REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 13, número 2
Págs. 94 - 123

Pruebas Dirigidas por Modelos para Sistemas Distribuidos: Estudio de caso Curso de Sistemas Distribuidos Universidad del Cauca

Luis C.Pito[1], Ruben J. Gaviria[1], Pablo A. Mage[2], Julio A. Hurtado[2] ‡

Fecha de Recibido: 21/06/2012 Fecha de Aprobación: 23/10/2012

Resumen

Actualmente el software y sus servicios derivados se evalúan de manera intuitiva y subjetiva, dependiendo del nivel de satisfacción de quien los consume. La calidad es una cualidad que cualquier servicio o producto debe poseer y mantener para asegurar su permanencia en el mercado. Particularmente, en el curso Laboratorio de Sistemas Distribuidos es demandante realizar la evaluación de los productos resultantes de las prácticas debido a su complejidad y el esfuerzo que demanda al evaluador. Para abordar esta problemática en este artículo se propone un proceso de pruebas *software* basado en el paradigma del desarrollo dirigido por modelos en el contexto de los Sistemas Distribuidos. Para soportar el enfoque se desarrolló una infraestructura de pruebas basada en modelos y se aplicó en un estudio de caso. Entre los principales resultados se muestra la aplicabilidad del modelo y la capacidad de reutilización de activos de prueba, así como las dificultades que deberán enfrentar las empresas al adoptar el enfoque por primera vez.

Palabras Claves: *Pruebas Software, Sistemas Distribuidos, Pruebas Dirigidas por Modelos, Calidad de Software, Metamodelos, Transformaciones de Modelo a Texto.*

[1] Universidad del Cauca - { lpito, rjgaviria, pmage, ahurtado}@unicauca.edu.co
[2] Los profesores Pablo Magé y Julio Ariel son profesores titulares de la Universidad del Cauca.

1. Introducción

A través del tiempo, en los entornos de desarrollo académicos e industriales de *software*, se ha trabajado para tratar de mejorar la calidad en todos los procesos que se encuentren involucrados en el desarrollo de productos *software*. Por medio de la experiencia obtenida en los proyectos exitosos y fallidos, ha ido evolucionando el concepto de calidad, de tal forma que hasta hoy en día, la calidad intenta ir en concordancia con las complejidades de los problemas que se resuelven por medio de la ingeniera de *software*. No obstante, los defectos, errores y fallas en el *software* seguirán existiendo, ya que no hay un proceso ideal que permita garantizar la ausencia de los mismos. Sin embargo, la competitividad y los esfuerzos de la industria del *software*, por generar productos de buena confianza en su funcionamiento, genera la necesidad de concebir productos de buena calidad[1]. Esto hace que surjan técnicas, métodos, metodologías y herramientas las cuales permiten llevar a cabo las actividades de verificación y validación. Estas validaciones se ocupan de controlar si el producto satisface las necesidades de los usuarios y uno de los mecanismos más utilizados han sido las pruebas que se le aplican al *software*, ayudando a los desarrolladores a reconocer y arreglar sus propios defectos en etapas tempranas de desarrollo. En las pruebas *software*, se busca ejecutar ciertos casos, por medio de la utilización directa de un actor denominado ***Tester***; estas pruebas se realizan con poca precisión, dado que este actor no puede garantizar la cobertura sistemática de la funcionalidad del sistema[2]. Además de ser un método costoso en recursos, ya que es necesario reproducir las pruebas cada vez que se realice un cambio por minúsculo que sea. La evolución de este enfoque manual de pruebas son las Pruebas Basadas en *Script*, en las cuales se fundamenta en la generación de un *script* encargado de ejecutar uno o más casos de prueba y transmitirle al sistema bajo prueba los valores de entrada, a continuación espera por los valores de salida los cuales son capturados por los puntos de control. Luego se verifica si las salidas del sistema son las esperadas. Esto implica que la generación de la prueba es una tarea de implementación en un lenguaje de programación estándar o un lenguaje de pruebas especiales como *Testing and Test Control Notation* (TTCN-3)[3]. Además de la gran cantidad de esfuerzo en mantener y actualizar los *Script* cada vez que los cambios lo requieran. Posteriormente, nace el enfoque de pruebas basado en modelos – MBT, el cual basa los diseños de casos, en un modelo abstracto en donde el diseñador de la prueba genera los posibles escenarios. La principal ventaja de este enfoque es generar una gran variedad de conjuntos de pruebas desde el mismo modelo del sistema a probar.

Este trabajo se enfoca en el uso de MBT, con el fin de aportar mejoras en la calidad a nivel de producto y más específicamente en las pruebas funcionales que se realizan en el proceso de pruebas durante las

prácticas de la asignatura de Sistemas Distribuidos - SD, orientada en séptimo semestre de la carrera de Ingeniería de Sistemas de la Universidad del Cauca. Dentro de esta asignatura se hace necesario que se apliquen los conceptos teóricos aprendidos en clase, en programas generados por los estudiantes.

Tradicionalmente, la evaluación funcional de los productos *software* generados en las prácticas, se hacía de manera manual, dando como resultado una tarea tediosa, para el evaluador (Docente) y el desarrollador (Estudiante). Este trabajo de investigación ha permitido generar un proceso práctico en el cual se utilizan los modelos previamente construidos en el desarrollo del sistema y se integran con un perfil UML, permitiendo modelar de manera abstracta las pruebas al SD. Posteriormente el modelo de pruebas obtenido es ingresado en un prototipo de herramienta de transformación de modelo a texto, el cual genera una aplicación encargada de desplegar una plataforma de agentes; Luego ejecuta las pruebas y evalúa los resultados obtenidos.

Lo novedoso de esta propuesta es la utilización de los modelos del sistema bajo prueba - SUT previamente construidos: específicamente los diagramas de clases y secuencia, aplicándoles el perfil UML creado, con el fin de reutilizar dichos artefactos generados en las metodologías de desarrollo de *software* como RUP, AUP, entre otras.

2. Metodología

Para realizar este trabajo de investigación se utilizó como fuente de recolección y documentación teórica la metodología de investigación documental propuesta por Serrano[4]. Para el desarrollo del prototipo se utilizó como metodología de desarrollo AUP [5]. A continuación se realizará una breve descripción de los conceptos utilizados en este trabajo de investigación:

2.1. Ingeniería Basada en Modelos

La ingeniería dirigida por modelos (MDE) es un enfoque que busca facilitar el proceso de desarrollo de *software* a través de la reutilización del conocimiento de refinamiento de modelos. Aunque las tecnologías han evolucionado rápidamente en forma positiva, también lo han hecho las necesidades de los usuarios y por tanto se requiere *software* cada vez más complejo, mantenible, con una alta calidad de servicio (QoS) y con rápida salida al mercado. El enfoque MDE aborda el problema con la combinación de tecnologías como: los “lenguajes de modelado específicos de dominio” (DSML) y los “motores de transformación y generación”[6], [7].

2.2. Pruebas Dirigidas por Modelos

Las pruebas dirigidas por modelos se refieren en general, a la derivación, la ejecución y análisis de casos de prueba para un sistema *software*, donde la derivación de los casos de prueba se realiza a partir de modelos formales o informales que especifican el sistema, y cuyo fin es medir la cantidad de errores o precisar la ausencia de estos.

***Definiciones de MBT*:** con el propósito de formalizar el concepto de pruebas dirigidas por modelos (MDT, *Model-Driven-Testing*) o pruebas basadas en modelos (MBT, *Model based testing*), en el cual, el presente documento no hace distinción entre las palabras "dirigidas" y "basadas", se presentan a continuación las definiciones obtenidas durante la investigación, yendo de las más simples hasta las más elaboradas:

- Binder [8], define como el "uso de un sistema *software*, que representa aspectos abstractos de un sistema bajo prueba para generar casos de pruebas".
- Utting [9], definen las pruebas basadas en modelos como "la derivación automática de casos de pruebas concretas desde modelos formales abstractos y su ejecución".
- Dias [10], definen las pruebas basadas en modelos como "una técnica de pruebas destinada a la generación automática de pruebas con modelos extraídos de los artefactos *software* producidos a lo largo del proceso de desarrollo".
- Tretmans[11], define que la especificación de pruebas dirigidas por modelos, es "probar la especificación del modelo del sistema, así como el sistema resultante automáticamente por medio de casos de prueba que son derivados desde el modelo de especificación del sistema y los requerimientos del mismo"

MBT busca la generación, ejecución y análisis automático de casos de prueba, por medio de algoritmos de generación que se basan en modelos estructurales o de comportamiento del sistema, o ambos; para mejorar la efectividad de las pruebas, y así lograr la reducción de costos de las mismas y mejorar la calidad del sistema en desarrollo.

