REICIS. Revista Española de Innovación, Calidad e Ingeniería del Software
E-ISSN: 1885-4486
reicis@ati.es
Asociación de Técnicos de Informática
España

Pérez Lamancha, Beatriz; Polo, Macario
Generación automática de casos de prueba para Líneas de Producto de Software
REICIS. Revista Española de Innovación, Calidad e Ingeniería del Software, vol. 5, núm. 2, septiembre, 2009, pp. 17-27
Asociación de Técnicos de Informática
Madrid, España

Revista Española de Innovación, Calidad e Ingeniería del Software, Vol.5, No. 2, 2009

Generación automática de casos de prueba para Líneas de Producto de Software

Beatriz Pérez Lamancha
Centro de Ensayos de Software (CES), Universidad de la República
Montevideo, Uruguay
bperez@fing.edu.uy

Macario Polo
Grupo Alarcos, Departamento de Tecnologías y Sistemas de Información,
Universidad de Castilla-La Mancha, Ciudad Real, España
macario.polo@uclm.es

Resumen

La generación automática de casos de prueba a partir de modelos de diseño para Líneas de Producto de Software requiere definir los mecanismos para gestionar la variabilidad en las pruebas y su trazabilidad a los demás artefactos de desarrollo. En este trabajo, los casos de prueba se generan automáticamente mediante el lenguaje de transformación QVT a partir de diagramas de secuencia extendidos para representar la variabilidad en la LPS. La trazabilidad entre los distintos modelos es gestionada mediante la definición de un Perfil de UML para el Modelo de Variabilidad Ortogonal.

Palabras clave: Línea de Productos Software, Ingeniería Dirigida por Modelos, Pruebas.

Automatic test case generation for software product lines

Abstract

The automatic generation of test cases from design models in Software Product Lines requires defining testing mechanisms for managing variability and traceability to other development artifacts. In our proposal, test cases are generated using the transformation language QVT from extended sequence diagrams representing the variability in the SPL. Traceability between different models is managed by defining a UML Profile for the Orthogonal Variability Model.

Key words: Software Product Line, Model Driven Engineering, Testing

Pérez-LaMancha, B. y Polo, M.," Generación automática de casos de prueba para Líneas de Producto de Software", REICIS, vol. 5, no.2, 2009, pp.17-27. Recibido: 22-6-2009; revisado: 6-7-2009; aceptado: 31-7-2009

1. Introducción

Una línea de productos software (LPS) 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" [3]. 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 al dominio y su mecanismo de variabilidad, y la Ingeniería de la Aplicación donde se desarrollan los productos concretos reutilizando los recursos creados en la Ingeniería del Dominio.

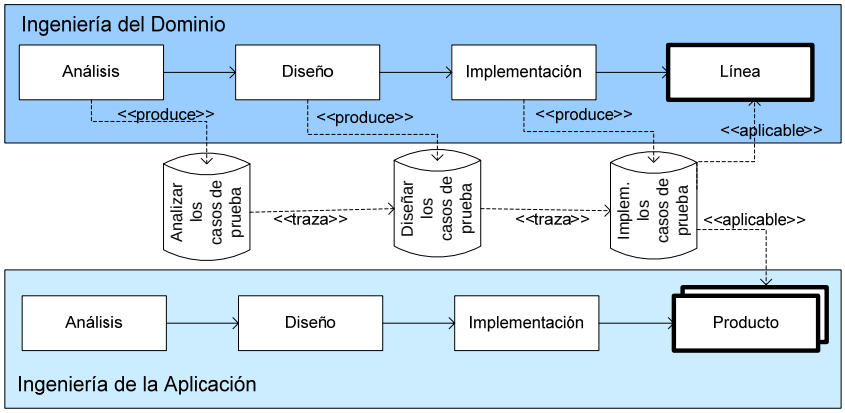

Figura 1. Proceso de desarrollo para líneas de producto software

La Figura 1 ilustra la forma en que se puede tomar ventaja de la trazabilidad para construir y reutilizar casos de prueba: suponiendo por simplicidad un modelo clásico de cascada, para cada etapa de la Ingeniería del Dominio es posible producir casos de prueba que pueden ser trazados de uno a otro nivel de abstracción (es decir, del Análisis hacia el Diseño), y además, para los nuevos artefactos producidos en cada etapa, se generan nuevos casos de prueba, que progresivamente, van enriqueciendo el conjunto de pruebas (es decir, casos de prueba producidos a partir de modelos de diseño completan los casos de prueba producidos desde el Análisis). Al final, todo este testware debe aplicarse a cada uno de los productos de la línea, y a la línea en sí.

