En español

Propuesta de utilización del razonamiento basado en casos para la recuperación de procedimientos de prueba funcionales

Martha Dunia Delgado Dapena [1], Yucely López Trujillo[2] e Indira Chávez Valiente [3]

RESUMEN

En este trabajo se presenta una propuesta de estructura de almacenamiento y los mecanismos de recuperación utilizados para aplicar el razonamiento basado en casos (RBC) en la generación de procedimientos de prueba funcionales en proyectos de *software*. Esta propuesta parte de los requisitos funcionales del proyecto de *software* y en ella se enuncian los algoritmos propuestos para considerar la semejanza entre cada par de proyectos, así como los que permiten adaptar la solución encontrada en la base de casos a las características de los nuevos proyectos.

Palabras clave: razonamiento basado en casos, ingeniería de *software*, calidad de *software*, pruebas funcionales, inteligencia artificial.

Recibido: junio 5 de 2009
Aceptado: noviembre 15 de 2010

Introducción

El tema de la calidad del *software* es de gran actualidad. Sin embargo, aunque las empresas dedican grandes recursos a la adopción o definición de estándares de calidad los resultados alcanzados no cubren las expectativas, ya que la productividad es baja, la cantidad de recursos a consumir es casi impredecible y el resultado casi nunca tiene la calidad y profesionalidad requerida. Desde las primeras etapas en el ciclo de vida de un *software* se pueden comenzar a planificar las pruebas a realizar (Everett, 2007; Pressman, 2005) incluso antes de llegar a la etapa de codificación, con esto se evita insertar errores en la aplicación que pueden volverse muy difíciles de encontrar en fases posteriores.

A pesar de constituir una buena práctica, muy pocos se acogen a esta idea y comienzan a pensar en las pruebas sólo después de tener el código. La mayoría de los equipos de desarrollo de *software* cuentan con tiempo limitado para la creación de pruebas minuciosas y bien planificadas, lo cual trae consigo que se ejecuten fundamentalmente pruebas funcionales y sin una planificación previa. Bajo estas

[1] Ingeniera Informatica. M.Sc., en Informática Aplicada. Ph.D., en Ciencias Técnicas, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. Centro de estudios de ingeniería de sistemas (CEIS), Facultad de Ingenieria Informatica, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. marta@ceis.cujae.edu.cu

[2] Ingeniera Informatica. M.Sc., en Informática Aplicada. Ph.D., en Ciencias Técnicas, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. Centro de estudios de ingeniería de sistemas (CEIS), Facultad de Ingenieria Informatica, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. ylopez@ceis.cujae.edu.cu

[3] Ingeniera Informatica. M.Sc., en Informática Aplicada. Ph.D., en Ciencias Técnicas, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. Centro de estudios de ingeniería de sistemas (CEIS), Facultad de Ingenieria Informatica, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. ichavez@ceis.cujae.edu.cu

In English

Using case-based reasoning for generating functional test procedures

Martha Dunia Delgado Dapena [4], Yucely López Trujillo[5] and Indira Chávez Valiente [6]

ABSTRACT

This paper presents a proposal for storage structure and retrieval mechanisms used for implementing case-based reasoning (CBR) in generating functional test procedures in software projects. This proposal was based on software project t functional requirements and sets out the proposed algorithms for considering the similarity between each pair of projects as well as those leading to adapting the solution found in the case base.

Keywords: case based reasoning, software engineering, software quality, functional testing, artificial intelligence.

Received: june 5th 2009
Accepted: november 15th 2010

Introduction

The issue of software quality receives a lot of attention nowadays. Although companies devote significant resources to adopting or defining quality standards, the results have not met expectations since productivity has been low, the actual amount of resources consumed has been almost unpredictable and has almost never resulted in the required quality and professionalism. The tests to be performed (Everett, 2007) (Pressman, 2005) can be planned from the earliest stages of software life-cycle, even before reaching the codification stage. This prevents errors being inserted into an application, which can become quite difficult to find in later stages.

Despite being good practice, few are benefiting from this idea and only begin to think about the tests after having the code. Most software development teams have limited time for creating detailed and well-planned testing, consequently leading to the execution of mainly functional tests without prior planning. In these circumstances, one can think of working on accumulating experience in the

[4] Computer Engineering. M.Sc., in Applied Computing. Ph.D., in Technical Sciences, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. Centro de estudios de ingeniería de sistemas (CEIS), School of Computer Engineering, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. marta@ceis.cujae.edu.cu

[5] Computer Engineering. M.Sc., in Applied Computing. Ph.D., in Technical Sciences, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. Centro de estudios de ingeniería de sistemas (CEIS), School of Computer Engineering, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. ylopez@ceis.cujae.edu.cu

[6] Computer Engineering. M.Sc., in Applied Computing. Ph.D., in Technical Sciences, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. Centro de estudios de ingeniería de sistemas (CEIS), School of Computer Engineering, Insitituto Superior Politécnico "José Antonio Echevarria" (CUJAE), La Habana, Cuba. ichavez@ceis.cujae.edu.cu

En español

condiciones puede pensarse en trabajar sobre la acumulación, en la computadora, de la experiencia que se obtiene en cada una de las pruebas, de forma tal que se disponga de un banco de casos para apoyar la fase de pruebas en nuevos proyectos, y en particular la generación de casos y procedimientos de prueba partiendo de otros que se han utilizado antes y que han permitido detectar defectos en proyectos previos.

Diferentes metodologías en desarrollo de *software* incluyen las pruebas, y algunas hacen más énfasis en ellas, como es el caso del desarrollo guiado por pruebas —en inglés *Test-Driven Development*— (Augustine, 2005) y otras lo ven como una fase del desarrollo que debe garantizar la calidad del producto final (Jacobson, 2004), pero todas las propuestas coinciden en que es un tema de suma importancia y debe planificarse desde las etapas tempranas de desarrollo del *software*. Varios trabajos refieren la importancia de estimar el esfuerzo asociado a las pruebas para decidir su automatización o ejecución en cada caso (Singh y Misra, 2008), así como la automatización de pruebas en entornos específicos (Xie y Memon, 2007; Masood, Bhatti, Ghafoor y Mathur, 2009; (KO y Myers, 2010). Existe un conjunto de propuestas que abordan directamente el tema de la reducción de casos de prueba (Heimdahl, 2004; Mogyorodi, 2008; Polo, 2005; Black, 2004; éstas utilizan algoritmos donde es vital disponer del tiempo necesario para la etapa de pruebas, siendo esto a veces imposible.