Las pruebas dirigidas por modelos inician con un modelo verificado de un sistema, para luego intentar demostrar que la implementación real del sistema se comporta de acuerdo con el modelo[12].

2.3. Lenguaje de Modelado Unificado

El lenguaje de modelado (*Unified Modeling Lenguage* - UML) es una especificación de la OMG. Este lenguaje estándar se utiliza para

escribir planos del *software*, además de proporcionar un conjunto de mecanismos para visualizar, especificar, construir y documentar los artefactos de un sistema a través de varios diagramas que expresan aspectos estáticos y dinámicos de una aplicación. Los aspectos estáticos están relacionados con la estructura del sistema, el propósito es describir entidades del sistema y cómo están relacionadas. Por otro lado, se encuentran los aspectos dinámicos los cuales se refieren a la interacción de la aplicación, la creación y destrucción de objetos, sus conexiones con el tiempo y las transformaciones de los estados de estos objetos[13].

La versión actual 2.1 de la especificación de UML está compuesta por cuatro partes[14]:

- **UML *Superstructure*:** que especifica el lenguaje desde el punto de vista de los usuarios finales. Define seis (6) diagramas estructurales, tres (3) diagramas de comportamiento, cuatro (4) diagramas de interacción sí como los elementos que los componen.
- **UML *Infrastructure*:** Especifica la construcción de las bases de UML. Para esto define un núcleo de un metalenguaje que podrá ser utilizado para definir los metamodelos de otro lenguaje.
- **UML *Object Constraint Language***: OCL es un lenguaje que permite definir una amplia semántica de modelos UML mediante la definición de precondiciones, post-condiciones, invariantes y otras condiciones.
- **UML *Diagram Interchange*:** Extiende el metamodelo de UML a otro paquete adicional que modela información de carácter gráfico asociada a los diagramas, permitiendo el intercambio de modelos y conservando su representación original.

2.4. Transformación de Modelos

La transformación de modelos hace referencia al proceso de convertir un modelo en otro del mismo sistema [15]. Es por esto, que dentro del desarrollo dirigido por modelos, las transformaciones juegan un papel clave a la hora de especificar y generar el sistema en una plataforma concreta.

Las transformaciones se pueden clasificar en dos tipos: de modelo a modelo (del inglés, *Model to Model - M2M*) las cuales parten de un modelo origen y pretenden llegar a un modelo de destino, aumentando o disminuyendo el nivel de abstracción según sea el caso. Por otro lado, tenemos las transformaciones de Modelo a Texto (*Model to Text - M2T*) las cuales parten de un modelo origen y llegan a generar un archivo con cierto formato o código fuente de un lenguaje en particular.

2.5. Agentes *Software*

Un agente *software* "es un sistema informático que está situado en algún ambiente, y que es capaz de actuar con autonomía en este entorno con el fin de cumplir con sus objetivos de diseño"[16]. La arquitectura del prototipo de la herramienta encargada de probar, construido por los autores del presente trabajo de investigación, utilizan los agentes *software* para soportar el enfoque MBT, tomando como base una plataforma de agentes *software* que permite el despliegue del entorno de ejecución de pruebas, donde los agentes son actores reales del sistema bajo prueba. Debido a la autonomía de entorno, que tienen los agentes *software* son capaces de emular los clientes del SD, teniendo la posibilidad de realizar invocaciones remotas al servidor que en sí mismo podría ser también un agente. De esta manera la herramienta generada pretende automatizar el proceso de pruebas en un entorno muy similar al real.

2.6. Sistemas Distribuidos

Un Sistema Distribuido (SD) es "aquel en el que los componentes situados en una red de computadores, comunican y coordinan sus acciones sólo por el paso de mensajes. Esta definición conlleva a las siguientes características especialmente significativas de los SD: concurrencia de los componentes, la falta de un reloj global y fallos de los componentes independientes"[17]. Por lo anterior, se tiene que un SD consiste en un conjunto independiente de componentes corriendo de manera concurrente en diferentes máquinas e interactuando entre sí, comunicados por medio de una red para lograr un objetivo en común. Debido a esto se presenta una serie de problemas, haciendo la tarea de realizar las pruebas al SD más compleja; algunos de estos problemas son: escalabilidad de los criterios pruebas, generación de datos de prueba, pruebas redundantes, mecanismo de seguimiento y control en las pruebas, reproducibilidad de eventos, interbloqueos y condiciones de carrera, pruebas para la escalabilidad y el rendimiento del sistema además de la prueba de tolerancia a fallos [18]. Este conjunto de problemas no serán abordados en el presente trabajo de investigación porque no están contemplados en el alcance del mismo, pero se analizará la conformidad de las pruebas contra los requerimientos del sistema bajo prueba (SUT).

3. PRUDIMA: Un Proceso de Pruebas Basado en Modelos

Para modelar las pruebas de SD se construyeron dos metamodelos, los cuales permitieron abstraer separadamente los conceptos involucrados

en el modelamiento tanto de un SD como de sus pruebas, enmarcados en el paradigma de comunicación de invocación remota [17]. Sin embargo, para lograr utilizar los metamodelos construidos de manera conjunta con UML, se hizo necesario integrarlos a través de un perfil UML que se nombró "Perfil UML de PRUDIMA"; este permitió extender la semántica de algunos conceptos de UML y así, utilizar este lenguaje de modelamiento ampliamente usado en la definición de los modelos de pruebas para SD. Los metamodelos construidos fueron:

- Metamodelo para SD
- Metamodelo para el diseño de las pruebas a SD (PSD)

En la construcción de los anteriores metamodelos se usó la guía del trabajo realizado por García, Fuentes y Gómez [19], que define cuatro actividades para la creación de metamodelos de tipo Entidad-Relación: *determinar las características estructurales del lenguaje de modelamiento*, *escoger entre representación homogénea o heterogénea*, *escoger entre representación redundante y no redundante* y, por último, *definir el metamodelo.* En la Fig. 1 Actividades de creación de un metamodelo, se muestra el flujo de dichas actividades.

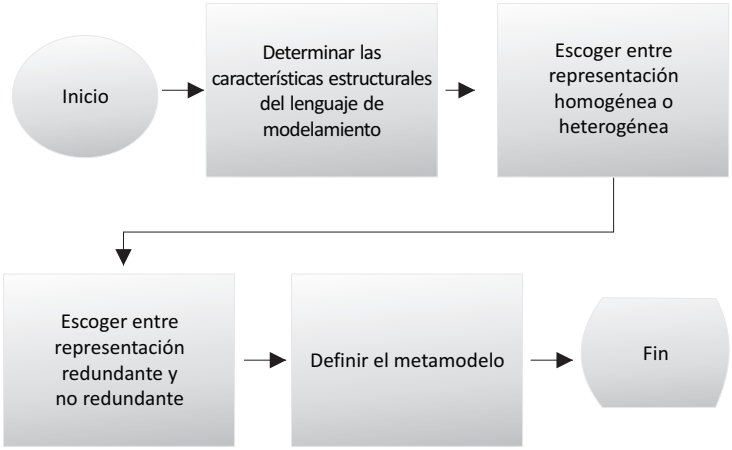

Fig. 1. Actividades de creación de un metamodelo.

La decisión de utilizar metamodelos basados en conexión y de tipo entidad-relación se debe a que la especificación actual de UML se basa en dichos enfoques [14], [19]. Se utilizó, además, el meta-metamodelo de *ECore* como lenguaje de modelamiento [20] para la construcción de los metamodelos. A continuación se detalla cada una de las actividades llevadas a cabo en la construcción de los metamodelos.

3.1. Metamodelo para SD (MSD)

El metamodelo para SD construido, se enfocó en los SD del tipo cliente/servidor (paradigma de comunicación de invocación remota), ya que abarcan las principales tecnologías utilizadas en las prácticas académicas descritas dentro del enfoque de este trabajo de investigación. Este contiene las entidades y relaciones necesarias que permiten modelar la estructura estática de los SD basándose en los conceptos del "lenguaje de definición de interfaces" (siglas en inglés IDL) [21], [22] y en el conocimiento de expertos, para abstraer los diferentes conceptos del dominio de los SD. A continuación, se describen las actividades llevadas a cabo para la definición de este metamodelo:

1) *Determinar las características estructurales del lenguaje de modelado:* al ser enfocado para integrarse con la especificación de UML se heredan las características estructurales. Las entidades, las relaciones y atributos del dominio de los SD tipo cliente/servidor son definidos por elementos del meta-metamodelo *Ecore[20]*, como son: *EClasses* para las entidades, *EReferences* y *EAttributes* para las relaciones finales y atributos respectivamente.