Para hacer frente a un proceso bien gestionado, que permita la derivación de casos de prueba a partir de otros previamente definidos, y a partir de artefactos del análisis o diseño, es necesario el uso de herramientas y métodos estandarizados de la Ingeniería de Software.

Las pruebas dirigidas por modelos (Model-driven testing) requieren la derivación sistemática y en lo posible automatizada de las pruebas a partir de modelos [1]. En este artículo se presenta una estrategia para la generación de casos de prueba para LPS donde se define cómo gestionar la variabilidad en los artefactos de prueba y su trazabilidad al resto de los artefactos de desarrollo de la LPS. El trabajo aquí presentado es la continuación de investigaciones anteriores [12, 13]. En [13] se describe cómo se pueden generar modelos de prueba basados en el Perfil de pruebas de UML (UML-TP)[9] en forma automática a partir de las funcionalidad descritas como diagramas de secuencia para desarrollo de software convencional, utilizando como lenguaje de transformación entre modelos Query/View/Transformation (QVT)[10]. En [12] se presentan las extensiones al diagrama de secuencia y al UML-TP para gestionar la variabilidad en los modelos de prueba para LPS. En la propuesta presentada aquí se describe cómo las transformaciones entre modelos pueden extenderse a las pruebas en LPS, resolviendo la trazabilidad mediante el Modelo de Variabilidad Ortogonal (OVM) [14] para gestionar la variabilidad, mostrando un ejemplo de su uso en la LPS de Juegos de Mesa. El artículo se organiza de la siguiente manera: la sección 2 resume los trabajos relacionados, la sección 3 describe la Línea de Producto de Software de Juegos de Mesa donde se pone en práctica la propuesta, la sección 4 presenta la propuesta para generación de casos de prueba en LPS. Finalmente, en la sección 5 se presentan las conclusiones.

2. Trabajos relacionados

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, se diferencian de la propuesta presentada aquí en que 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[9] ni utilizan lenguajes de transformación entre modelos como QVT. Una descripción completa de los trabajos existentes sobre pruebas en LPS puede encontrarse en [11], a continuación se resumen brevemente los trabajos que definen metodologías para la derivación de casos de prueba en LPS.

Nebut et al. [7] 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. [2] 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. [4] extiende la notación del diagrama de secuencia para representar los escenarios de los casos de uso con variabilidad. Reuys et al. [15] 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. [8] 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.

Dado que nuestro trabajo utiliza el Perfil de Pruebas de UML 2.0 (UML Testing profile, UML-TP) [9], a continuación se resume brevemente. El UML-TP 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[9]. 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 [1].

3 Ejemplo: Línea de Producto de Juegos de Mesa

Esta sección describe un resumen del caso de estudio, 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 [5, 6], 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 2 se muestran algunos de los puntos de variación y sus variantes siguiendo la notación OVM [14]. 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 2):

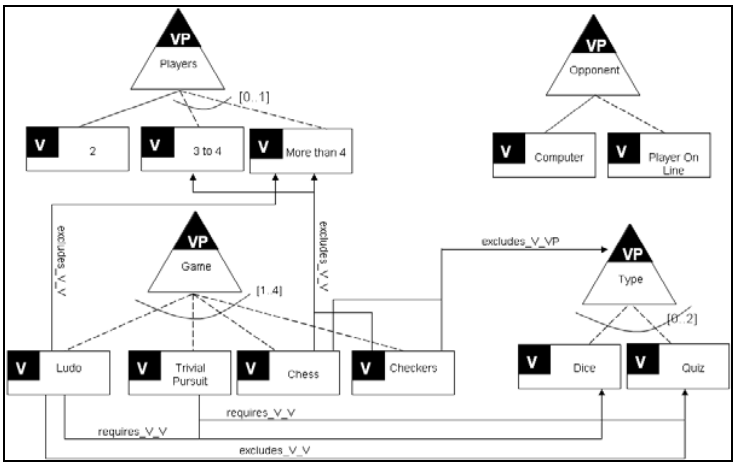

Figura 2. Puntos de variación y sus variantes para la LPS de Juegos de Mesa

- 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, 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.

