Una herramienta de apoyo a las Revisiones de Proyectos de Software utilizando Razonamiento Basado en Casos

Martha Delgado Dapena, Iren Lorenzo Fonseca, Josué Carralero Iznaga, Javier Travieso Arencibia, Alejandro Rosete Suárez

Centro de Estudios de Ingeniería de Sistemas (CEIS), Instituto Superior Politécnico "José Antonio Echevarría" (CUJAE), Habana, Cuba
{marta,ilorenzo,jcarralero,jtravieso,rosete}@ceis.cujae.edu.cu

Resumen
En este trabajo se presenta una propuesta de estructura de almacenamiento y de función de semejanza entre proyectos de software para aplicar el Razonamiento Basado en Casos (RBC) en las Revisiones a estos. Estas revisiones se enfocan en la etapa de Requisitos, debido a la importancia de la detección de los defectos en las etapas tempranas del desarrollo de los proyectos. 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. Además se describen las características generales de una herramienta automatizada que implementa el RBC de forma que se pueda acumular la experiencia en la detección de defectos en los proyectos de software y emplearla en la ejecución de nuevas revisiones.

Palabras clave: Razonamiento Basado en Casos, Ingeniería de Software, Revisiones, Inteligencia Artificial.

1. Introducción

El tema de la calidad del software tiene gran actualidad en el mundo. 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 real de recursos a consumir es casi impredecible y el resultado casi nunca tiene la calidad y profesionalidad requerida.

Varios trabajos reflejan la importancia de establecer un Proceso de Revisión en las empresas de software [CMMI02, Basili01, Biffl00, O'Neill00], sustentado en que las dos terceras partes de los defectos de los sistemas son el resultado de errores cometidos en etapas tempranas del desarrollo del proyecto, por lo que se hace necesario prevenir los defectos o detectarlos en esas etapas [August04, McEwen04]. Schulmeyer [Schulmeyer97] plantea que las revisiones al software son un potente método para la detección de defectos que encuentran de un 60 a un 90% del total de estos, y proveen retroalimentación que permitirá a los desarrolladores evitar la inserción de defectos en trabajos futuros. Algunas empresas, que han aplicado las Revisiones han obtenido resultados alentadores [Markus00].

Con la aparición del Lenguaje Unificado de Modelado [OMG03, Rumbaugh99], la definición del Proceso Unificado de Rational [Jacobson00] y la amplia utilización de ambos, que los han convertido en estándares empleados por una gran cantidad de empresas de software, se abre una vía para trabajar

en la detección de defectos de forma automatizada. Existen en el mercado herramientas automatizadas, como las que componen el paquete de Rational [Rational05, Rational01], que detectan defectos relacionados con el balance entre artefactos y otros chequeos de consistencia, y otras dedicadas a la detección automatizada de defectos a partir del código [Abits05, Markosian03, Reitzig03], que detectan fundamentalmente errores sintácticos.

Sin embargo estas herramientas no detectan defectos relacionados con la semántica propia de cada trabajo, como puede ser la omisión de algún requisito, que es el tipo de defecto que se presenta en las etapas tempranas de los proyectos. Para abordar la detección de estos defectos relacionados con el modelado de sistemas, es conveniente analizar las semejanzas entre los proyectos a diseñar y las soluciones que han sido dadas con anterioridad a problemas del mismo tipo, y considerar los defectos que han sido detectados en Revisiones a proyectos similares, con el fin de prevenir su ocurrencia en etapas posteriores del proyecto.

Según Schulmeyer la manera más efectiva de realizar revisiones es utilizar solamente personal entrenado en la conducta propia de las revisiones [Schulmeyer97]. La experiencia de los inspectores determina la efectividad de la revisión, pues podrán detectar más defectos [Biffl00]. Además, es importante que el inspector conozca los defectos detectados y la solución dada a estos en proyectos similares, pues esto le dará mayores posibilidades para encontrar los defectos.