2) *Escoger entre representación homogénea y heterogénea*: debido a que la infraestructura de UML utiliza una representación heterogénea se sigue esta misma línea de representación[14]. *3) Escoger entre representación redundante y no redundante*: para este caso es adoptada la representación redundante, ya que permite tener una mejor navegabilidad desde las distintas entidades del metamodelo y su eficiente procesamiento.

4) *Definición del metamodelo*: en la Fig. 2 Vista de entidades metamodelo DS, se presentan las entidades de dominio, relaciones y atributos utilizados por el metamodelo. Los tipos de relación utilizados en el metamodelo son obtenidos a partir de las relaciones utilizadas en el lenguaje UML, como las asociaciones por agregación y composición.

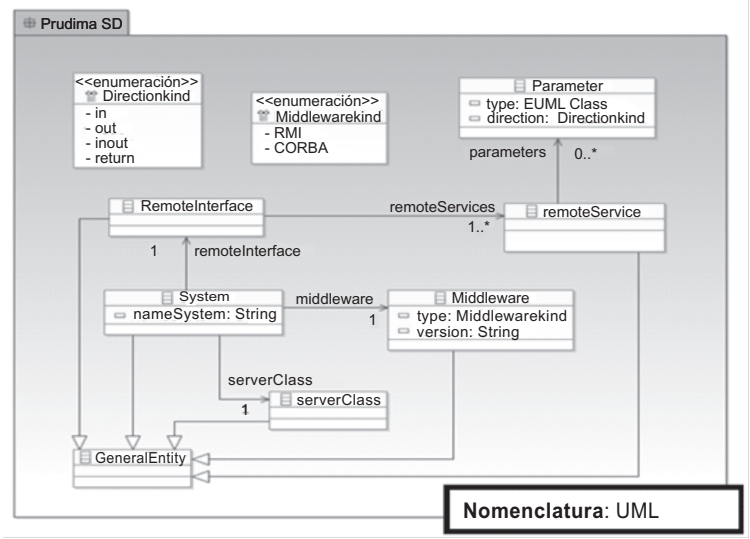

Fig. 2. Vista de entidades metamodelo DS.

3.2. Metamodelo para PSD (MPSD)

Se crea un metamodelo para las pruebas a los SD basándose en los conceptos de UML-TP[23]. La decisión de construir un metamodelo propio para las pruebas a los SD se debe a que UML-TP está concebido con conceptos genéricos para varios dominios de aplicación [24]; sin embargo, para el enfoque práctico que se le quiso dar a este trabajo de investigación, resultaba muy complejo de implementar. No obstante, se analizaron los conceptos como: *arquitectura de pruebas*, *datos de pruebas*, *comportamiento de pruebas*, *tiempo de pruebas*. A partir de estos, se definieron los elementos necesarios para poder tener un metamodelo de pruebas que permitiera modelar tanto pruebas unitarias como funcionales de una manera práctica y que dichos modelos pudiesen ser integrados con las instancias del metamodelo de SD.

Para la construcción de este metamodelo se utilizó la misma guía que para el metamodelo de SD y se tomaron las mismas decisiones respecto a las tres primeras actividades: “Determinar las características estructurales del lenguaje de modelado”, “Escoger entre representación homogénea y heterogénea”, “Escoger entre representación redundante y no redundante”, debido a que se siguen los lineamientos de UML. Lo que diferencia a este metamodelo del metamodelo de SD, se especifica en la cuarta actividad, que trata de la definición del mismo:

Definición del metamodelo: en la Fig. 3 Vista Metamodelo PDS , se definen las entidades de dominio que corresponden a: *Test, Parameter,*

MockClient, TestData, Step y Assertion. Los tipos de relaciones utilizadas por el metamodelo son tomados a partir de las relaciones definidas en UML. Luego, en la Figura 8 se presenta otra vista del metamodelo mostrando las relaciones entre las distintas entidades del dominio.

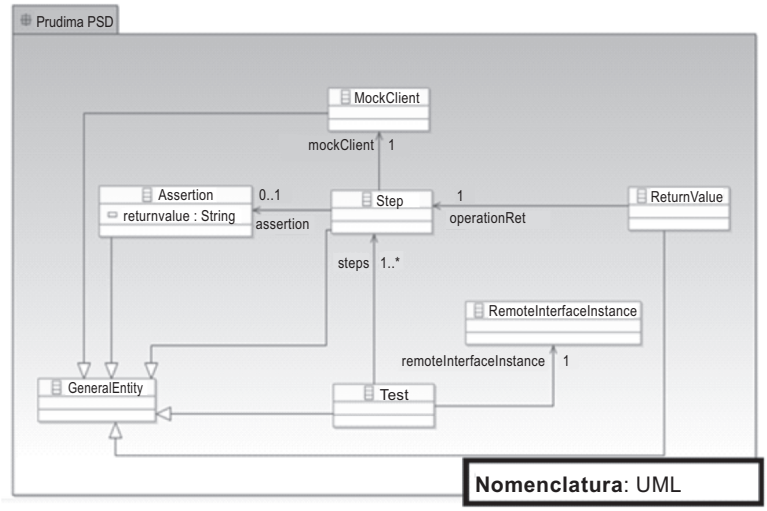

Fig. 3. Vista Metamodelo PDS.

3.3. Integración de los Metamodelos SD y PSD en el Perfil UML PRUDIMA

Un perfil UML permite extender la semántica de UML, la definición es "un perfil UML es definido como un paquete estereotipado *profile*, que puede extender un metamodelo u otro Perfil. Los perfiles UML son definidos en términos de tres mecanismos básicos: estereotipos, restricciones y valores etiquetados"[25].

Para lograr utilizar en conjunto los metamodelos previamente construidos, se hizo necesario definir un perfil UML que permitiera integrar las entidades definidas en estos y a su vez poder utilizar las entidades de UML.

Debido a lo anterior, se siguen los pasos que se encuentran en la guía [19]: *Paso 1*: se debe definir el conjunto de elementos que comprenden el sistema y las relaciones entre estos. Aquí se utilizan los metamodelos previamente construidos, de manera que las entidades definidas se conviertan en "Estereotipos" que permiten generalizar y extender la

semántica de algunas "metaclases" de UML. *Paso 2***:** se define el Perfil UML, donde se establecen los "Estereotipos" para cada elemento relevante del metamodelo. *Paso 3*: definir a qué "Metaclase" de UML será aplicado cada estereotipo. *Paso 4*: establecer los atributos que aparecen en el metamodelo como "Valores Etiquetados*"*, incluyendo sus tipos y valores iniciales. *Paso 5*: definir las "Restricciones Del Dominio", como por ejemplo la multiplicidad de las relaciones. Dando como resultado el Perfil UML PRUDIMA para pruebas a SD, que se presenta en la Fig. 4 Perfil UML PRUDIMA para pruebas a SD.

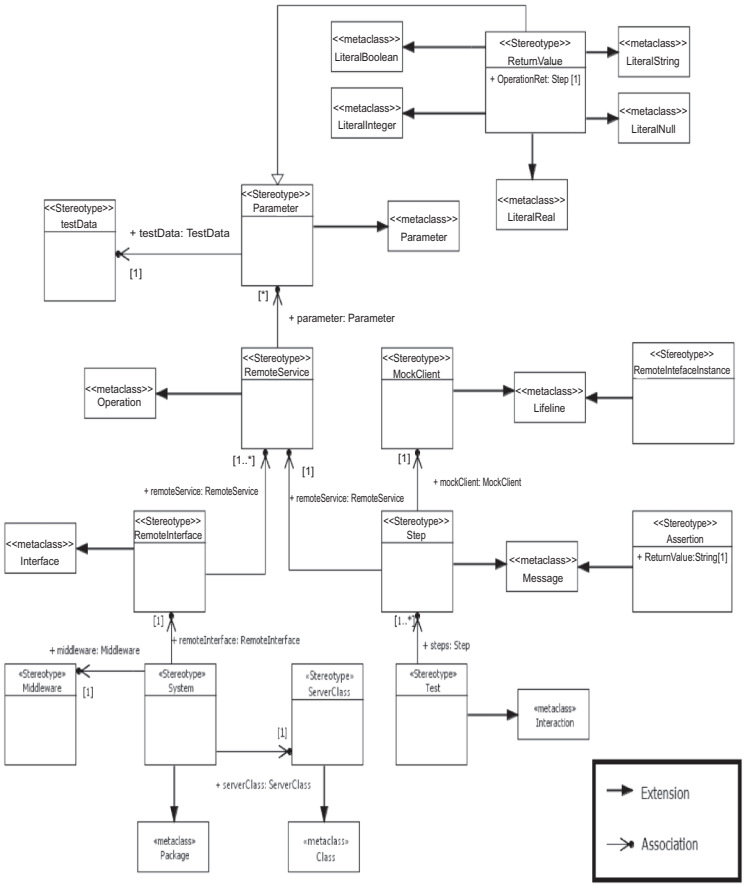

Fig. 4. Perfil UML PRUDIMA para pruebas a SD.