4. Generación de casos de prueba para Líneas de Producto Software

La generación automática de los casos de prueba para Líneas de Producto de Software requiere definir los mecanismos para gestionar la variabilidad. Los artefactos de diseño de la LPS contienen la variabilidad que permite luego derivar cada producto. La trazabilidad de dicha variabilidad a los modelos de prueba, ayuda al reuso de las pruebas y a la mantenibilidad de la LPS. Es necesario definir cómo se trazará la variabilidad desde los artefactos de diseño del dominio de la LPS a los artefactos de prueba. En nuestra propuesta, para la generación automática de casos de prueba en LPS se utilizan modelos que definen las funcionalidades de la LPS como entrada y se generan automáticamente los modelos de prueba que contienen los casos de prueba para esas funcionalidades. Las transformaciones entre modelos se realizan utilizando el lenguaje QVT, la Figura 3 muestra los modelos fuente y destinos en la transformación. En la Ingeniería del Dominio de la LPS, el diagrama de secuencia UML con variabilidad y el modelo de variabilidad OVM se transforman mediante QVT en los siguientes modelos: Modelo de Variabilidad OVM enriquecido con las relaciones a los elementos de prueba, Diagrama de secuencia describiendo el caso de prueba y Arquitectura de Pruebas.

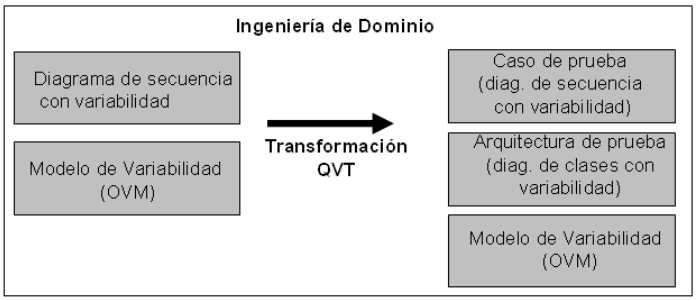

Figura 3. Generación de las pruebas usando QVT

Se utilizan los siguientes modelos para gestionar la variabilidad a nivel de la ingeniería de Dominio en LPS:

- Diagrama de secuencia con variabilidad: Este modelo describe un escenario de un caso de uso representado como un diagrama de secuencia UML. La variabilidad se representa utilizando el estereotipo *Variation Point* en el *CombinedFragment* del diagrama de secuencia UML. El diagrama de secuencia con variabilidad se utiliza tanto para describir la funcionalidad a probar, como para describir el comportamiento del caso de prueba para esa funcionalidad.
- Modelo Ortogonal de Variabilidad: El Modelo de Variabilidad Ortogonal (OVM) [14] contiene los datos sobre los puntos de variación (VP) y sus variantes. OVM permite representar las dependencias entre puntos de variación y elementos variables, así como las asociaciones entre los VP y las variantes con los artefactos de desarrollo. Dado que nuestra propuesta se encuentra enmarcada dentro de UML, fue necesario definir el Modelo Ortogonal de Variabilidad como Perfil de UML.
- Arquitectura de pruebas: Es un diagrama de clases donde se describen los elementos del UML-TP que se requieren para generar el caso de prueba.

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 VariationPoint. 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 realized by.

El modelo de variabilidad representado como Perfil de UML contiene la misma información que la notación gráfica de OVM de la Figura 2, 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 vemos que las variantes Ludo, Trivial y Dice están relacionadas con el *CombinedFragment* del diagrama de secuencia. Es posible porque en el Perfil UML para OVM se asocian con el concepto *Element* de *UML MetaClass*.