Han surgido nuevos procesos y metodologías para diseñar pruebas (Myers, 2004; Guvenc, 2006; Meyer, 2006; Gutiérrez, 2007; Dias; 2008; Naslavsky, 2008), pero en general ellas no permiten la reutilización de casos o procedimientos generados con anterioridad.

La propuesta que se presenta en este artículo toma en cuenta solamente las pruebas a los requisitos del *software* (funcionales o de caja negra), cuyo objetivo fundamental es validar si el comportamiento observado de él cumple o no con las especificaciones. Este trabajo forma parte de un proyecto más ambicioso a cargo de un grupo de investigadores y profesores del Centro de Estudios de Ingeniería de Sistemas (CEIS) perteneciente a la Facultad de Informática del Instituto Superior Politécnico José Antonio Echeverría (Cujae), cuyo objetivo fundamental es hacer propuestas para la planificación y reutilización de pruebas funcionales.

El proyecto abarca tres fases: una primera, en la que se traduce la especificación de requisitos del *software*, escrito en un lenguaje no formal, a un listado de requisitos funcionales reducido a un verbo y un sustantivo que caracterizan la funcionalidad en cuestión. En la segunda fase se llevan a cabo varios procesos que generan un conjunto de casos y procedimientos de prueba para un proyecto de *software* partiendo de pruebas funcionales desarrolladas a proyectos anteriores. Por último, una tercera fase, en la que se reduce el conjunto de casos y procedimientos de prueba propuestos de acuerdo a determinados criterios de prioridad y que serán considerados como la primera aproximación al plan de pruebas funcionales.

Este reporte está enfocado en la segunda fase del proyecto, relacionada con los algoritmos y mecanismos que permiten la generación de procedimientos de prueba funcionales en un proyecto de *software*.

Para lograr una herramienta de apoyo a este tipo de decisión parece aconsejable utilizar técnicas de inteligencia artificial (IA). Si se considera que a los posibles expertos les resulta muy difícil establecer cadenas de reglas generalizables que permitan inferir los casos de prueba y procedimientos de prueba a utilizar en cualquier tipo de proyecto, y les es más fácil expresar su conocimiento en términos de

In English

computer which has been obtained from each test. Doing so, a bank of cases can be created to support new projects' testing phase and, in particular, generating test cases and procedures from others which have already been used and have led to identifying flaws in previous projects.

Software development methodologies include testing; some place more emphasis on this aspect, as in the case of test-driven development (Augustine, 2005). Other methodologies consider testing as a developmental stage that should ensure final product quality (Jacobson, 2004). Nonetheless, all proposals agree that it is an issue of paramount importance and should be planned during early stages of software development. Some studies have referred to the importance of estimating the effort associated with testing to determine automation or performance in each case (Singh and Misra 2008) and also test automation in specific environments (Xie and Memon 2007) (Masood Bhatti Ghafoor and Mathur 2009) (KO and Myers 2010). One set of proposals directly addresses the reduction of test cases (Heimdahl, 2004) (Mogyorodi, 2008) (Polo, 2005) (Black, 2004); they use algorithms in which it is vital to have sufficient time for the testing stage, this being sometimes impossible.

New processes and methodologies for designing tests are available (Myers, 2004) (Guvenc, 2006) (Meyer, 2006) (Gutiérrez, 2007) (Dias, 2008) (Naslavsky, 2008) but they do not usually allow reuse of cases or previously-generated procedures.

The proposal presented in this paper only takes into account testing software requirements (functional or black box) whose main objective is to validate whether observed software behavior complies with its specifications. This work forms part of a larger project by a group of researchers and teachers from the Systems Engineering Study Center (CEIS), forming part of the Computer Science Faculty at the José Antonio Echeverría Higher Polytechnic Institute (CUJAE). Its main objective was to make proposals for planning functional tests and reusing them.

The project consisted of three phases. The first consisted of translating software requirements written in non-formal language to a list of functional requirements reduced to a verb and noun characterising the functionality in question. Several processes were carried out in the second phase generating a set of test cases and procedures for a software project based on functional tests from previous projects. A third phase reduced the set of cases and test procedures proposed in accordance with priority criteria and will be considered as a first approach to functional test planning.

This report focus on the project's second phase related to algorithms and mechanisms generating functional test procedures in a software project.

Artificial intelligence (AI) techniques were used for achieving a tool for supporting this type of decision. Case-based reasoning (CBR)) (Althoff, 2001) was used, taking into consideration that potential experts find it very difficult to establish chains of generalized rules generating test cases and test procedures for use in any type of project, and it is easier to express knowledge in terms of cases.

En español

las pruebas de *software* se han utilizado varias técnicas de IA, de las que algunos ejemplos son el análisis de resultados utilizando una red neuronal y algoritmos de minería de datos presentados en Last (2006), la priorización de casos de prueba utilizando algoritmos genéticos y otras técnicas (Elbaum, 2004; Fraser, 2007)

Razonamiento basado en casos para reutilizar procedimientos de prueba en proyectos de *software*

Estructura de la base de casos

La información referente a los proyectos y los defectos detectados en las revisiones a éstos se almacenan en un repositorio presentado en Delgado (2006), que ha sido enriquecido con los conceptos de casos y procedimientos de prueba, así como los mecanismos de recuperación que permiten recuperar los casos de prueba (CP) y los procedimientos de prueba (PP), asociados con los proyectos almacenados en la base de casos y que son semejantes al nuevo proyecto en el que se trata de generar los CP. Es importante precisar que esta propuesta es para generación en etapas tempranas, donde la información de entrada es requisito del nuevo proyecto.

Ha sido necesario introducir la asociación entre el requisito del proyecto y una descripción de los diferentes escenarios a considerar en su prueba funcional (descripción de prueba funcional, DPF). Esta descripción contiene el orden de las acciones involucradas en el desarrollo de la funcionalidad y que incluyen todos los escenarios posibles que deben ser considerados en la prueba funcional del requisito. La estructura interna de esta entidad ha sido tomada de la definición realizada por Delgado (2006).

Adicionalmente, la DPF es relacionada con cada uno de los PP diseñados para ejecutar su prueba funcional y éstos a su vez están referenciados por cada uno de los CP en los que se utilizan.

La estructura definida para almacenar los PP se muestra en la tabla 2. Los PP son identificados en la base de proyectos por la combinación del par <verbo> <sustantivo> que definen su nombre.

El *sustantivo* que nombra el PP constituye la referencia a un *marco* o *frame* dentro de una jerarquía de *frame*s en la que se expresan las relaciones entre las entidades de un dominio particular y que son utilizadas en esta propuesta para la recuperación de PP entre proyectos similares. La estructura del *frame* que modela una entidad particular es la que se muestra en la tabla 1.