En la Tabla 1 Descripción estereotipos, contiene los requisitos para los pasos del segundo al quinto, haciendo referencia a los identificadores de los que definen los metamodelos SD y PSD respectivamente.

Estereotipo	Semántica
Remote Interface	Es una entidad que define las funcionalidades del componente servidor, por medio de un conjunto de servicios remotos, los cuales pueden ser invocados por un cliente. Metaclase que extiende: *Interface*
Remote Service	Esta entidad declara un servicio remoto, con sus parámetr os de entrada y salida. El cual realizará una acción en el componente servidor y si es requerido, retornará el resultado al cliente que lo haya invocado. Metaclase que extiende: *Operation*
System	Esta entidad caracteriza el SD y define su contexto tecnológico sobre el cual se soporta, este contexto puede ser RMI ó CORBA. Además de asociar por composición una interface remota, la cual es el punto de entrada para que los clientes consuman los servicios del sistema. Metaclase que extiende: *Package*
Middleware	Esta entidad permite especificar a través de los atributos *middlewareType* a qué tipo de plataforma de tecnología estará enfocado el SD y la versión de la plataforma tecnológica. Metaclase que extiende: *Ninguna*
Server Class	Esta entidad representa la clase servidor remoto, la cual tiene como funcionalidad desplegar los servicios del SD. Metaclase que extiende: *Class*
Parameter	Extiende un elemento *Parameter* de UML para que se comporte como una entidad *Parameter* del metamodelo SD. Metaclase que extiende: *Parameter*
TestData	Estereotipo que se relaciona con la entidad *Middleware* del metamodelo SD. Metaclase que extiende: *Ninguna*
Return Value	Esta entidad representa un valor de retorno qu e será devuelto por un paso de una prueba en ejecución. Metaclases que extiende: *LiteralBoolean, LiteralInteger, LiteralString, LiteralReal, LiteralNull*
Step	Esta entidad especifica un paso, el cual será el encargado de hacer el llamado a una operación de l SUT utilizando para ello el atributo asociado con la entidad RemoteService. Metaclase que extiende: *Message*
Mock Client	Esta entidad define a un actor cliente el cual es el encargado de ejecutar pasos dentro de una prueba del sistema. Metaclase que extiende: *Lifeline*
Test	Esta entidad permite representar pruebas unitarias o funcionales dentro de las pruebas al sistema, por medio de una lista de pasos ordenados. Metaclase que extiende: *Interaction*
Assertion	Esta entidad define una condición que se esper a se cumpla o no, y cuya comparación se define por el valor retornado de los servicios remotos del SD. Metaclase que extiende: *Message*
Remote Interface Instance	Esta entidad permite representar una instancia en ejecución de una interfaz remota que puede recibir solicitudes a través de sus servicios remotos. Metaclase que extiende: *Lifeline*

Tabla 1. Descripción estereotipos.

Cabe resaltar que se utilizó la herramienta *Papyrus* dentro del entorno *Eclipse* como herramienta de modelado.

3.4. Proceso para Pruebas a SD

De acuerdo con el análisis de la literatura, se encontraron varios trabajos donde se presentan diferentes procesos –[2], [9], [12], [26], [32] para aplicar MBT y la descripción de sus pasos en contextos determinados. A partir de ellos, se realizó un análisis de los pasos que mejor se ajustaban para lograr mejorar la calidad a las pruebas de SD. Con lo cual se definió el proceso práctico de la aplicación de MBT para SD (P-MBT-SD), ver la Fig. 5 Proceso de pruebas, el cual define los pasos que se deben llevar con el orden específico que el flujo indica. Este proceso consta tanto de pasos manuales, los cuales son efectuadas por el *Tester*, como también por tareas automáticas realizadas por el sistema *software* que apoya el

proceso, que es un prototipo de una herramienta CASE para MBT construida por los autores de este trabajo, que se denominó **PRUDIMA**.

Se hallaron similitudes en los procesos estudiados y se tomó como base el proceso más genérico, que consta de los siguientes pasos: *Modelar, Generar, Concretizar, Ejecutar y Analizar,* los cuales están presentes en todos los trabajos.

Algunos de los pasos anteriores se llevan a cabo de varias maneras utilizando diferentes enfoques, dependiendo del contexto donde se esté aplicando MBT.

P-MBT-SD es un proceso que puede ser aplicado de una manera tanto iterativa e incremental como aplicarlo en la última fase de pruebas, como se maneja en algunas metodologías de desarrollo[3]. Sin embargo, es recomendable que sea aplicado desde un principio, ya que permite aprovechar los beneficios del enfoque MBT, como encontrar errores en las etapas tempranas del desarrollo del SUT, corrección de requerimientos pobremente definidos, entre otros[9], [26].

3.4.1. Descripción de los pasos de P- MBT- SD

A continuación se realiza una descripción detallada de cada uno de los pasos que compone el P-MBT-SD.

1) Modelar el SUT: el primer paso corresponde a diseñar el SD que será probado (SUT). Esto lo realizará el *diseñador*[4] del sistema, quien utilizará los requerimientos del sistema previamente definidos para modelar tanto la parte estática como la parte dinámica del SD. En este caso, PRUDIMA soporta el modelamiento con el estándar UML[14].

- ***Diagrama de clases*:** es un artefacto el cual permite tener una vista estática del sistema. Para este proceso, dentro del diagrama se definen los contratos necesarios a través de interfaces y clases relacionadas que permiten conocer los servicios remotos del SD que van a ser el objeto de pruebas.

- ***Diagramas de interacción*:** este artefacto permite tener una vista dinámica del sistema, es decir, se pueden observar comportamientos por medio del paso de mensajes entre objetos del sistema. Dentro de la especificación de UML [14] existen dos tipos de estos diagramas: *diagramas de colaboración* y *secuencia*. Se decide utilizar los diagramas de secuencia porque enfatizan en el ordenamiento temporal del paso de mensajes entre los objetos.

[3] **El modelo de desarrollo en cascada**: es un ejemplo de modelo, en el cual se tiene una fase de pruebas que se realiza al final del desarrollo del sistema.
[4] **Diseñador**: persona o grupo de personas encargadas de modelar el sistema

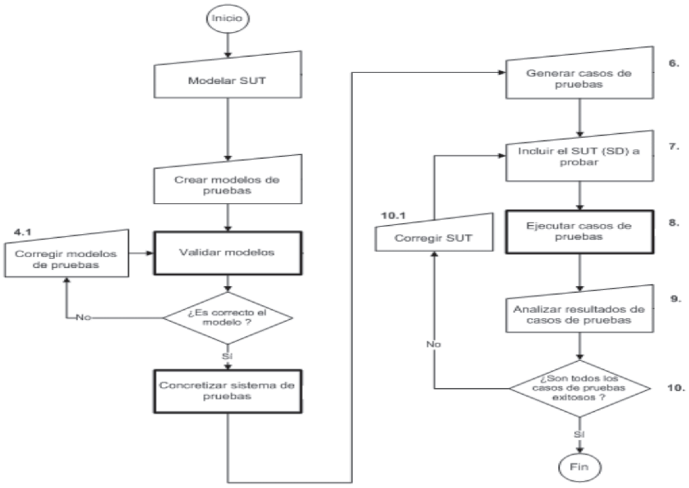

Fig. 5. Proceso de pruebas.

2) Modelar Casos de Prueba: en este paso se requiere de los entregables realizados en el paso previo: diagrama de clases y diagramas de interacción.

A partir de estos insumos, se aplica el *perfil UML de PRUDIMA* a los diagramas de clases e interacción, con lo cual, se realiza el proceso de modelamiento de casos de pruebas, convirtiendo en escenarios de pruebas abstractos los diagramas de secuencia. Este paso sigue un enfoque de especificación explicita de casos de pruebas [2], debido a la ventaja de tener completo control sobre los casos de pruebas que se están generando.