Sin embargo, en la Pequeña y Mediana Empresa (PYME) el personal no está entrenado en la conducta de las revisiones, pues en general son pocos especialistas dedicados al desarrollo de software y en la mayoría de los casos es personal de poca experiencia en la detección de defectos [Febles04, Delgado04]. Una solución a este problema es incorporar herramientas que ayuden a los desarrolladores y revisores a detectar la mayor cantidad de defectos en cada revisión y contar con la experiencia acumulada en el desarrollo de proyectos anteriores y en las revisiones realizadas a ellos. En estas condiciones puede pensarse en trabajar en la acumulación, en la computadora, de la experiencia que se vaya obteniendo en cada una de las revisiones, de forma tal que se disponga de un banco de casos para apoyar la detección de defectos en nuevos proyectos.

Para lograr una herramienta de apoyo a este tipo de decisión parece aconsejable utilizar técnicas de Inteligencia Artificial (IA). Si se considera además que a los posibles expertos les resulta muy difícil establecer cadenas de reglas generalizables que permitan inferir los defectos en cualquier tipo de proyecto, y les es más fácil expresar su conocimiento en términos de casos ya ocurridos, parece aconsejable emplear el Razonamiento Basado en Casos (RBC) [Althoff01, Manjares01].

El RBC es una técnica de Inteligencia Artificial que permite aprovechar la experiencia acumulada en la solución de nuevos problemas [Bello02, Turban98]. Con esta técnica se almacenan casos con la solución que se ha dado anteriormente y cuando se presenta un nuevo problema esta información o experiencia acumulada es empleada para resolverlo.

Esta técnica intenta llegar a la solución de nuevos problemas, de forma similar a como lo hacen los seres humanos [Bergmann99]. Cuando un individuo se enfrenta a un nuevo problema comienza por buscar en su memoria experiencias anteriores similares a la actual y a partir de ese momento establece semejanzas y diferencias y combina las soluciones dadas con anterioridad para obtener una nueva solución.

Los casos están compuestos por un conjunto de atributos que describen el problema y entre los que se encuentran los rasgos predictores y por la solución dada al problema descrito [Watson97].

El funcionamiento del RBC comprende procesos estrechamente relacionadas y que hacen posible el objetivo de proponer solución a nuevos problemas a partir de soluciones dadas con anterioridad a problemas similares, estos procesos son: la recuperación de los casos más parecidos, la adaptación de la solución propuesta y el almacenamiento o incorporación de la nueva solución como parte de un nuevo caso.

El RBC comienza a tener aplicaciones en la Ingeniería de Software [InrecaII03, OOReuse03, ROSA03, Shepperd03, GarcíaF01, Ganesan00, GarcíaF00, Vicinanza91]. No se conocen aplicaciones anteriores del RBC a la recuperación de proyectos similares, teniendo en cuenta la semejanza de sus requisitos. Este problema tiene retos particulares para el RBC como son: los requisitos se expresan en lenguaje natural y la semejanza no la da un requisito aislado sino un grupo de estos.

2. Razonamiento basado en casos aplicado a la detección de defectos en Proyectos de Software

2.1. 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 de Proyectos (ver

Figura 1). Los proyectos podrán ser recuperados utilizando como característica distintiva los requisitos preliminares de cada uno de ellos.

Para su almacenamiento se han definido un grupo de patrones que tienen como punto de partida los definidos en [Durán97]. Analizando los patrones de requisitos presentados por Durán se propone utilizar el Patrón de Requisito de Información *Patrón_Ri* y el Patrón de Requisito Funcional *Patrón_Rf*, además se entiende pertinente incorporar uno nuevo, el Patrón de Requisito de Información del Dominio *Patrón_Rd*.