Tabla1. Estructura de una entidad

Nombre de la Entidad:	<SUSTANTIVO>	
Atributos de <SUSTANTIVO>		
Atributos	Tipo de Atributo	Posibles Valores
Reglas de validación de atributos		
Atributo	Regla de atributo	
Reglas de validación de entidad		
Atributos involucrados	Regla de entidad	

Los CP en su sección de e*ntradas* hacen además referencia al paso del PP en el que el probador del proyecto debe introducir los valores de los atributos y entidades correspondientes. El almacenamiento separado de PP y CP garantiza que el mismo PP pueda ser utilizado en varios CP. Como puede observarse, ambas estructuras están ligadas a la lógica funcional propia del requisito que se quiere probar, donde se combinan el orden de acciones a ejecutar

In English

CBR has had applications in software engineering (Shepperd, 2003) (Ganesan, 2000) (García, 2001). Several AI techniques have been used in software testing; some examples would be analysing results using a neural network and data-mining algorithms presented by Last (Last, 2006), prioritising test cases using genetic algorithms (Elbaum, 2004) (Fraser, 2007)

Case-based reasoning for reusing test procedures on software projects

Base case structure

Information regarding the projects and all defects detected were stored in a repository presented by Delgado (Delgado, 2006). This repository had been enriched with test cases, test procedures and recovery mechanisms. The latter allowed retrieval of test cases (TC) and test procedures (TP) associated with the projects stored in the case database which were similar to a new project for which one may want to generate a TC. It should be noted that this proposal consisted of generation during early stages where input information was required for a new project.

It has been necessary to associate project requirements and a description of different scenarios to be considered in the functional test (functional test description - FTD). The FTD contained the order of actions involved in developing functionality and included all possible scenarios to be considered in the requirement's FDT. This entity's internal structure has been taken from Delgado's definition (2006).

The FTD was linked to each TP designed for performing functional testing and these, in turn, were referenced by each TC where they were used.

The structure defined for storing the TP is shown in Table 2. TP were identified in the project database by combining the pair (<VERB>, <NOUN>) defining its name.

The NOUN identifying the TP referred to a frame within a hierarchy of frames expressing relationships between entities in a particular domain and were used in this proposal for recovery of TP amongst similar projects. The frame structure modelling a particular entity is shown in Table 1.

Table1. Entity structure

Name of entity:	<NOUN>	
Attributes of <NOUN>		
Attribute	Type of attribute	Possible values
Validation rule attributes		
Attribute	Validation rule attribute	
Validation rule entity		
Attributes involved	Validation rule entity	

At its input section, *TC* also referred to the *TP* step where the project tester had to enter values for corresponding attributes and entities. Separate storage of TP and TC ensured that the same TP could be used in various TC. As can be seen, both structures were linked to the functional logic of the requirement to be tested; they combined the order in which actions were executed, as described

En español

descritas en el PP y determinadas por el *verbo*, con los valores propios del CP determinados por el o los *sustantivos* que definen las entidades y atributos referenciados en el CP y que están directamente relacionados con la jerarquía de *frames*.

Ahora bien, para poder reutilizar la información sobre los pasos u orden de ejecución de las acciones que están en el PP en nuevos proyectos, ha sido necesario definir un conjunto de patrones de PP (*Patrón_PP*) basados en las definiciones de patrones de Delgado (2006).

Cada patrón representa el conjunto de pasos que debe seguir el probador durante la ejecución de la prueba, para acciones específicas como por ejemplo "registrar", independientemente de si se desea probar la funcionalidad "registrar estudiante" o la funcionalidad "registrar profesor". Con esta definición, cada PP se relaciona con uno o varios Patrón_PP, lo que permitirá reutilizar ese Patrón_PP en la generación de nuevos PP en proyectos con características similares a los almacenados en la base de proyectos.

Tabla 2. Estructura de un procedimiento de prueba.

Procedimiento de Prueba:	<VERBO> <SUSTANTIVO>
Actores:	<ACTOR>[,<ACTOR>,..., <ACTOR>]
Procedimiento de Prueba	
Acción del Actor	Respuesta del Sistema
Cursos Alternos	
Referencia al procedimiento	Cursos Alternos para el paso del procedimiento

Los Patrón_PP forman parte de una red semántica en la que existen tres tipos de relaciones: "es un", "incluye a" y "se extiende de". Estos tipos de relaciones están determinadas por las relación existente entre los verbos que identifican cada uno de los Patrón_PP. En la tabla 3 se describe el significado semántico de estas relaciones, considerando A y B dos Patrón_PP pertenecientes a la red semántica.

Tabla 3. Tipos de asociaciones en la red semántica de Patrón_PP.

Relación	Descripción Semántica
A "es un" B	Relación en la que A es una generalización del *Patrón_PP* B o B es una especialización del *Patrón_PP* A.
A "incluye a" B	Relación en la que A incluye íntegramente al *Patrón_PP* B como parte de su secuencia de ejecución.
B "extiende de" A	Relación en la que A puede extenderse para ejecutar como parte de su secuencia el *Patrón_PP* B.

Hay un cuarto tipo de relación entre los verbos que expresa su equivalencia con la relación "es equivalente a", por ejemplo, entre los verbos "registrar" y "crear". Este tipo de relación se considerará asociando un único Patrón_PP al conjunto de verbos equivalentes.

El Patrón_PP contiene ranuras que podrán ser instanciadas en el momento de la adaptación de los PP de un problema particular. En la tabla 4 aparecen los tipos de ranuras que pueden aparecer en un Patrón_PP y donde pueden ubicarse los valores para su instanciación durante el proceso de adaptación de PP.

Determinación de la semejanza entre proyectos

Para poder ejecutar la fase de recuperación del sistema de RBC, es de-cir, recuperar los proyectos (casos), es necesario antes analizar cuáles son los rasgos que distinguen a cada proyecto de software. En este caso se considera como rasgo predictor a cada requisito del proyecto, tal como se presenta en la propuesta realizada para revisiones a proyectos de software, en Delgado (2006).

In English

in the TP, with TC input values. TP were determined by the verb and the TC were determined by nouns corresponding to the entities and attributes referenced in the TC. These entities were directly related to frame hierarchy.

A set of TP patterns (Pattern_TP) based on the pattern definitions described by Delgado (2006) had to be defined to reuse information about the order of executing actions included in a new project's TP.

Each pattern represented the set of steps to be followed by the tester during test execution. For example, for specific verb such as "to register", regardless of whether one wanted to test the functionality "register student" or "register teacher", the set of steps to be executed were the same. With this definition, each TP was linked to one or more Pattern_TP, allowing such Pattern_TP to be reused in generating TP in projects having similar characteristics to those stored in the project database.