3) Validar modelos: es el paso a seguir una vez estén listos los modelos de pruebas, es analizar los modelos de pruebas buscando posibles errores de aplicación del perfil UML de PRUDIMA, los cuales no permitan transformarlos a un formato ejecutable. Este paso es automáticamente soportado por PRUDIMA. Lo cual, ayuda a reducir costos, ya que permite encontrar errores de modelamiento en etapas tempranas. Para esto PRUDIMA utiliza unas reglas definidas así:

- Se deben encontrar aplicados los estereotipos del perfil UML de PRUDIMA a los elementos necesarios para concretizar los casos de pruebas que se encuentren en los diagramas de clases e interacción.
- Las aserciones dentro del modelo deben contener valores válidos para el sistema.

4) ¿Es correcto el modelo?: en este paso el modelador verifica el resultado de la validación anterior; si todo es correcto sigue al paso 5, si no se dirige al paso 4.1.

4.1) Corregir modelos de pruebas: el modelador corrige los defectos encontrados en el modelo. Seguido vuelve a realizar al paso 3 que valida el modelo.

5) Concretizar sistema de pruebas: este paso hace referencia a la concretización de las pruebas abstractas siguiendo un enfoque de *transformación* [2], es decir, se transforma a código fuente (M2T) el conjunto de pruebas abstractas que se construyeron a partir de los modelos abstractos del SUT en el primer paso.

Aquí se genera el código ejecutable de las pruebas. PRUDIMA genera también toda la arquitectura necesaria para tener un sistema de ejecución de pruebas completo.

Este sistema incluye el soporte para inclusión, despliegue y estimulación del SUT, métodos del oráculo de pruebas, motor de ejecución de pruebas y reporte de resultados de pruebas.

6) Generar casos de pruebas: luego de tener los escenarios de pruebas se requiere una tarea manual por parte del *tester,* el cual crea los casos de prueba, es decir, crea una instancia de un escenario de prueba; para luego insertar la información requerida por cada uno de los casos de prueba. Dicha información es almacenada en un repositorio de datos de pruebas, para luego ser consumida en ejecución. Dado que la generación automática de datos no es el foco de este trabajo, aquí se genera manualmente, en concordancia con el criterio de selección de casos de prueba escogido en el segundo paso. Sin embargo, se realizó la exploración de trabajos relacionados con la generación automática y se encontraron técnicas como: particiones de equivalencia, análisis de valores, generación de datos aleatorios, modelos de datos, entre otros [2][33][34].

7) Incluir el SD a probar: el *tester* deberá indicarle al sistema previamente generado cuál va a ser el SD a probar (SUT), estableciendo los canales de comunicación adecuados para una correcta estimulación del SUT; estos canales se derivan a partir de los modelos generados en el primer paso. Para realizar este paso es necesario apoyarse de la GUI del sistema PRUDIMA.

8) Ejecutar casos de pruebas: el *tester* procede a ejecutar a través de PRUDIMA el conjunto de pruebas que se encuentran en el sistema de pruebas generado que a su vez irá generará los respectivos reportes de

los veredictos hechos por los oráculos. El enfoque de ejecución utilizado es el *off-line,* ya que para ejecutar las pruebas estas deben haber sido estrictamente generadas previamente, al contrario del enfoque de ejecución *on-line* las cuales se van ejecutando a medida que se modelan [2]. Existen algunas ventajas que corresponden al enfoque escogido, como son el administrar y ejecutar las pruebas con una herramienta *software*, el poder repetir un conjunto de pruebas las veces que sean necesarias incluso en distintas máquinas, lo cual conlleva a hacer factibles las pruebas de regresión[9].

9) Analizar resultados de los casos de pruebas: este paso, el cual es común en todos los trabajos incluso en pruebas con enfoques diferentes como manuales, *scripting*, automáticas, entre otros [2][35]. El *tester* debe verificar el resultado de las pruebas, para reportar las incidencias y detallando los errores que causaron el fallo. Los errores encontrados pueden ser en el SUT, en el modelo de pruebas o en los requerimientos definidos para el sistema. Brindando retroalimentación al área correspondiente, dependiendo del tipo de errores encontrados, las áreas pueden ser diseño, desarrollo o pruebas para que se realicen las correcciones adecuadas.

10) ¿Son todos los casos de pruebas exitosos?: aquí el *tester* verifica si hay algún resultado fallido, es decir, sí su veredicto es *fail.* Si es así, se debe seguir al paso 10.1, si no, el proceso termina satisfactoriamente.

10.1) Corregir SUT: se debe corregir el SUT por parte de los desarrolladores del mismo, después se vuelve al paso 7 y se itera hasta que todos los resultados de los casos de pruebas sean satisfactorios.

3.5. Prototipo PRUDIMA

Para el anterior proceso se hizo necesaria la construcción de un prototipo de herramienta CASE que soportara cada uno de los pasos que lo componen. A continuación se describe el proceso de desarrollo.

3.5.1. Desarrollo del prototipo PRUDIMA

El proceso de desarrollo se llevó a cabo utilizando la metodología de proceso unificado ágil - AUP y se realizaron dos iteraciones de la misma para finalizar el prototipo. AUP consta de las siguientes fases iterativas e incrementales: *inicio*, *elaboración*, *construcción* y *transición.*

Primera iteración: en la fase de elaboración se identificaron los requerimientos para el sistema. Se debe poder diseñar pruebas a través de diagramas de clases y secuencia utilizando el lenguaje UML y el perfil UML PRUDIMA, los modelos se deben transformar en código

fuente *Java* para su posterior ejecución, debe existir una GUI principal que permita seguir el proceso definido para MBT en este trabajo, ver la Fig. 6 GUI principal de PRUDIMA ,y finalmente debe soportarse sobre agentes *software*, por lo tanto un esquema general de la aplicación se puede apreciar en laFig. 7 Entorno de PRUDIMA.

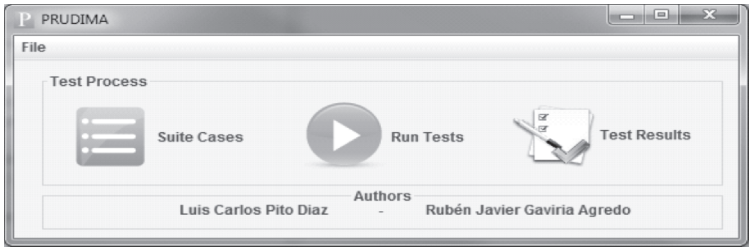

Fig. 6. GUI principal de PRUDIMA.

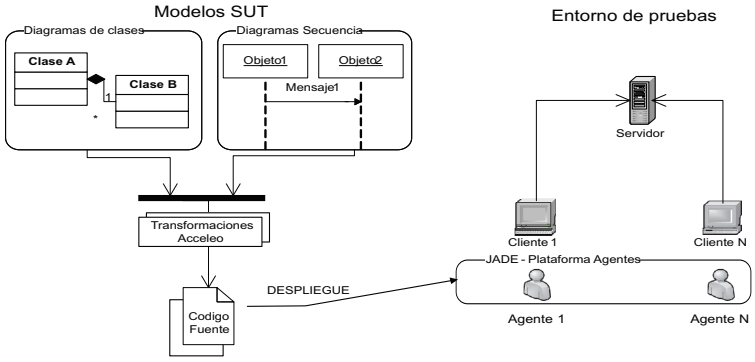

Fig. 7. Entorno de PRUDIMA.

En la fase de elaboración se plantearon soluciones para cumplir los requerimientos. Para el diseño de las pruebas y aplicación del perfil UML PRUDIMA se debía utilizar el modelador *Papyrus*. *Acceleo* sería el lenguaje encargado de realizar las transformaciones M2T, se definió la arquitectura para el GUI que soportara el proceso MBT y su integración con los agentes *software* bajo la plataforma JADE. La fase de construcción se realizó bajo el IDE de *Eclipse*, donde se integraron todos los componentes a través de código Java. En la última fase de transición se realizaron pruebas Beta al prototipo y se obtuvo retroalimentación por parte de los *testers.*

Segunda iteración: aquí se llevaron a cabo las mismas fases anteriores pero refinando los diseños y corrigiendo los defectos encontrados en la primera iteración. Finalmente, el proceso de desarrollo terminó y se

liberó una versión para el entorno de producción, que para este trabajo fue el despliegue para el estudio de caso.

3.6. Estudio de Caso

En esta sección se expone un estudio empírico, realizado con la finalidad de validar los elementos construidos a lo largo de la investigación, a saber, el proceso definido de aplicación del enfoque MBT a pruebas de SD en un entorno académico, el uso del perfil UML que agrupa los metamodelos tanto para SD como para PSD y el prototipo PRUDIMA.

3.6.1. Metodología

Para llevar a cabo este estudio se utilizaron los lineamientos definidos en [36], ya que es una metodología de investigación adecuada en el campo de ingeniería de *software*, la cual permite observar y medir las unidades de análisis en un contexto real, manteniendo la integridad y las características significativas de los eventos. A continuación, se describen los pasos llevados a cabo.