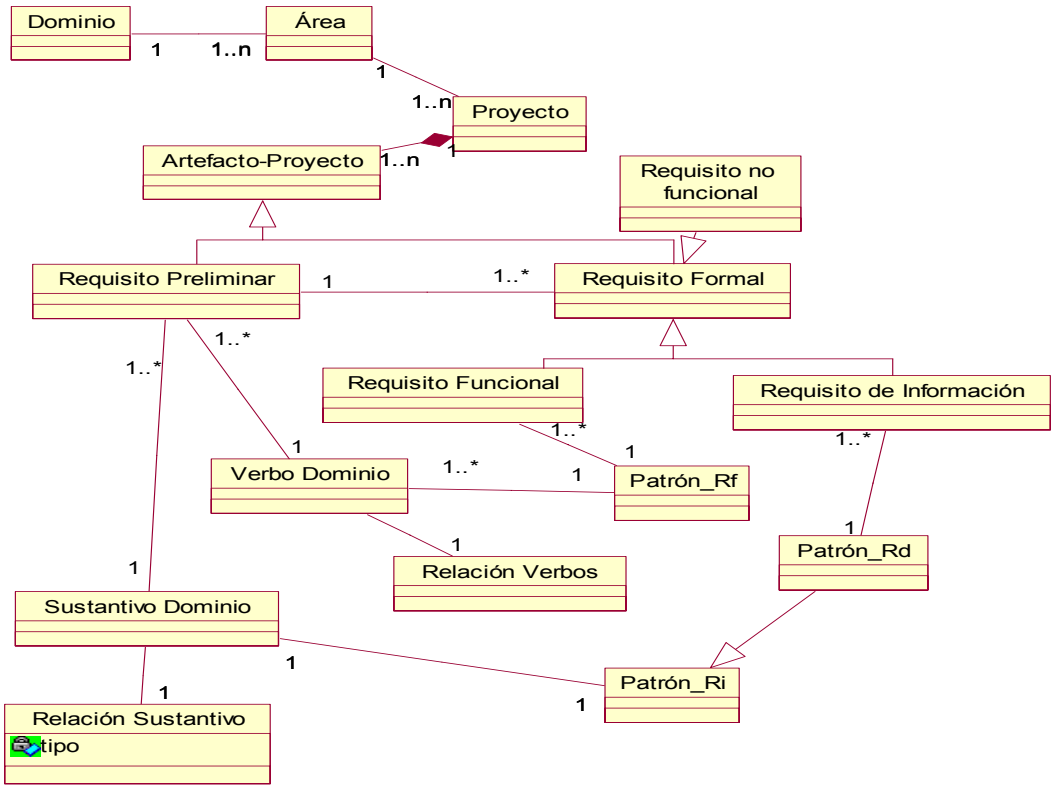

Figura 1. Diagrama UML que muestra la información de la base de casos.

Los Patrones de Requisitos de Información y los Patrones de Requisitos de Dominio, son patrones de información pero con diferente nivel de generalidad. Ellos tienen la misma estructura, la diferencia está en que los patrones de dominio almacenan atributos de entidades que son propias de un dominio determinado y en el que clasifican solo un grupo de proyectos de software, y los patrones de información constituyen generalizaciones de varios patrones de dominio y almacenan los atributos comunes a un grupo de patrones de dominio que pueden pertenecer a dominios diferentes. Por ejemplo, un *Patrón_Rd* puede representar la información de un profesor (categoría docente, departamento docente, etc.) y otro la de un estudiante (especialidad, año docente, etc.), por su parte el *Patrón_Ri* correspondiente puede representar los datos generales de cualquier persona (número de identidad, nombre, edad, etc.).

Cada patrón puede estar relacionado con varios requisitos, lo que permitirá hacer una propuesta de requisito complementado con una descripción formal del mismo, sin que necesariamente el requisito presente en el nuevo proyecto se encuentre explícitamente en el repositorio. Cada requisito preliminar almacenado en el repositorio contendrá una referencia a un verbo y otra a un sustantivo ubicados en una red semántica. Además tiene una relación con el conjunto de requisitos de información y requisitos funcionales que caracterizan su comportamiento dentro del proyecto de software. Cada requisito de información está asociado con un *Patrón_Rd,* a su vez éste tendrá una relación de tipo "es un" con un *Patrón_Ri*. El *Patrón_Ri* está relacionado con varios sustantivos del dominio que son equivalentes entre sí.

El requisito funcional, por su parte, referencia un *Patrón_Rf*, que contiene la especificación de la secuencia de pasos que genera la acción del verbo

correspondiente contenido en el requisito. A su vez el *Patrón_Rf* se relaciona con un conjunto de verbos del dominio que son equivalentes entre sí.

Tanto el *Patrón_Rd* como el *Patrón_Rf* contienen ranuras que podrán ser instanciadas en el momento de la adaptación de los requisitos de información (*Ranura_Rd* y *Ranura_Ri*) y requisitos funcionales (*Ranura_Rf*) de un problema particular a otro.