Table 2. Test procedure structure

Test procedure:	<VERB> <NOUN>
Actors:	<ACTOR>[,<ACTOR>,..., <ACTOR>]
Test procedure	
Actor action	System response
Alternate courses	
Step of test procedure	Alternate courses for test procedure step

Pattern_TP formed part of a semantic network in which there were three types of relationship: "is a", "includes to" and "extends from". These types of relationships were determined by the relationship between verbs identifying each Pattern_TP. Table 3 describes the semantic meaning of these relationships where A and B were type Pattern_TP and belonged to the semantic network.

Table 3. Types of associations in the semantic network of Pattern_TP.

Relationship type	Semantic description
A "is a" B	Relationship in which A is a generalisation of Pattern_TP B or B is a specialisation of Pattern_TP A.
A "includes to" B	Relationship fully integrating Pattern_TP B as part A execution sequence
B "extends from" A	Relationship where A can be extended to include Pattern_TP B. execution sequence

A fourth kind of relationship named "is equivalent to", expressing equivalence, existed between the verbs. For example, the verbs "register" and "create" were equivalent. All equivalent verbs were associated with a single Pattern_TP in this kind of relationship.

The Pattern_TP contained slots that could be instantiated at the time of TP adaptation in a particular project. Table 4 shows the types of slots that could be presented in a Pattern_TP and where the values for instantiation during TP adaptation could be located.

Determining similarity between projects

To run the CBR system recovery phase, i.e. project recovery (cases), it was first necessary to analyse which were each software project's distinguishing features. In this case, each project requirement was considered as a trait predictor, as presented in Delgado (2006), in software project revisions.

En español

Tabla 4. Tipos de ranuras en un Patrón_PP.

Ranuras	Posibles valores de instanciación en proceso de adaptación
VERBO	Verbos en infinitivo entre los que existe una relación de tipo "es equivalente a".
ENTIDAD	Sustantivo o Frame con el cual se relaciona el *PP* y que forma parte de la red de *entidades o frames*.
ATRIBUTO	Valores de la columna ***Atributos*** de la sección ***Atributos de <SUSTANTIVO>*** dentro de la Estructura ***ENTIDAD*** presentada en la tabla 1.
ATRIBUTO.Tipo	Valores de la columna ***Tipo de Atributo*** de la sección ***Atributos de <SUSTANTIVO>*** dentro de la Estructura ***ENTIDAD*** presentada en la tabla 1.
Regla de validación de ATRIBUTO	Valores de la columna ***Regla de atributo*** de la sección ***Reglas de atributo*** dentro de la Estructura ***ENTIDAD*** presentada en la tabla 1.
condición inserción en BD	Valores de la columna ***Regla de entidad*** de la sección ***Reglas de entidad*** dentro de la Estructura ***ENTIDAD*** presentada en la tabla 1.
ATRIBUTO. PosiblesValores	Valores de la columna ***Posibles Valores*** de la sección ***Atributos de <SUSTANTIVO>*** dentro de la Estructura ***ENTIDAD*** presentada en la tabla 1.

La función de semejanza entre proyectos es $f(w_i,\delta_i(x_i(O_0),x_i(O_B)))=\frac{\sum_{i=1}^{n} w_i\cdot\delta_i(x_i(O_0),x_i(O_B))}{\sum_{i=1}^{n} w_i}$ donde w_i es el nivel de importancia o significación de cada requisito dentro del nuevo problema. Esto permite dar mayor importancia, dentro del proyecto que se desea revisar, a los requisitos que el analista o inspector consideren como los más significativos. El valor por defecto de w_i es 1.

$\delta_1(x_1(O_0),x_1(O_B))=f_r\left(\left(w_k,\delta_{ij}(z_1(x_1(O_0)),z_1(x_1(O_B)))\right)_{k=1}^{2}\right)$, es la función que evalúa la semejanza entre un par de requisitos y que es utilizada para combinar los valores de semejanza entre los verbos y los sustantivos para obtener el valor de semejanza entre dos requisitos.

$$f_r\left(\left(w_k,\delta_{ij}\left(z_k\left(y_i\right),z_k\left(y_j\right)\right)\right)_{k=1}^{2}\right)=\sum_{k=1}^{2} w_k\cdot\delta_{ij}\left(z_k\left(y_i\right),z_k\left(y_j\right)\right)$$

Para determinar la semejanza entre los sustantivos y los verbos se utilizó una red semántica en la cual se consideran dos tipos de relaciones: las relaciones jerárquicas ("es un", "es una instancia de") y las relaciones de equivalencia ("es equivalente a"). La función empleada en la determinación de la semejanza entre los verbos $\delta_{ij}\left(z_1(y_i),z_1(y_j)\right)$ y entre los sustantivos $\delta_{ij}\left(z_2(y_i),z_2(y_j)\right)$ es la presentada en Delgado (2006)

Los casos comparados con el nuevo problema se almacenarán en una lista lineal ordenada por valor descendente de la función f. Esta información será utilizada en la generación de la nueva solución.

Generación y adaptación de la nueva solución

Después de realizar la búsqueda se toma el caso o proyecto con valor mayor de f como potencial solución y se analiza si en este caso son cubiertos todos los requisitos que están presentes en el nuevo problema; de no ser así, se completa la solución potencial con los requisitos del segundo caso más parecido y así sucesivamente hasta completar los requisitos del nuevo problema o terminar de analizar el conjunto de casos recuperados.

In English

Table 4. Types of slots in a Pattern_TP

Slots	Instantiation possible values in adaptation
VERB	Infinitive verbs between a relationship type "is equivalent to"
ENTITY	Noun or frame to which the PP relates was part of the frame network
ATTRIBUTE	Values of the Attributes column from the Attributes of <NOUN> section in the ENTITY structure presented in Table 1
ATTRIBUTE.Type	Values of the Attributes Type column from the Attributes of <NOUN> section in the ENTITY structure presented in Table 1
Validation rule ATTRIBUTE	Values of the Validation Rule Attribute column from the Validation Rules Attributes section in the ENTITY structure presented in Table 1
BD insertion condition	Values of the Validation Rule Entity column from the Validation Rules Entity section in the ENTITY structure presented in Table 1
ATTRIBUTE. Possible Values	Values of the Possible Values column from the Attributes of <NOUN> section in the ENTITY structure presented in Table 1

The function of similarity between projects $f(w_i,\delta_i(x_i(O_0),x_i(O_B)))=\frac{\sum_{i=1}^{n} w_i\cdot\delta_i(x_i(O_0),x_i(O_B))}{\sum_{i=1}^{n} w_i}$ where w_i was the level of importance or significance of each requirement within the new problem. This gave greater emphasis to the requirements analyst or inspector considering the most significant within the project to be reviewed. The default value for w_i was 1.