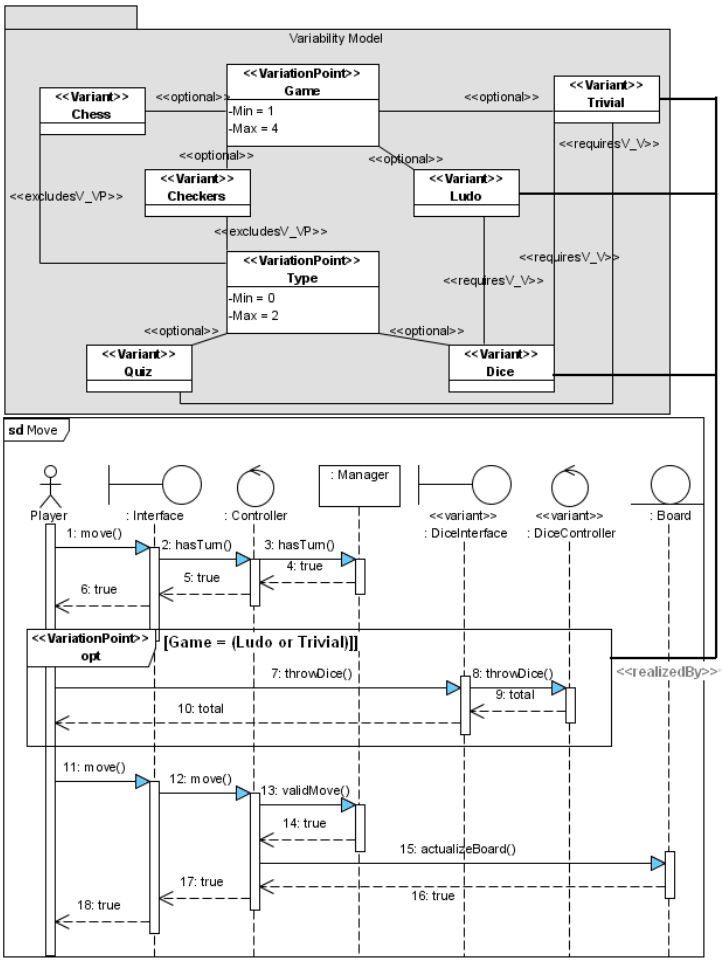

Figura 4. Diagrama de secuencia de Mover Pieza y su relación con el Modelo OVM

El segundo modelo representado en la Figura 4 es un diagrama de secuencia, que utiliza una extensión de las *Interaction* de UML para LPS, dicha extensión se basa en agregar el estereotipo *Variation Point* al *CombinedFragment* en los diagramas de secuencia. En el

ejemplo de la Figura 4, el diagrama de secuencia 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. Una descripción completa de la extensión definida para manejar la variabilidad en los diagramas de secuencia puede consultarse en [12].

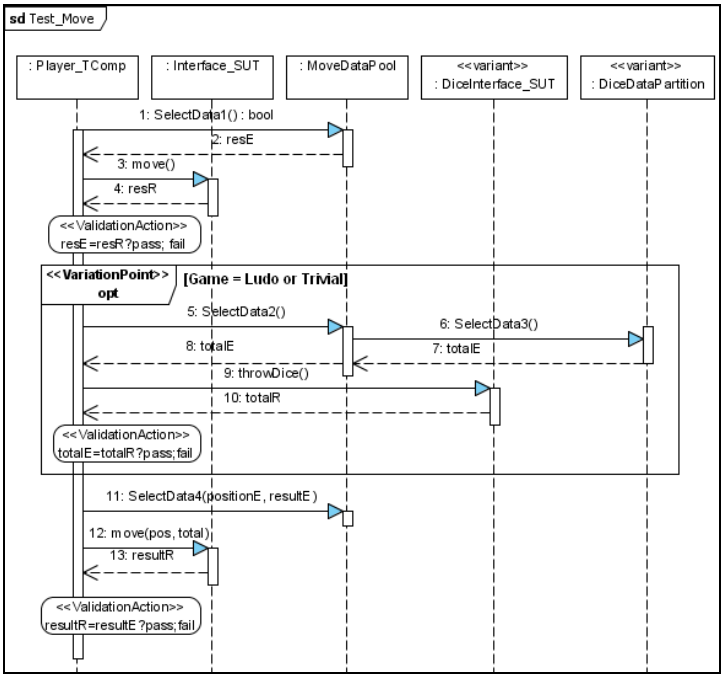

Figura 5.Caso de prueba para la funcionalidad “Mover pieza”

A partir de los modelos fuente de la Figura 4, se debe generar automáticamente mediante QVT el caso de prueba y la arquitectura de prueba. Además el modelo OVM debe ser actualizado con la trazabilidad de la variabilidad a los artefactos de prueba. El comportamiento del caso de prueba conforme con el Perfil de Pruebas UML se muestra en la Figura 5 y cumple los siguientes pasos:

- Obtener los datos de prueba: El *testComponent Player_TComp* invoca el *DataSelector* en el *dataPool* que retorna los datos necesarios para probar la función.
- Ejecutar el caso de prueba en el SUT: el *testComponent Player_TComp* simula al actor y éste, a su vez, invoca la función a probar en el *Interface_SUT.*

- 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.