Una *Ranura_Rd* es una referencia a un sustantivo que se encuentra sin instanciar en el *Patrón_Rd* y que tomará el valor del propio sustantivo con el que está conectado o de otro sustantivo equivalente en el momento en que se esté adaptando el requisito de dominio correspondiente.

Una *Ranura_Rf* es una referencia a actores, verbos o atributos del requisito particular y que en el *Patrón_Rf* están sin instanciar, pues dependerán del requisito particular con el que se trabaje. Por ejemplo, los verbos en infinitivo que forman parte del nombre del patrón son ranuras que podrán ser instanciadas con cualquiera de los verbos que pertenecen a la lista de verbos equivalentes o sinónimos, según sea el caso del requisito funcional que se analice.

2.2. Determinación de la semejanza entre proyectos

Para poder ejecutar la fase de recuperación del sistema de RBC, es decir 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. Ahora bien, la cantidad de estos rasgos predictores de un nuevo proyecto y de otro de la base de casos no tiene porque ser el mismo. Como consecuencia de esto, es necesario determinar qué pares de requisitos (uno del nuevo problema y otro del caso almacenado) serán considerados en la función de semejanza entre ellos $\delta_i(x_i(O_0), x_i(O_B))$ del Algoritmo de recuperación de los proyectos.

Una vez determinados los pares de requisitos $\overline{\lambda} = ((x_{1o}, x_{1b}), (x_{2o}, x_{2b}), ..., (x_{\ell o}, x_{\ell b}))$ semejantes se ejecuta el Algoritmo de recuperación de los proyectos, considerando la siguiente función:

$$\delta_i(x_i(O_0), x_i(O_B)) = f_r\left(\left(w_k, \delta_{ij}(z_i(x_i(O_0)), z_i(x_i(O_B)))\right)_{k=1}^{2}\right)$$

donde $x_i(O_o) = x_{io}$, $x_i(O_B) = x_{ib}$ (x_{io}, x_{ib}) es un par de valores resultante de aplicar el Algoritmo 1, que será descrito a continuación.

$$f(w_i, \delta_i(x_i(O_0), x_i(O_B))) = \frac{\sum_{i=1}^{n} w_i \cdot \delta_i(x_i(O_0), x_i(O_B))}{\sum_{i=1}^{n} w_i}$$

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.

Para realizar el pareo de los requisitos se propone emplear el Algoritmo 1, que se describe a continuación que comienza por evaluar la semejanza entre cada par de requisitos teniendo en cuenta el verbo z_1 y el sustantivo z_2 de cada uno.

Algoritmo 1. Selección de Pares

Entrada: $\overline{\psi}_o = (y_1, y_2, ..., y_n)$, $\overline{\psi}_b = (y_1, y_2, ..., y_m)$, θ (vector que contiene los requisitos del nuevo problema, vector con los requisitos del caso de la base y cantidad de valores diferentes de w_i)

Salida: $\overline{\beta} = ((x_{1o}, x_{1b}), (x_{2o}, x_{2b}), ..., (x_{\ell o}, x_{\ell b}))$ (pares de requisitos que se consideran semejantes)

1) Para cada requisito y_i de $\overline{\psi}_o$:
   a) Para cada requisito y_j de $\overline{\psi}_b$:
      i) Buscar los valores $z_1(y_i)$ y $z_1(y_j)$ (valores del verbo en los requisitos y_i y y_j).
      ii) Calcular la semejanza $\delta_{ij}(z_1(y_i), z_1(y_j))$ entre los valores anteriores.
      iii) Buscar los valores $z_2(y_i)$ y $z_2(y_j)$ (valores del sustantivo en los requisitos y_i y y_j).
      iv) Calcular la semejanza $\delta_{ij}(z_2(y_i), z_2(y_j))$ entre los valores anteriores.
      v) Completar matriz con el valor resultante de calcular f_r considerando los pesos w_1 y w_2 correspondientes al verbo y sustantivo respectivamente.

$$f_r\left(\left(w_k, \delta_{ij}\left(z_k\left(y_i\right), z_k\left(y_j\right)\right)\right)_{k=1}^{2}\right) \rightarrow [0,1]$$