$\delta_1(x_1(O_0),x_1(O_B))=f_r\left(\left(w_k,\delta_{ij}(z_1(x_1(O_0)),z_1(x_1(O_B)))\right)_{k=1}^{2}\right)$, where $f_r\left(\left(w_k,\delta_{ij}\left(z_k\left(y_i\right),z_k\left(y_j\right)\right)\right)_{k=1}^{2}\right)=\sum_{k=1}^{2} w_k\cdot\delta_{ij}\left(z_k\left(y_i\right),z_k\left(y_j\right)\right)$, was the function evaluating similarity between a pair of requirements. This function was used to combine the similarity values between nouns and verbs and obtain the value of similarity between two requirements.
A semantic network in which there were two main types of relationships, hierarchical relationships ("is a") and equivalence relations ("is equivalent to") was used for determining the similarity between nouns (and verbs). The function used for determining the similarity between verbs $\delta_{ij}\left(z_1(y_i),z_1(y_j)\right)$ and between nouns $\delta_{ij}\left(z_2(y_i),z_2(y_j)\right)$ was presented in Delgado (2006).

The cases compared with the new problem were stored in a linear list sorted by descending function value f. This information was used in generating a new solution.

Generating and adapting a new solution

After performing the search, the project having the highest similarity value f was used as a potential solution; it was checked that it covered all the requirements of a new project. In case not all requirements were covered, the potential solution was completed with the requirements of the second most similar case, and so on, until all the new project's requirements were completed or all recovered cases were analysed.

En español

Al asociar un requisito del nuevo proyecto $x_i(O_N)$ con otro de un proyecto almacenado en la base de casos $x_j(O_N)$ se puede asignar al requisito del nuevo problema el conjunto de CP y PP que es recuperado, pero antes es necesario valorar el nivel de semejanza entre ambos requisitos para valorar si pueden tomarse los CP y PP tal como aparecen en la base de casos, o es necesario hacer alguna adaptación. Si el valor de f es "1", pueden tomarse tal como están, porque es exactamente el mismo requisito, pero si no es el mismo se tienen dos casos que merecen un tratamiento diferenciado:

1. El valor de f está en [μ, 1), donde μ es un valor de umbral por encima del cual se pueden considerar semejantes un par de requisitos. El algoritmo 1 (Figuras 1 y 3) muestra cómo se realiza la adaptación en este caso.
2. El valor de f es "0"; no se encontró un requisito en la base de casos semejante al requisito del nuevo proyecto. El algoritmo 2 (Figura 2) muestra cómo se realiza la adaptación en este caso.

En el desarrollo de estos algoritmos se utilizan dos umbrales que permiten decidir cuándo un par de verbos y un par de sustantivos de requisitos pueden ser considerados semejantes, ellos son: μ_1 y μ_2 definidos para los verbos y sustantivos, respectivamente.

Algoritmo 1. Adaptación de PP a partir de requisito semejante

Entrada: $x_i(O_N), x_j(O_P)$ (nuevo requisito y requisito del caso almacenado),

$\overline{\alpha_2} = (y_1, y_2, ..., y_m)$ (vector que contiene los *PP* asociados a $x_j(O_P)$)

Salida: $\overline{\beta_2} = (\omega_1, \omega_2, ..., \omega_m)$ (vector que contiene los *PP* asociados a $x_i(O_N)$)

1) Si $\left(\phi_\psi\left(\pi_1(x_i), \pi_1(x_j)\right) \geq \mu_1\right) \wedge \left(\phi_\psi\left(\pi_2(x_i), \pi_2(x_j)\right) \geq \mu_2\right)$ (los sustantivos y verbos de los requisitos son semejantes) entonces: $\overline{\beta_1} = \overline{\alpha_1}$, $\overline{\beta_2} = \overline{\alpha_2}$

2) SINO

   a) Si $\left(\phi_\psi\left(\pi_1(x_i), \pi_1(x_j)\right) = 1\right)$ (los verbos de los requisitos son equivalentes) entonces: $\overline{\beta_1} = \overline{\alpha_1}$

   b) Sino entonces, Para cada Patrón_PP en la red de patrones, Comprar semejanza entre verbo del patrón y verbo de requisito hasta encontrar uno equivalente.

      i) Si se encontró el verbo equivalente entonces

         (1) $\overline{\delta}$ = Instanciar Ranuras de *PP*, donde $\overline{\delta} = (v_1, v_2, ..., v_l)$ es un vector que contiene un conjunto de Patrones de *PP* tal que v_1 es un patrón de la red semántica de patrones cuyo identificador es equivalente al verbo del requisito $x_i(O_N)$ y los restantes v_i, con $2 \leq i \leq l$, son patrones derivados de v_1. Utilizar Algoritmo 3.

         (2) $\overline{\beta_2} = \overline{\delta}$

Figura 1. Algoritmo adaptación de PP

Algoritmo 2. Generación de PP para requisito sin semejantes

Entrada: $x_i(O_N)$ (nuevo requisito),

Salida: $\overline{\beta} = (\omega_1, \omega_2, ..., \omega_m)$ (vector que contiene los *PP* asociados a $x_i(O_N)$)

1) Generar Procedimientos de Prueba para el requisito $x_i(O_N)$

   a) Para cada Patrón_PP en la red de patrones, Comprar semejanza entre verbo del patrón y verbo de requisito hasta encontrar uno equivalente.

   b) Si se encontró el verbo equivalente entonces

      i) $\overline{\delta}$ = Instanciar Ranuras de *PP*, donde $\overline{\delta} = (v_1, v_2, ..., v_l)$ es un vector que contiene un conjunto de Patrones de *PP* tal que v_1 es un patrón de la red semántica de patrones cuyo identificador es equivalente al verbo del requisito $x_i(O_N)$ y los restantes v_i, con $2 \leq i \leq l$, son patrones derivados de v_1. Utilizar Algoritmo 3.

      ii) $\overline{\beta} = \overline{\delta}$

Figura 2. Generación de PP para requisito sin semejantes.