***Diseño*:** el estudio de caso fue definido para validar el proceso de MBT definido en este trabajo, el cual utiliza un perfil UML que agrupa los metamodelos de SD y PSD, a través del prototipo PRUDIMA, que ayuda a guiar el proceso paso a paso. Para esto, se escogió el tipo de estudio de caso, se definieron métricas e indicadores, que permitieran analizar posteriormente los datos obtenidos durante la ejecución del estudio y se determinaron las formas de recolección de los datos.

***Preparación*:** se definieron los instrumentos necesarios tanto para el desarrollo del caso como para la recolección de los datos durante su ejecución, que fueron:

- Guías para realizar los procesos de pruebas manuales y MBT, este último con la ayuda del prototipo PRUDIMA
- Reportes entregados por el prototipo PRUDIMA
- El formato de encuesta

***Ejecución y recolección de los datos*:** la ejecución fue llevada a cabo en dos sesiones, en las cuales participaron diferentes grupos de personas, que fueron capacitadas para realizar el proceso de pruebas MBT y en el uso del prototipo PRUDIMA. En dichas sesiones se entregaron las guías para el desarrollo de la práctica. En la primera sesión debían llevar a cabo el proceso completo de MBT y en la segunda sesión únicamente se dedicaron a probar un SD utilizando dos enfoques de pruebas, el manual

y MBT. Los datos recolectados fueron a través de los reportes generados por el prototipo, como también por plantillas en Excel que las personas debían llenar. Todo lo anterior bajo el monitoreo por parte de los miembros del equipo de investigación

.

***Análisis*:** para este paso se realizó un análisis cuantitativo y cualitativo de los datos recolectados durante la ejecución del caso.

***Reporte*:** se reportan los hallazgos encontrados con los datos obtenidos en el paso previo.

3.6.2. Pregunta de Investigación

El prototipo PRUDIMA es una herramienta CASE que soporta el enfoque MBT definido en este trabajo. Esta permite definir modelos de pruebas utilizando el lenguaje de modelado UML, específicamente haciendo uso de los diagramas de clases y secuencia, a los cuales, se les aplica un perfil UML definido para las pruebas a SD para luego transformar automáticamente dichos modelos en un formato ejecutable permitiendo desplegar las pruebas y posteriormente obtener sus resultados.

En busca de mitigar los problemas de calidad que conlleva el uso del proceso de pruebas manuales, por ser propenso a errores y por el alto consumo de recursos tanto en tiempo como en recurso humano, se plantea la siguiente pregunta de investigación:

¿Se puede mejorar el proceso de pruebas aplicado a las prácticas de LSD siguiendo un enfoque MBT reduciendo el consumo del recurso tiempo?

3.6.3. Objetivo del Estudio de Caso

El objetivo de este caso de estudio es validar la utilización de los metamodelos de SD y PSD, el perfil UML que los agrupa y el proceso de pruebas con enfoque MBT, a través del uso del prototipo PRUDIMA, para modelar, ejecutar y analizar pruebas a un SD, donde todos los elementos mencionados anteriormente fueron desarrollados por los autores del presente trabajo de investigación. Para esto, se realizarán dos sesiones prácticas, en las cuales se evaluará el proceso de MBT y se compararán los resultados obtenidos de este enfoque con los del trabajo de López[37], que tiene los reportes de los tiempos de pruebas manuales aplicados a la evaluación de prácticas en LSD. Siendo estas prácticas ejecutadas dentro del contexto académico de LSD de la Universidad del Cauca.

3.6.4. Selección del Estudio de Caso

Ya que la metodología requiere tener interpretaciones integrales de un caso y sus relaciones con otras variables, en este trabajo se analiza un solo caso con el propósito de obtener un análisis bien detallado. Por lo tanto, se selecciona un estudio de caso del tipo holístico, siendo crítico y revelador.

La unidad de análisis para este caso es la aplicación del proceso MBT para SD en un contexto académico [38]. El proceso de pruebas a SD es llevado a cabo por ingenieros con conocimientos en el área, por lo tanto, son ellos los que pueden evaluar el proceso para MBT propuesto y su selección corresponde a los criterios de disponibilidad [39] de ingenieros o estudiantes de ingeniería que tuviesen conocimientos en pruebas manuales a SD, quienes serán las principales fuentes de información.

3.6.5. Contexto del Estudio de Caso

La organización: El LSD es una asignatura dictada en séptimo semestre de la carrera de Ingeniería de Sistemas, la cual aplica los conceptos de la asignatura de Sistemas Distribuidos, por medio de desarrollo de prácticas de ejercicios de desarrollo, de aplicaciones distribuidas, en los lenguajes y herramientas más conocida como: *RPC, RMI, CORBA y DCOM*.

El caso y sus sujetos de investigación: El presente estudio de caso es de tipo holístico[40], teniendo en cuenta los modelos de pruebas y la fuente primaria de información, en este caso dos grupos de estudiantes de Ingeniería de Sistemas, quienes han aprobado satisfactoriamente la asignatura de LSD, en semestres anteriores. Cabe resaltar que estos estudiantes poseen la experiencia en el desarrollo y aplicación de pruebas de las aplicaciones que se realizan en LSD. Este estudio de caso, tiene un propósito positivista [41], ya que busca probar que existe una mejora en la aplicación de un enfoque de pruebas por medio de MBT, en comparación con el enfoque manual que se viene realizando. Con el objetivo de aplicar el proceso definido en el presente trabajo se realizan dos sesiones de trabajo, las cuales serán dirigidas por medio de una guía, en las que realizará una especificación el sistema a probar y los pasos a seguir para practicar el procedimiento MBT y el procedimiento de pruebas manuales.

3.6.6. Indicadores y Métricas

Para evaluar objetivamente este estudio de caso, se definió un conjunto de métricas e indicadores que permitieran responder la pregunta de

investigación. A continuación se detalla cada uno de los indicadores identificados y su métricas asociadas a este estudio de caso, basados en [42]:

***Eficacia*:** se define como el análisis del cumplimento de los requisitos definidos.

$$Ef = \frac{Ce}{Cx} \tag{1}$$

Donde *Ef* es la eficiencia, *Ce* es el número total de casos de pruebas exitosos, *Cx* es el número total de casos de prueba ejecutados.

***Cobertura de pruebas definidas*:** se define como el grado de cubrimiento de las pruebas definidas.

$$Cx = \frac{Px}{Pd} \tag{2}$$

Donde Cx es la cobertura de las pruebas ejecutadas, *Px* es el número de pruebas ejecutadas, *Pd* es el número de pruebas definidas.

***Eficiencia*:** se define como el grado en que los objetivos se cumplen con el menor costo posible.

$$E = \frac{Px}{t} \tag{3}$$

Donde E es la eficiencia, *Px* es el número de pruebas ejecutadas, *t* es el tiempo total de las pruebas.

***Facilidad de uso*:** la usabilidad se ha definido como el grado en que un producto puede ser usado por determinados usuarios para lograr sus propósitos con eficacia, eficiencia y satisfacción, en un contexto de uso específico, siendo en este caso medido el uso del proceso MBT, definido en este trabajo para las pruebas a SD. La fórmula que se definió para hacer el cálculo de la usabilidad percibida es:

$$U = \frac{\sum_{i=1}^{n} \frac{RE_i}{RD_i}}{n} \tag{4}$$

Donde *U* es la usabilidad, *REi* es el resultado evaluado para la pregunta *i*, el cual puede tomar valores de uno a cinco, *RDi* es el resultado deseado en la encuesta a la pregunta *i*, cuyo resultado esperado es 5, *n* es el número de preguntas evaluadas en la encuesta.

3.6.7. Ejecución del Estudio de Caso

Se realizaron dos sesiones que se describirán a continuación:

Primera sesión: se realizó a los participantes una introducción del enfoque MBT por medio de una presentación, como también la explicación de la práctica a realizar. Luego, se procedió a realizar una capacitación sobre la utilización del entorno del prototipo PRUDIMA implementado en este trabajo. Una vez finalizada la capacitación se realizaron los pasos indicados en la guía para esta sesión, donde se establece que primero se deben realizar el modelado del SUT y de las pruebas a ejecutar, teniendo en cuenta los requerimientos especificados dentro de la guía. En seguida, se deben realizar las transformaciones de modelo a texto para la generación de la aplicación que ejecutará la prueba y posteriormente, se solicitará realizar la carga de los datos a través del prototipo PRUDIMA. Finalmente deben realizar la ejecución de las pruebas para seis SUT.