Dado que el diagrama de secuencia que vamos a probar tiene un *CombinedFragment* etiquetado *Variation Point* (ver Figura 4), puede verse en la Figura 5 que la prueba de esa porción de la funcionalidad también se realiza dentro de un *CombinedFragment* etiquetado *Variation Point.* Para cada *interaction operand* 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 interaction operand 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. Han sido desarrolladas las transformaciones QVT que permiten generar los artefactos de prueba con variabilidad para LPS.

3. Conclusiones

Se ha presentado una propuesta para la generación automática de casos de prueba para LPS. La propuesta gestiona la variabilidad en los modelos de prueba y mantiene la trazabilidad con los modelos de diseño de la LPS a nivel de Ingeniería de Dominio. Como trabajo futuro se encuentra la derivación de los modelos de prueba a nivel de la Ingeniería de la Aplicación, lo que implica resolver la variabilidad para cada producto.

Agradecimientos

Este trabajo ha sido parcialmente soportado por los proyectos de la Junta de Comunidades de Castilla-La Mancha: PRALIN (PAC08-0121-1374) y MECCA (PII2I09-0075-8394). Se ha contado además con el apoyo de la Beca de Investigación. Orden de 13-11-2008 de la Consejería de Educación y Ciencia. Junta de Comunidades de Castilla-La Mancha (JCCM).

Referencias

[1] Baker, P., et al., Model-Driven Testing: Using the UML Testing Profile. 2007: Springer.

[2] Bertolino, A., S. Gnesi, y A. di Pisa, PLUTO: A Test Methodology for Product Families. Software Product-family Engineering: 5th International Workshop, 2003.

[3] Clements, P.y L. Northrop, Salion, Inc.: A Software Product Line Case Study. 2002, DTIC Research Report ADA412311.

[4] Kang, S., et al., Towards a Formal Framework for Product Line Test Development. 7th IEEE International Conference on Computer and Information Technology, 2007.

[5] Kim, K., et al. ASADAL: a tool system for co-development of software and test environment based on product line engineering, 28th Inter. Conf. on Soft. Engin. 2006.

[6] Kishi, T.y N. Noda, Formal verification and software product lines. Communications of the ACM, 2006. 49(12): p. 73-77.

[7] Nebut, C., et al., Automated requirements-based generation of test cases for product families. 18th IEEE International Conference on Automated Software Engineering, 2003.

[8] Olimpiew, E.y H. Gomaa, Customizable Requirements-based Test Models for Software Product Lines. International Workshop on Software Product Line Testing, 2006.

[9] OMG, UML 2.0 Testing Profile Specification. 2004, Object Management Group.

[10] OMG, Meta Object Facility 2.0 Query/View/Transformation Specification, v1.0. 2007.

[11] Pérez Lamancha, B., M. Polo Usaola, y M. Piattini. Software Product Line Testing, A systematic review, 4th International Conference on Software and Data Technologies, 2009.

[12] Pérez Lamancha, B., M. Polo Usaola, y M. Piattini. Towards an Automated Testing Framework to Manage Variability Using the UML Testing Profile, Workshop on Automation of Software Test. IEEE Catalog CFP0915D, ISBN978-1-4244-3711-5. Vancouver, Canadá, 2009.

[13] Pérez Lamancha, B., et al., Propuesta para pruebas dirigidas por modelos usando el perfil de pruebas de UML 2.0. REICIS, 2008. 4(4): p. 28-41. ISSN: 1885-4486.

[14] Pohl, K., G. Böckle, y F. Van Der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques. 2005: Springer.

[15] Reuys, A., et al., Model-based System Testing of Software Product Families. Advanced Information Systems Engineering, CAiSE, 2005.