Los PP generados son utilizados para generar los CP correspondientes utilizando la información almacenada en la red de entidades y el conjunto de PPP que ya han sido generados. Los PP y CP obtenidos son puestos a consideración del usuario y éste puede hacer ajustes a

In English

When a requirement from the new project $x_i(O_N)$ was associated with another from a project stored in the case database $x_j(O_N)$, the set of TC and TP which have been recovered can be assigned to the new project requirement. However, the degree of similarity between the two requirements had to be assessed to decide whether they could take the TC and TP as they appeared in the case database or whether some adaptation were needed. If the value of f was "1" TC and TP could be taken as they were because it was exactly the same requirement, otherwise two cases would deserve special treatment:

1. Value of f was [μ, 1), where μ was a threshold value. If the value of f was greater than μ then the pair of requirements was similar. Algorithm 1 (Figure 1 y 3) shows how the adjustment was made in this case.

2. Value of f was "0". No requirement found in the case database similar to the requirement of a new project. Algorithm 2 (Figure 2) shows how the adjustment was made in this case.

Two thresholds values were used in developing these algorithms to decide when a pair of verbs and a few nouns could be considered similar. They were defined for verbs μ_1 and μ_2 for nouns.

Algorithm 1. Adaptation of test procedure from similar requirement

Input: $x_i(O_N), x_j(O_P)$ (new requirement and stored requirement)

$\overline{\alpha_2} = (y_1, y_2, ..., y_m)$ (Vector containing the TP related to $x_j(O_P)$)

Output: $\overline{\beta_2} = (\omega_1, \omega_2, ..., \omega_m)$ (vector containing the TP related to $x_i(O_N)$)

1) IF $\left(\phi_\psi\left(\pi_1(x_i), \pi_1(x_j)\right) \geq \mu_1\right) \wedge \left(\phi_\psi\left(\pi_2(x_i), \pi_2(x_j)\right) \geq \mu_2\right)$ (nouns and verbs are similar) THEN: $\overline{\beta_1} = \overline{\alpha_1}$, $\overline{\beta_2} = \overline{\alpha_2}$

2) ELSE

   a) IF $\left(\phi_\psi\left(\pi_1(x_i), \pi_1(x_j)\right) = 1\right)$ (verbs are equivalent) THEN: $\overline{\beta_1} = \overline{\alpha_1}$

   b) ELSE, FOR EACH Pattern_TP in the patterns network, compare similarity between verb and verb pattern until to find an equivalent.

      i) IF (equivalent verb is found) THEN

         (1) $\overline{\delta}$ = Instantiate Slots of *TP*, where $\overline{\delta} = (v_1, v_2, ..., v_l)$ is a vector that contains a set of TP such that v_1 is a pattern of the semantic network whose identifier is equivalent to the verb of the new requirement $x_i(O_N)$ and other v_i, with $2 \leq i \leq l$, are patterns derived from v_1. Using Algorithm 3.

         (2) $\overline{\beta_2} = \overline{\delta}$

Figure 1. Adapting test procedure

Algorithm 2.TP generation for requirement with no similar

Input: $x_i(O_N)$ (new requirement),

Output: $\overline{\beta} = (\omega_1, \omega_2, ..., \omega_m)$ (vector containing the TP related to $x_i(O_N)$)

1) Generate Test Procedures for requirement $x_i(O_N)$

   a) FOR EACH Pattern_TP in the patterns network, compare similarity between verb and verb pattern until to find an equivalent.

   b) IF the equivalent verb is found THEN

      i) $\overline{\delta}$ = Instantiate Slots of *TP*, where $\overline{\delta} = (v_1, v_2, ..., v_l)$ is a vector that contains a set of TP such that v_1 is a pattern of the semantic network whose identifier is equivalent to the verb of the requirement $x_i(O_N)$ and other v_i, with $2 \leq i \leq l$, are patterns derived from v_1. Using Algorithm 3.

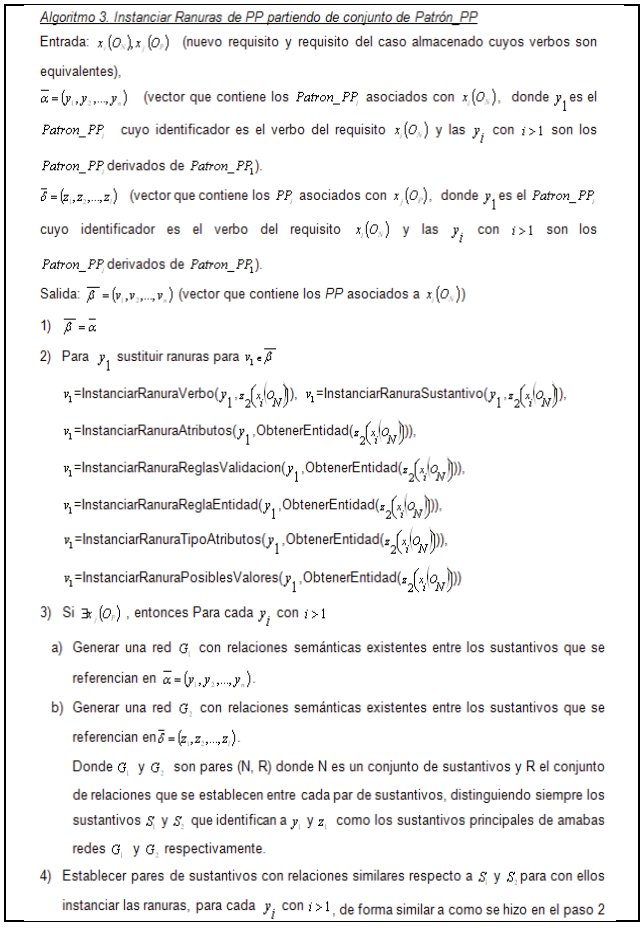
      ii) $\overline{\beta} = \overline{\delta}$

Figure 2. TP generation for requirement having no similarity

The TP so generated were used for generating TC for using information stored in the network of entities and the set of patterns which had been generated. The TP and TC obtained were presented to the user. Doing so, adjustments could be made to the solution and new

En español

la solución e incorporar nuevas consideraciones de acuerdo a la situación concreta que modela el proyecto, y esa nueva solución se almacenará en la base de ejemplos como un nuevo caso.



Algoritmo 3. Instanciar Ranuras de PP partiendo de conjunto de Patrón_PP

Entrada: $x_i(O_{x_i}), x_j(O_{x_j})$ (nuevo requisito y requisito del caso almacenado cuyos verbos son equivalentes),

$\bar{\alpha} = (y_1, y_2, ..., y_n)$ (vector que contiene los *Patron_PP* asociados con $x_i(O_{x_i})$, donde y_1 es el *Patron_PP* cuyo identificador es el verbo del requisito $x_i(O_{x_i})$ y las y_i con $i > 1$ son los *Patron_PP* derivados de $Patron\_PP_1$).