Segunda sesión: en la primera parte de esta sesión se realizó una introducción de los enfoques MBT y manual, además de mostrar la guía a realizar. Posteriormente se realiza una corta capacitación del prototipo PRUDIMA implementado en este trabajo y del enfoque de pruebas manuales. Luego, se procederá a realizar una distribución al azar de los estudiantes, dentro de grupos de trabajo. El participante deberá aplicar el trabajo en el orden designado para su grupo. Por ejemplo: un estudiante que pertenezca al grupo 3, en la primera hora deberá practicar el enfoque manual a los laboratorios terminados en B y para la segunda hora deberá aplicar el enfoque automático para los laboratorios terminados en A.

Dentro de la guía entregada a los estudiantes se encuentra cómo practicar los enfoques a los laboratorios designados.

Para realizar el enfoque MBT, el estudiante cuenta con el modelo y las pruebas previamente definidas. Con lo cual, solo basta con realizar la transformación de modelo a texto, para obtener la herramienta encargada de probar. Luego, debe cargar los datos, ejecutar el sistema para que realice las pruebas automáticas. Esperar para obtener los resultados, los cuales debe registrar dentro de un archivo de reporte final.

Para realizar el enfoque manual, el estudiante debe ejecutar cada uno de los laboratorios a probar, luego ingresar los datos entregados, esperar y observar las respuestas de los SUT. Finalmente, anotar los resultados dentro del archivo de reporte final.

3.6.8. Resultados

Resultados cuantitativos

A continuación se presentan los registros de datos tomados durante la aplicación del estudio de caso. Es importante tener en cuenta que las sesiones contaban con una restricción en tiempo de aplicación, de tres horas para la primera sesión y de dos horas para la segunda sesión.

Eficacia
Para el cálculo de este indicador, se utilizan los valores promedio:

- *Eficacia en MBT*: el valor obtenido es del 94 por ciento de eficacia.

$$Ef = \frac{Ce}{Cx} = \frac{18.8}{20} = 0.94 * 100 = 94$$

- *Eficacia en pruebas manuales*: el valor obtenido de eficacia fue del 83,5 por ciento.

$$Ef = \frac{Ce}{Cx} = \frac{10,4}{12,6} = 0.835*100 = 83.5$$

Cobertura
Al igual que para el indicador de eficacia, fueron utilizados los datos de la segunda sesión.

- *Cobertura en MBT*: el valor obtenido fue del 100 por ciento.

$$Ef = \frac{Px}{Pd} = \frac{20}{20} = 1 * 100 = 100$$

- *Cobertura en pruebas manuales*: el valor obtenido fue del 63 por ciento.

$$Ef = \frac{Px}{Pd} = \frac{12,6}{20} = 0.63 * 100 = 63$$

Eficiencia
Para los resultados de este indicador se debe tener en cuenta que sus valores son dependientes de la complejidad de las pruebas a ejecutar. Debido a esto, se midió este indicador en dos conjuntos de casos de pruebas diferentes, los cuales se tomaron a partir de las dos sesiones ejecutadas.

***Primera sesión*:** para los valores de eficiencia en las pruebas manuales, se utilizaron los datos del trabajo de López [37], donde se realiza una medición del tiempo necesario para la ejecución de 207 pruebas.

- *Eficiencia MBT*: aquí se obtuvo un valor de 2,37 pruebas ejecutadas por minuto

$$E = \frac{Px}{t} = \frac{207}{86.032} = 2.37$$

- *Eficiencia pruebas manuales*: se obtuvo un valor de 1,83 pruebas ejecutadas por minuto.

$$E = \frac{Px}{t} = \frac{207}{111} = 1.83$$

***Segunda sesión*:** aquí, el conjunto de pruebas a ejecutar fue de 20, sin embargo la cantidad de pasos requeridos en las pruebas fue mayor, por lo cual el valor del indicador es menor respecto a los resultados de la primera sesión.

- *Eficiencia MBT*: se obtuvo un valor de 0,54 pruebas ejecutadas por minuto.

$$E = \frac{Px}{t} = \frac{20}{45} = 0.54$$

- *Eficiencia pruebas manuales*: se obtuvo un valor promedio de 0,28 pruebas por minuto.

$$E = \frac{Px}{t} = \frac{12.6}{45} = 0.28$$

Facilidad de uso

A partir de la encuesta diligenciada por los participantes de la primera sesión, se analizaron los datos recolectados, con el fin obtener un valor cuantitativo de la facilidad de uso del proceso de MBT, descrito en el presente trabajo. Teniendo en cuenta los resultados obtenidos de los participantes en la primera sesión se puede observar que se evaluó sobresaliente nivel de satisfacción; en la utilización del enfoque MBT para las pruebas a un SD, por medio de una herramienta CASE.

Resultados de la encuesta a nivel general

De acuerdo con los resultados obtenidos por la encuesta realizada a los participantes de la primera sesión, consideran que es *Excelente* de manera unánime que el perfil UML de PRUDIMA junto con UML, permite describir de manera adecuada las pruebas a SD de forma adecuada. Por otro lado se tiene una percepción *Sobresaliente* en la sencillez de la tarea de modelar las pruebas para SD, en la facilidad para

el usuario en realizar las transformaciones de modelo a código y de la forma ágil de probar múltiples implementaciones de un mismo sistema.

Análisis de los Resultados

A continuación se presentan el análisis de comparación por cada indicador entre los enfoques de pruebas MBT y manuales. Como se puede inferir a partir de los resultados obtenidos, el enfoque MBT es más eficiente que el manual, dado que el 94 por ciento de los casos de pruebas ejecutados en MBT fueron exitosos, contrastándolo con el 84 por ciento del enfoque manual. Sin embargo, se puede apreciar cómo MBT tampoco cumple el cien por ciento de eficacia, ya que existe una dependencia del factor humano, que en este trabajo se ve reflejado en la parte de modelamiento de las pruebas, carga de los datos e inclusión del SUT a probar. MBT cumple con el cien por ciento de la cobertura de pruebas definidas, es decir se ejecutaron todas las pruebas que se definieron inicialmente, a partir de los requerimientos, dentro de un periodo de tiempo limitado. Por otra parte, el enfoque manual alcanzó un sesenta por ciento de cobertura dentro del mismo periodo de tiempo. Sin embargo, si el tiempo para realizar la fase de pruebas hubiese sido mayor, es posible que ambos enfoques alcanzaran el mismo nivel de cobertura, pero con MBT se realizaría en menor tiempo.

Eficiencia, es el indicador que permite saber la cantidad de pruebas realizadas en un intervalo de tiempo definido, de la fase de pruebas en ambos enfoques. Se presenta una mayor eficiencia en MBT con un valor del 0,57 pruebas por minuto en comparación con los 0,29 pruebas por minuto del enfoque manual.

Finalmente, se responde la pregunta de investigación del estudio de caso, concluyendo en base a los resultados, que hay una mejora en la calidad de las pruebas a los SD al usar el enfoque MBT para el proceso de pruebas a SD, con la respectiva reducción del recurso tiempo.

4. Conclusiones

La industria del *software* que utiliza MDE, genera modelos que pueden ser reutilizados en diferentes fases del proceso de ingeniería de *software*. Es en la fase de implementación, donde comúnmente son usadas transformaciones a los modelos, generando automáticamente código fuente. En este trabajo se reutilizaron modelos de UML para la fase de pruebas, siguiendo el enfoque MBT. Donde se adaptaron por medio de un perfil UML, para realizar pruebas a sistemas SD en construcción, optimizando el uso de recursos de tiempo y esfuerzo. Evidenciando así la potencialidad de los enfoques dirigidos por modelos.

Durante el desarrollo de este trabajo de investigación, se aprendieron nuevos conceptos, paradigmas y herramientas asociados a la fase de pruebas, ampliando nuestra visión de las tendencias del mercado del *software*. Durante la carrera de Ingeniería de Sistemas, no se profundiza en dicha área, sin embargo, los fundamentos aprendidos de modelado de *software*, fueron fundamentales para entender estos nuevos conceptos.

Los metamodelos diseñados para SD y sus pruebas, permiten modelar adecuadamente pruebas a SD, basados en el paradigma de comunicación cliente/servidor. No obstante, es a través de su integración dentro de un perfil UML, que son utilizados de manera práctica, por medio de la extensión semántica de algunos elementos de los modelos que componen el lenguaje UML, reutilizando así, algunos de los artefactos generados por metodologías de desarrollo de *software*, como son los modelos estáticos y dinámicos del sistema a construir.

