Sciences, 2006. 152: p. 125-142.
2. Miller, J. and J. Mukerji, MDA Guide Version 1.0. 1, in Object Management Group. 2003.
3. Harmon, P., The OMG's Model Driven Architecture and BPM. Newsletter of Business Process Trends, 2004.
4. OMG, Meta Object Facility 2.0 Query/View/Transformation Specification, v1.0. 2007.
5. Clements, P. and L. Northrop, Salion, Inc.: A Software Product Line Case Study. 2002.
6. Díaz, S. and S. Trujillo, Líneas de Producto de Software, in Fábricas de Software: Experiencias, tecnologías y organización. 2008, Ra-ma: España.
7. Pérez Lamancha, B., M. Polo Usaola, and M. Piattini. Towards an Automated Testing Framework to Manage Variability Using the UML Testing Profile. in Workshop on Automation of Software Test (AST09) at ICSE. 2009,Vancouver,Canadá:IEEE Dig Library.
8. Pérez Lamancha, B., M. Polo Usaola, and J. García Rodriguez de Guzmán. Model-Driven Testing in Software Product Lines. in 25th International IEEE Conference on Software Maintenance (ICSM09). 2009. Edmonton, Canadá: ISSN: 1063-6773 pp. 511-514
9. Pérez Lamancha, B., et al. Automated Model-based Testing using the UML Testing Profile and QVT. in MODEVVA'09. 2009. Denver, Colorado.
10. Baker, P., et al., Model-Driven Testing: Using the UML Testing Profile. 2007: Springer.
11. OMG, UML 2.0 Testing Profile Specification. 2004, Object Management Group.
12. Pérez Lamancha, B., M. Polo Usaola, and M. Piattini. Software Product Line Testing, A systematic review. in 4th International Conference on Software and Data Technologies (ICSOFT09). 2009. Sofia, Bulgaria: To be published.
13. Nebut, C., et al., Automated requirements-based generation of test cases for product families. Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on, 2003: p. 263-266.
14. Bertolino, A., S. Gnesi, and A. di Pisa, PLUTO: A Test Methodology for Product Families. Software Product-family Engineering: 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003: Revised Papers, 2004.
15. Kang, S., et al., Towards a Formal Framework for Product Line Test Development. Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on, 2007: p. 921-926.
16. Reuys, A., et al., Model-based System Testing of Software Product Families. Pastor, O.; Falcao e Cunha, J.(Eds.): Advanced Information Systems Engineering, CAiSE, 2005: p. 519–534.
17. Olimpiew, E. and H. Gomaa, Customizable Requirements-based Test Models for Software Product Lines. International Workshop on Software Product Line Testing, 2006.
18. Dai, Z. Model-Driven Testing with UML 2.0. in Second European Workshop on MDA with an emphasis on Methodologies and Transformations. 2004. Canterbury, England.
19. Zander, J., et al., From U2TP models to executable tests with TTCN-3-an approach to model driven testing. Lecture Notes in Computer Science, 2005. 3502: p. 289-303.
20. Naslavsky, L., H. Ziv, and D.J. Richardson, Towards traceability of model-based testing artifacts, in Proceedings of the 3rd international workshop on Advances in model-based testing. 2007, ACM: London, United Kingdom. p. 105-114.
21. Pohl, K., G. Böckle, and F. Van Der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques. 2005: Springer.
22. Kim, K., et al. ASADAL: a tool system for co-development of software and test environment based on product line engineering. in Proceedings of the 28th international conference on Software engineering. 2006: ACM New York, NY, USA.
23. Kishi, T. and N. Noda, Formal verification and software product lines. Communications of the ACM, 2006. 49(12): p. 73-77.