$\bar{\delta} = (z_1, z_2, ..., z_n)$ (vector que contiene los *PP* asociados con $x_i(O_{x_i})$, donde y_1 es el *Patron_PP* cuyo identificador es el verbo del requisito $x_i(O_{x_i})$ y las y_i con $i > 1$ son los *Patron_PP* derivados de $Patron\_PP_1$).

Salida: $\bar{\beta} = (v_1, v_2, ..., v_n)$ (vector que contiene los *PP* asociados a $x_i(O_{x_i})$)

1) $\bar{\beta} = \bar{\alpha}$
2) Para y_1 sustituir ranuras para $v_1 \in \bar{\beta}$

   v_1=InstanciarRanuraVerbo(y_1,$z_2(x_i(O_N))$), v_1=InstanciarRanuraSustantivo(y_1,$z_2(x_i(O_N))$),

   v_1=InstanciarRanuraAtributos(y_1,ObtenerEntidad($z_2(x_i(O_N))$)),

   v_1=InstanciarRanuraReglasValidacion(y_1,ObtenerEntidad($z_2(x_i(O_N))$)),

   v_1=InstanciarRanuraReglaEntidad(y_1,ObtenerEntidad($z_2(x_i(O_N))$)),

   v_1=InstanciarRanuraTipoAtributos(y_1,ObtenerEntidad($z_2(x_i(O_N))$)),

   v_1=InstanciarRanuraPosiblesValores(y_1,ObtenerEntidad($z_2(x_i(O_N))$))
3) Si $\exists x_j(O_{x_j})$, entonces Para cada y_i con $i > 1$
   a) Generar una red G_1 con relaciones semánticas existentes entre los sustantivos que se referencian en $\bar{\alpha} = (y_1, y_2, ..., y_n)$.
   b) Generar una red G_2 con relaciones semánticas existentes entre los sustantivos que se referencian en $\bar{\delta} = (z_1, z_2, ..., z_n)$.

   Donde G_1 y G_2 son pares (N, R) donde N es un conjunto de sustantivos y R el conjunto de relaciones que se establecen entre cada par de sustantivos, distinguiendo siempre los sustantivos S_1 y S_2 que identifican a y_1 y z_1 como los sustantivos principales de amabas redes G_1 y G_2 respectivamente.
4) Establecer pares de sustantivos con relaciones similares respecto a S_1 y S_2 para con ellos instanciar las ranuras, para cada y_i con $i > 1$, de forma similar a como se hizo en el paso 2

Figura 3. Instanciar ranuras de PP partiendo de conjunto de Patrón_PP

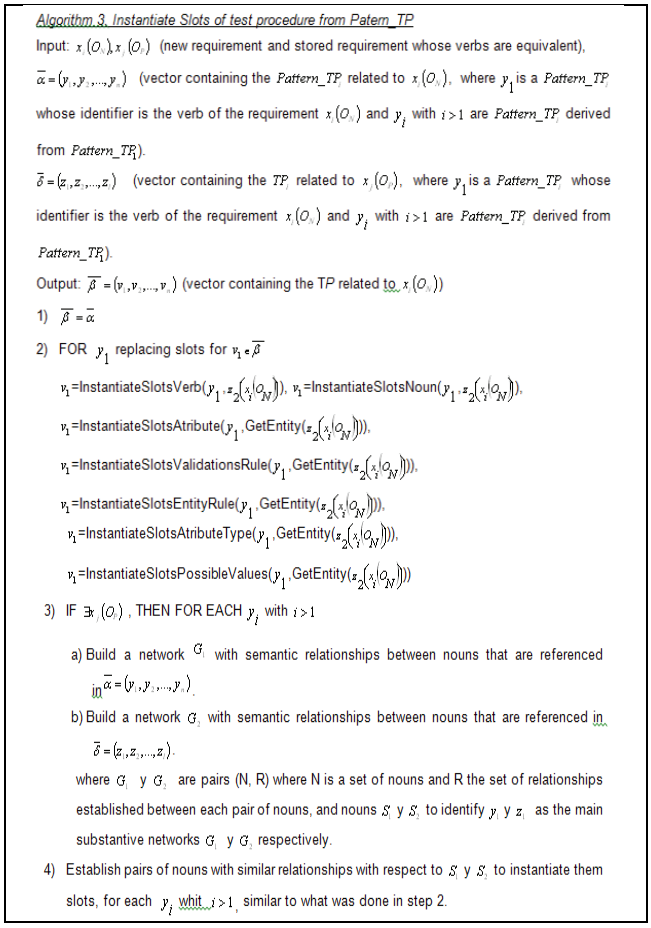
Conclusiones

Este trabajo hace una propuesta para la generación de procedimientos de pruebas funcionales, basada en el uso del razonamiento basado en casos para reutilizar información que haya sido generada con anterioridad. El aporte fundamental de la propuesta a juicio de sus autores está en el diseño de los algoritmos que permiten la recuperación y adaptación de procedimientos de prueba almacenados en la base de proyectos y que permite que se aproveche la experiencia anterior.

No obstante, la investigación no es una propuesta terminada, pues se trabaja en el desarrollo de las restantes fases del proyecto para llegar a ofrecer una solución que llegue hasta la priorización de qué casos de prueba ejecutar considerando restricciones de tiempo. En el momento en que se escribe este reporte se trabaja en la implementación de una herramienta que haga posible la experimentación en el ámbito docente como una primera prueba para comprobar los resultados que se pueden obtener con la propuesta.

In English

considerations could be incorporated, according to a new project's t specific situation. The new solution was stored in the case database as a new case.



Algorithm 3. Instantiate Slots of test procedure from Patern_TP

Input: $x_i(O_{x_i}), x_j(O_{x_j})$ (new requirement and stored requirement whose verbs are equivalent),

$\bar{\alpha} = (y_1, y_2, ..., y_n)$ (vector containing the *Pattern_TP* related to $x_i(O_{x_i})$, where y_1 is a *Pattern_TP* whose identifier is the verb of the requirement $x_i(O_{x_i})$ and y_i with $i > 1$ are *Pattern_TP* derived from $Pattern\_TP_1$).

$\bar{\delta} = (z_1, z_2, ..., z_n)$ (vector containing the *TP* related to $x_j(O_{x_j})$, where y_1 is a *Pattern_TP* whose identifier is the verb of the requirement $x_j(O_{x_j})$ and y_i with $i > 1$ are *Pattern_TP* derived from $Pattern\_TP_1$).