2) Seleccionar pares de requisitos semejantes:
   a) A partir de la matriz creada anteriormente se crean θ submatrices, cada una de las cuales contendrá los valores correspondientes a los requisitos del nuevo problema con el mismo nivel de significación w_i.
   b) Para cada submatriz, comenzando por la de mayor valor de significación:
      i) Buscar el mayor valor presente en la submatriz.
      ii) Si el valor es único, crear el par de requisitos semejantes $\left(x_{io}, x_{ib}\right)$.
      iii) Si el valor no es único, analizar las filas (o columnas) con las que hay colisión buscando el "siguiente mayor valor" en cada una de ellas y crear el par de requisitos semejantes $\left(x_{io}, x_{ib}\right)$ entre la columna (o fila) y la fila (o columna) cuyo "siguiente mayor valor" es el menor de las filas que han colisionado.
      iv) Asignar valor cero a los elementos de la fila y columna de la submatriz, correspondientes al par creado en los pasos iii) o iv).
      v) Repetir desde el paso i) hasta que todos los elementos de la submatriz sean cero.

Para determinar la semejanza entre los sustantivos y los verbos se utilizó una red semántica en la que se consideran dos tipos de relaciones, las relaciones jerárquicas ("es un", "es una instancia de") y relaciones de equivalencia ("es equivalente a"). La función empleada en la determinación de la semejanza entre los sustantivos y entre los verbos es la siguiente:

$$\delta_{ij}\left(z_k\left(y_i\right), z_k\left(y_j\right)\right) = \begin{cases} 1 & si \quad z_k\left(y_i\right) = z_k\left(y_j\right) \vee z_k\left(y_i\right) \equiv z_k\left(y_j\right) \\ 1 - \left|\dfrac{n\left(z_k\left(y_l\right)\right) - n\left(z_k\left(y_i\right)\right)}{n}\right| & \\ si \quad \exists y_l \left(z_k\left(y_l\right) \equiv z_k\left(y_j\right) \wedge \begin{pmatrix} es\_ancestro\left(z_k\left(y_l\right), z_k\left(y_i\right)\right) \\ \vee\, es\_ancestro\left(z_k\left(y_i\right), z_k\left(y_l\right)\right) \end{pmatrix}\right) & \\ \vee \left(z_k\left(y_l\right) \equiv z_k\left(y_i\right) \wedge \begin{pmatrix} es\_ancestro\left(z_k\left(y_j\right), z_k\left(y_l\right)\right) \\ \vee\, es\_ancestro\left(z_k\left(y_l\right), z_k\left(y_j\right)\right) \end{pmatrix}\right), & \\ 1 - \left|\dfrac{n\left(z_k\left(y_l\right)\right) - n\left(z_k\left(y_m\right)\right)}{n}\right| & \\ si \quad \exists y_l \exists y_m \left(z_k\left(y_l\right) \equiv z_k\left(y_j\right) \wedge z_k\left(y_m\right) \equiv z_k\left(y_i\right)\right. & \\ \left. \wedge \begin{pmatrix} es\_ancestro\left(z_k\left(y_l\right), z_k\left(y_m\right)\right) \\ \vee\, es\_ancestro\left(z_k\left(y_m\right), z_k\left(y_l\right)\right) \end{pmatrix}\right), & \\ 1 - \left|\dfrac{n\left(z_k\left(y_i\right)\right) - n\left(z_k\left(y_j\right)\right)}{n}\right| & \\ si \quad es\_ancestro\left(z_k\left(y_j\right), z_k\left(y_i\right)\right) \vee es\_ancestro\left(z_k\left(y_i\right), z_k\left(y_j\right)\right) & \\ 0 \qquad\qquad e.o.c \end{cases}$$

La función f_r es utilizada para combinar los valores de semejanza entre los verbos y los sustantivos para obtener el valor de semejanza entre dos requisitos.

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

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.

2.3. Generación y adaptación de la nueva solución

Después de realizar la búsqueda, se toma el caso 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. De esta forma es adaptada la solución potencial original.