En este trabajo se definió un proceso que permite la generación de una plataforma de pruebas para SD soportada por agentes *software*, cuyo código es generado por un conjunto de transformaciones M2T, utilizando como insumo modelos de UML, más específicamente, utilizando los modelos definidos en diagramas de clases y de secuencia. A los elementos que componen estos diagramas, se les aplican los estereotipos del perfil UML de PRUDIMA, permitiendo extender su semántica hacia modelos de pruebas. De estos modelos, se pueden generar manualmente casos de pruebas, que finalmente son ejecutados contra el SUT, obteniendo un análisis y reporte de los mismos, evaluando así, la conformidad de los requerimientos definidos para el SUT.

Los resultados obtenidos en el estudio de caso realizado, en el entorno académico de la Universidad del Cauca, muestran que la utilización del enfoque MBT, a través de una herramienta que lo soporte, permite aumentar los niveles de calidad del proceso de pruebas a SD, además de reducir los tiempos requeridos por parte de los ingenieros encargados de evaluar las prácticas de laboratorios de SD.

En MBT se presentaron errores en las pruebas, principalmente por el factor humano, sin embargo el porcentaje es menor en comparación con el enfoque de pruebas manuales, debido a que en MBT se mitigan por medio de validaciones automáticas en dichas actividades manuales y el uso de código pre-construido para la ejecución de la pruebas.

Se evidencia que para seguir un enfoque MBT, en el cual utiliza modelos soportados en gráficos, se debe utilizar una herramienta que tenga un buen nivel de usabilidad, para que el tiempo empleado en las

fases de diseño de los modelos no contrarreste los beneficios obtenidos por la automatización de los demás pasos dentro del proceso de pruebas.

Se muestra cómo MBT podría ser portado a diferentes tipos de dominios *software*. En este trabajo de investigación se utilizó, para realizar pruebas en el ámbito de los SD enfocados en el paradigma cliente servidor.

Referencias

[1] J. S. Osmundson, J. B. Michael, M. J. Machniak, and M. A. Grossman, "Quality management metrics for software development," vol. 2033, pp. 1–14, 2002.

[2] M. Utting and B. Legeard, Practical model-based testing a tools approach, 1st ed. San Francisco: Elsevier Inc, 2006, p. 455.

[3] M. Lahami, F. Fakhfakh, M. Krichen, and M. Jmaiel, "Towards a TTCN-3 Test System for Runtime," pp. 71–86, 2012.

[4] C. Serrano, "Un Modelo Integral para un Profesional en Ingeniería," Univ. del Cauca, 2003.

[5] S. W. Ambler, "The Agile Unified Process (AUP)," 2006. [Online]. Available: www.ambysoft.com/unifiedprocess/agileUP.html.

[6] D. C. Schmidt, "Model-Driven Engineering," no. February, pp. 25–31, 2006.

[7] M. R. John Hutchinson Jon Whittle, "Model-Driven Engineering Practices in Industry." School of Computing and Communications, Lancaster University, UK, 2011.

[8] R. V Binder, "Model-based Testing User Survey: Results and Analysis." System Verification Associates Technical Report, p. 6, 2011.

[9] M. Utting, A. Pretschner, and B. Legeard, "A Taxonomy of Model-Based Testing." School of Computing and Mathematical Sciences, University of Waikato, Hamilton, New Zealand, 2006.

[10] A. C. Dias-Neto and G. H. Travassos, "Model-based testing approaches selection for software projects," Inf. Softw. Technol., vol. 51, no. 11, pp. 1487–1504, Nov. 2009.

[11] J. Tretmans, F. Prester, P. Helle, and W. Schamai, "Model-Based Testing 2010: Short Abstracts," Electron. Notes Theor. Comput. Sci., vol. 264, no. 3, pp. 85–99, Dec. 2010.

[12] K. Nylund, E. Östman, D. Truscan, and R. Teittinen, "Towards Rapid Creation of Test Adaptation in On-line Model-Based Testing," in 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops, 2011, pp. 174–179.

[13] L. C. Briand, Y. Labiche, and S. He, "Automating regression test selection based on UML designs," Inf. Softw. Technol., vol. 51, no. 1, pp. 16–30, Jan. 2009.

[14] Object Management Group, "Unified Modeling Language, Infrastructure," 2011.

[15] M. Hebach, "MDA with QVT," Present. Borl. Together, 2006.

[16] M. Wooldridge, An Introduction to MultiAgent Systems. Department of Computer Science, University of Liverpool, UK: John Wiley & Sons, Ltd, 2002.

[17] G. Coulouris, J. Dollimore, and T. Kindberg, Distributed Systems: Concepts and Design Edition 3, vol. 15. 2001, p. 4.

[18] J. D. Ahmad Saifan, "Model-Based Testing of Distributed Systems," p. 57, 2008.

[19] I. García-Magariño, R. Fuentes-Fernández, and J. J. Gómez-Sanz, "Guideline for the definition of EMF metamodels using an Entity-Relationship approach," Inf. Softw. Technol., vol. 51, no. 8, pp. 1217–1230, 2009.

[20] Eclipse Modeling Framework: A Developer's Guide. Addison-Wesley Professional, 2004, p. 680.

[21] D. Watkins and D. Thompson, "Adding semantics to interface definition languages," in Proceedings 1998 Australian Software Engineering Conference (Cat. No.98EX233), 1998, pp. 66–78.

[22] H. Owens II and J. H. Hill, "Generating Valid Interface Definition Language from Succinct Models," in 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, 2011, pp. 205–212.

[23] P. Baker, Z. R. Dai, J. Grabowski, Ø. Haugen, I. Schieferdecker, and C. Williams, Model-driven testing. Using the UML Testing Profile. 2008, p. 188.

[24] OMG, "UML Testing Profile." 2005.

[25] "An introduction to UML profiles," UML …, 2004.

[26] J. Boberg, "Early fault detection with model-based testing," Proc. 7th ACM SIGPLAN Work. …, 2008.

[27] W. Prenninger and A. Pretschner, "Abstractions for Model-Based Testing," Electron. Notes Theor. Comput. Sci., pp. 59–71, 2005.

[28] D. Streitferdt, F. Kantz, P. Nenninger, T. Ruschival, H. Kaul, T. Bauer, T. Hussain, and R. Eschbach, "Model-Based Testing of Highly Configurable Embedded Systems in the Automation Domain," Int. J. Embed. Real-Time Commun. Syst., vol. 2, no. 2, pp. 22–41, 2011.

[29] A. Bertolino, E. Marchetti, and H. Muccini, "Introducing a Reasonably Complete and Coherent Approach for Model-based Testing," Electron. Notes Theor. Comput. Sci., vol. 116, pp. 85–97, Jan. 2005.

[30] F. Abbors, A. Bäcklund, and D. Truscan, "MATERA - An Integrated Framework for Model-Based Testing," in 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems, 2010, pp. 321–328.

[31] J. Botella, F. Bouquet, J.-F. Capuron, F. Lebeau, B. Legeard, and F. Schadle, "Model-Based Testing of Cryptographic Components -- Lessons Learned from Experience," in 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013, pp. 192–201.

[32] S. R. Shahamiri, W. M. N. W. Kadir, and S. Z. Mohd-Hashim, "A Comparative Study on Automated Software Test Oracle Methods," in 2009 Fourth International Conference on Software Engineering Advances, 2009, pp. 140–145.

[33] F. Abbors, A. Bäcklund, and D. Truscan, "MATERA - An Integrated Framework for Model-Based Testing," in 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems, 2010, pp. 321–328.

[34] I. Rauf, M. Z. Z. Iqbal, and Z. I. Malik, "UML Based Modeling of Web Service Composition - A Survey," in 2008 Sixth International Conference on Software Engineering Research, Management and Applications, 2008, pp. 301–307.

[35] M. Cristia, "Introducción al Testing de Software," pp. 1–8, 2009.

[36] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empir. Softw. Eng., 2009.

[37] Gineth Andrea López Hoyos, "Modelado y propuesta de Mejora de Procesos de Desarrollo para un entorno académico," 2013.

[38] H. Maimbo, G. Pervan, and W. Perth, "Designing a case study protocol for application in IS research," Proc. Ninth Pacific Asia …, 2005.

[39] I. Benbasat, D. Goldstein, and M. Mead, "The case research strategy in studies of information systems," MIS Q., 1987.

[40] R. Yin, "Case study research: Design and methods," 2009.

[41] H. Klein and M. Myers, "A set of principles for conducting and evaluating interpretive field studies in information systems," MIS Q., 1999.

[42] P. B. Lakey, "A Measurement Framework for Assessing Model-Based Testing Quality," in 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, 2010, pp. 19–27.