Output: $\bar{\beta} = (v_1, v_2, ..., v_n)$ (vector containing the TP related to $x_i(O_{x_i})$)

1) $\bar{\beta} = \bar{\alpha}$
2) FOR y_1 replacing slots for $v_1 \in \bar{\beta}$

   v_1=InstantiateSlotsVerb(y_1,$z_2(x_i(O_N))$), v_1=InstantiateSlotsNoun(y_1,$z_2(x_i(O_N))$),

   v_1=InstantiateSlotsAtribute(y_1,GetEntity($z_2(x_i(O_N))$)),

   v_1=InstantiateSlotsValidationsRule(y_1,GetEntity($z_2(x_i(O_N))$)),

   v_1=InstantiateSlotsEntityRule(y_1,GetEntity($z_2(x_i(O_N))$)),

   v_1=InstantiateSlotsAtributeType(y_1,GetEntity($z_2(x_i(O_N))$)),

   v_1=InstantiateSlotsPossibleValues(y_1,GetEntity($z_2(x_i(O_N))$))
3) IF $\exists x_j(O_{x_j})$, THEN FOR EACH y_i with $i > 1$
   a) Build a network G_1 with semantic relationships between nouns that are referenced in $\bar{\alpha} = (y_1, y_2, ..., y_n)$.
   b) Build a network G_2 with semantic relationships between nouns that are referenced in $\bar{\delta} = (z_1, z_2, ..., z_n)$.

   where G_1 y G_2 are pairs (N, R) where N is a set of nouns and R the set of relationships established between each pair of nouns, and nouns S_1 y S_2 to identify y_1 y z_1 as the main substantive networks G_1 y G_2 respectively.
4) Establish pairs of nouns with similar relationships with respect to S_1 y S_2 to instantiate them slots, for each y_i whit $i > 1$, similar to what was done in step 2.

Figure 3. Instantiate slots of test procedure from Patern_TP

Conclusions

This paper proposes generating functional test procedures based on the use of case-based reasoning for reusing previously generated information. The proposal's fundamental contribution lies in designing algorithms leading to the recovery and adaptation of test procedures stored in the project database and allows past experience to be taken advantage of.

However, this research is not a complete proposal; further work is being carried out on developing the remaining phases of the project to provide a solution establishing the priority in which test cases must be run, considering time constraints. At the moment, efforts are directed towards implementing a tool for experimenting during teaching to generate a test bench for checking the results to be obtainned from the proposal.

Bibliografía / References

Althoff, K., Case-Based Reasoning. Handbook of Software Engineering and Knowledge Engineering., Kaiserslautern, Germany, Fraunhofer Institute for Experimental Software Engineering (IESE), 2001.

Augustine, S., Managing Agile Projects. Publicaciones Prentice Hall PTR, 2005.

Black, R., Pragmatic Software Testing-Becoming and Effective and Efficient Test Profesional., Publicaciones Wiley, 2007.

Delgado, M., Lorenzo, I., Carralero, J., Travieso, J., Rosete, A., Una propuesta de apoyo a las Revisiones de Proyectos de Software utilizando Razonamiento Basado en Casos., Revista Iberoamericana de Inteligencia Artificial, Vol. 10, No. 30, 2006, pp. 55-68.

Dias, A. C., Horta, G., Supporting the Selection of Model-based Testing., Approaches for Software Projects, AST '08: Proceedings of the 3rd international workshop on Automation of software test, ACM, mayo 2008.

Elbaum, S., Rothermel, G., Malishevsky, M., Selecting a Cost-Efective Test Case Prioritization Technique., Software Quality Journal, 2004 – Springer.

Everett, G. McLeod, R., Software Testing: Testing across the entire software development life cycle., John Wiley Edition, 2007.

Fraser, G., Wotawa, F., Test-Case Prioritization with Model-Checkers, 25th conference on IASTED International, 2007.

Ganesan, K., Khoshgoftaar, T., Allen, E., Case-Based Software Quality Prediction., International Journal of Software Engineering and Knowledge Engineering, Vol. 10, No. 2, 2000, pp. 139-152.

García, F., Corchado, J., Laguna, M., CBR Applied to Development with Reuse Based on mecanos., Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering, Buenos Aires, Argentina, 2001.

Gutiérrez, J., Generación automática de objetivos de prueba a partir de casos de uso mediante partición de categorías y variables operacionales., XVI Jornadas de Ingeniería del Software y Bases de Datos, JISBD07, España, 2007.

Guvenc, M., Writing Testable and Code-able Requirements., Quality Software and Testing, Volumen 4, 2006.

Heimdahl M., George D., Test-Suite Reduction for Model Based Tests: Effects on Test Quality and Implications for Testing., 19th IEEE International Conference on Automated Software Engineering (ASE'04), 2004, pp. 176-185.

Jacobson, I., Booch, G., Rumbaugh, J., El Proceso Unificado de desarrollo de Software., Vol. 1, Editorial Félix Varela, 2004.

Ko, A., Myers, B., Extracting and Answering Why and Why Not Questions about Java Program Output., ACM Transactions on Software Engineering and Methodology, Vol. 20, No. 2, Article 4, August 2010.

Last, M., The Uncertainty Principle of Cross-Validation., 2006 IEEE International Conference on Granular Computing, 2006.

Masood, A., Bhatti, R., Ghafoor, A., Mathur, A., Scalable and Effective Test Generation for Role-Based Access Control Systems., IEEE Transactions on Software Engineering, Vol. 35, No. 5, September/October 2009.

Meyer, B., XP and TDD: Extreme Programing and Test-Driven Development., Chair of Software Engineering, Zurich, 2006.

Mogyorodi, E., Requirements-Based Testing: Ambiguity Reviews., Testing Experience: The Magazine for Professional Testers, 2008.

Myers, G. J., The Art of Software Testing., 2da Edición, Editorial John Wiley, 2004.

Naslavsky, L., Ziv, H., Richardson, D., Using Model Transformation to Support Model-BasedTest Coverage Measurement., AST '08: Proceedings of the 3rd international workshop on Automation of software test, ACM, mayo 2008.

Polo, M., Priorización de casos de prueba mediante mutación., Taller sobre Pruebas de Ingeniería del Software, PRIS 2007. Vol. 1, No. 4.

Pressman, R., Ingeniería del Software: un enfoque práctico., Vol. 1, Editorial Félix Varela, 5ta edición, 2005.

Shepperd, M., Case-based Reasoning and Software Engineering., Empirical Software Engineering Research Group, Universidad de Bournemouth, UK, 2003.

Singh, D., Misra, A. K., Software Test Effort Estimation, ACM SIGSOFT Software Engineering Notes, Vol. 33, No. 3, May 2008.

Xie, Q., Memon, A., Designing and Comparing Automated Test Oracles for GUI-Based Software Applications., ACM Transactions Software Engineering and Methodology, Vol. 16, No. 1, Article 4, February 2007.