El proceso de completar los requisitos requiere de un análisis de la información que suministra el nuevo requisito $x_i(O_N)$ que se pretende incorporar al caso propuesto $x_i(O_P)$, pues esto puede traer consigo redundancias en la información ya existente en el caso propuesto. Para incorporar un nuevo requisito, se comparan los requisitos de información y funcionales que él aporta con cada uno de los aportados por los requisitos ya existentes en el caso propuesto. Para lograrlo se aplican los algoritmos 2 y 3, cuya función es ajustar el nuevo requisito e incorporarlo al caso propuesto.

En la comparación de los requisitos solo se considerarán como semejantes aquellos requisitos cuyo valor de semejanza sea mayor que un umbral μ_1 y μ_2 definido para los requisitos de información y requisitos funcionales respectivamente.

Algoritmo 2. Incorporación de requisito al caso propuesto

Entrada: $x_i(O_N), O_P$ (nuevo requisito a incorporar y caso propuesto),

Salida: O_P' (caso propuesto resultante de incorporar un nuevo requisito)

1) Para cada $x_i(O_p)$:
   a) Realizar ajuste del requisito $x_i(O_N)$ con respecto a $x_i(O_p)$ ejecutando el Algoritmo 3 y almacenar los valores de salida para su posterior procesamiento.

2) Realizar ajuste definitivo del requisito $x_i(O_N)$:
   a) Para cada requisito de información asociado con $x_i(O_N)$, buscar el mayor valor de semejanza devuelto por el algoritmo 4 para este requisito:
      i) Si este valor es cero entonces mantener el requisito de información tal y como está en $x_i(O_N)$ e incorporarlo al caso O_P' como parte del nuevo requisito $x_i(O_N)$.
      ii) En cualquier otro caso sustituir el requisito de información asociado con $x_i(O_N)$ por el correspondiente requisito de información presente en O_P e incorporarlo al caso O_P' como parte del nuevo requisito $x_i(O_N)$.
   b) Para cada requisito funcional asociado con $x_i(O_N)$, buscar en la columna correspondiente de la matriz de los requisitos funcionales el mayor valor:
      i) Si este valor es cero entonces mantener el requisito funcional tal y como está en $x_i(O_N)$ e incorporarlo al caso O_P' como parte del nuevo requisito $x_i(O_N)$.
      ii) En cualquier otro caso sustituir el requisito de información asociado con $x_i(O_N)$ por el correspondiente requisito funcional presente en O_P e incorporarlo al caso O_P' como parte del nuevo requisito $x_i(O_N)$.

Algoritmo 3. Ajuste de requisito al caso propuesto

Entrada: $x_i(O_N), x_i(O_P)$ (nuevo requisito a incorporar y requisito del caso propuesto),
$\overline{\alpha_1} = (y_1, y_2, ..., y_n)$ (vector que contiene los requisitos de información asociados a $x_i(O_N)$),
$\overline{\alpha_2} = (y_1, y_2, ..., y_m)$ (vector que contiene los requisitos funcionales asociados a $x_i(O_N)$)
$\overline{\delta_1} = (r_1, r_2, ..., r_l)$ (vector que contiene los requisitos de información asociados a $x_i(O_P)$),
$\overline{\delta_2} = (r_1, r_2, ..., r_t)$ (vector que contiene los requisitos funcionales asociados a $x_i(O_P)$)

Salida: $\overline{\beta_1} = (\omega_{11}, \omega_{12}, ..., \omega_{nl})$ (vector que contiene los requisitos de información asociados a $x_i(O_N)$)
$\overline{\beta_3} = (v_1, v_2, ..., v_{nl})$ (vector que contiene los valores de semejanza entre los requisitos de información comparados)
$\overline{\beta_2} = (\omega_1, \omega_2, ..., \omega_{mt})$ (vector que contiene los requisitos funcionales asociados a $x_i(O_N)$)

$\overline{\beta_4} = (v_1, v_2, ..., v_{mt})$ (vector que contiene los valores de semejanza entre los requisitos funcionales comparados)

1) Incorporar requisitos de información. Para cada y_i de $\overline{\alpha_1}$:
   a) Para cada r_j de $\overline{\delta_1}$:
      i) Si $\delta_{ij}\left(z_2(y_i), z_2(r_j)\right) < \mu_1$ (los sustantivos de los requisitos de información no son semejantes) entonces:
      $\omega_{ij} = y_i$, donde $\omega_{ij} \in \overline{\beta_1}$
      $v_{ij} = 0$, donde $v_{ij} \in \overline{\beta_3}$
      ii) En cualquier otro caso (los sustantivos son semejantes) entonces:
      $\omega_{ij} = r_j$, donde $\omega_{ij} \in \overline{\beta_1}$
      $v_{ij} = \delta_{ij}\left(z_2(y_i), z_2(r_j)\right)$, donde $v_{ij} \in \overline{\beta_3}$
2) Incorporar requisitos funcionales. Para cada y_i de $\overline{\alpha_2}$:
   a) Para cada r_j de $\overline{\delta_2}$:
      i) Si $f_r\left(\left(w_k, \delta_{ij}\left(z_k(y_i), z_k(r_j)\right)\right)_{k=1}^{2}\right) < \mu_2$ (los requisitos funcionales no son semejantes) entonces:
      $\omega_{ij} = y_i$, donde $\omega_{ij} \in \overline{\beta_2}$
      $v_{ij} = 0$, donde $v_{ij} \in \overline{\beta_4}$
      ii) En cualquier otro caso (los requisitos funcionales son semejantes) entonces:
      $\omega_{ij} = r_j$, donde $\omega_{ij} \in \overline{\beta_2}$
      $v_{ij} = \delta_{ij}\left(z_2(y_i), z_2(r_j)\right)$, donde $v_{ij} \in \overline{\beta_4}$

Una vez que han sido cubiertos todos los requisitos del nuevo problema es necesario expresar los requisitos del problema propuesto en el lenguaje en que se describe el nuevo problema. Para generar los nuevos requisitos funcionales y requisitos de información correspondientes se aplica el Algoritmo 4 a cada requisito del nuevo problema.

Algoritmo 4. Adaptación de requisito

Entrada: $x_i(O_0), x_i(O_p)$ (requisito del nuevo problema y requisito del caso propuesto),
$\overline{\alpha_1} = (y_1, y_2, ..., y_n)$ (vector que contiene los requisitos de información asociados al caso propuesto),
$\overline{\alpha_2} = (y_1, y_2, ..., y_m)$ (vector que contiene los requisitos funcionales asociados al caso propuesto)

Salida: $\overline{\beta_1} = (\omega_{i1}, \omega_{i2}, ..., \omega_{in})$ (vector que contiene los requisitos de información del caso propuesto adaptados al lenguaje utilizado en el nuevo problema)
$\overline{\beta_2} = (\omega_{i1}, \omega_{i2}, ..., \omega_{im})$ (vector que contiene los requisitos funcionales del caso propuesto adaptados al lenguaje utilizado en el nuevo problema)

1) Generar requisitos de información:
   a) Si $z_2\left(x_i(O_0)\right) = z_2\left(x_i(O_p)\right)$ (los sustantivos de los requisitos son iguales) entonces, para cada requisito de información y_j de $\overline{\alpha_1}$:
   $\omega_{ij} = y_j$, donde $\omega_{ij} \in \overline{\beta_1}$
   b) Si $z_2\left(x_i(O_0)\right) \neq z_2\left(x_i(O_p)\right)$ $\wedge$ $\delta_{ii}\left(z_2(x_i(O_0)), z_2(x_i(O_p))\right) = 1$ (sustantivos equivalentes) entonces, para cada requisito de información y_j de $\overline{\alpha_1}$:
   ω_{ij}=InstanciarRanuras(y_j), donde $\omega_{ij} \in \overline{\beta_1}$
   c) Sino (los sustantivos de los requisitos son diferentes y no equivalentes, pero semejantes) entonces, para cada requisito de información y_j de $\overline{\alpha_1}$:
      i) Si es_ancestro$\left(z_2\left(x_i(O_0)\right), z_2\left(x_i(O_p)\right)\right)$ entonces:
      ω_{ij}= Nuevo Requisito de Información que contiene el valor de las ranuras, donde $\omega_{ij} \in \overline{\beta_1}$
      asociar ω_{ij} con *Patrón_Ri*(y_j)
      ii) Sino entonces:
      ω_{ij}= Nuevo Requisito de Información que contiene el valor de las ranuras, donde $\omega_{ij} \in \overline{\beta_1}$
      asociar ω_{ij} con *Patrón_Rd*(y_j)
      Es importante hacer notar que en el Patrón_Rd están incluidas las propiedades presentes en el Patrón_Ri, por la propia relación de generalización-especialización presente entre ambos.

2) Generar Requisitos funcionales:

   a) Si $z_1\left(x_i\left(O_0\right)\right) = z_1\left(x_i\left(O_p\right)\right)$ (los verbos de los requisitos son iguales o equivalentes) entonces, para cada requisito funcional y_j de $\overline{\alpha_2}$:

   ω_{ij} = Nuevo Requisito Funcional que contiene valor de las ranuras, donde $\omega_{2i} \in \overline{\beta_2}$

   asociar ω_{ij} con *Patrón_Rf*(y_i)

Luego de realizados los pasos anteriores el caso propuesto se evalúa utilizando una lista de chequeo en la que se verifica que el caso propuesto no presente problemas de completitud, trazabilidad, alcance y otras buenas prácticas. Luego, el caso propuesto se compara con el proyecto a inspeccionar (nuevo problema) y se usa para detectar los defectos presentes en este último. Además se presentan un grupo de defectos que han sido encontrados en el proyecto recuperado de la base de casos, lo que contribuye a que el equipo de proyecto pueda trabajar en su prevención.

Los defectos detectados son puestos a consideración del usuario y si así lo entiende lo registrará en el Registro de Defectos de la Revisión como defectos detectados en el proyecto en cuestión. El usuario puede hacer ajustes a la solución e incorporar nuevas consideraciones de acuerdo a la situación concreta que modela el proyecto que se está revisando y esa nueva solución se almacenará en la base de ejemplos como un nuevo caso.

La nueva solución será considerada experiencia acumulada a partir de este momento. La base de ejemplos se enriquecerá y con ella el conocimiento y la experiencia disponible para el desarrollo de nuevos proyectos de software y el proceso de detección de defectos en las revisiones.

3. Asistente para Revisiones a Proyectos aprovechando la experiencia Anterior

A partir de las ideas anteriores, se desarrolló una herramienta automatizada que implementa la propuesta presentada en este trabajo, ARPA: Asistente para Revisiones a Proyectos aprovechando la experiencia Anterior. ARPA utiliza el RBC para hacer una propuesta a los revisores de un conjunto de defectos que están presentes en el producto que se revisa, en la fase correspondiente. Además muestra un conjunto de defectos que han sido encontrados en otros proyectos revisados anteriormente lo que puede ser utilizado también por el equipo de desarrollo para prevenir la ocurrencia de dichos defectos.

ARPA parte de los requisitos candidatos o necesidades de los usuarios y de la solución que dieron los desarrolladores, es esta última la información que se somete a revisión por parte de los inspectores. El asistente desarrollado cuenta en la actualidad con algoritmos para asistir la detección de defectos en la etapa de Requisitos. El sistema tiene tres tipos de usuarios:

- Miembro del equipo de revisión o revisor: En este caso la información requerida por el sistema es la especificación completa que se quiere Revisar, que incluye el listado de Requisitos del Proyecto y su Modelo de Casos de Uso. La respuesta del sistema para este usuario está compuesta por un listado de defectos potenciales (Ver Figura 2), encontrados en el proyecto que se está revisando, a partir de una comparación con una solución propuesta que considera la experiencia acumulada en la Base de Casos, y una propuesta de solución a estos defectos, materializada en un diagrama de casos de uso y una descripción de los requisitos (Figura 3).

- Miembro del equipo de desarrollo o desarrollador: La información a suministrar al sistema por el usuario es únicamente la lista de requisitos preliminares y el sistema de RBC devolverá al usuario un listado completo de Requisitos con su Modelo de Casos de Uso del sistema. Esta información se obtiene a partir de los casos más parecidos almacenados en la base de casos.

- Administrador de la Base de Casos: Las actividades asociadas a este usuario son las de mantenimiento de la Base de Casos. La inserción de los casos a la Base sólo puede ser realizada desde el módulo de administración, puesto que el administrador es el único con permiso para hacer variaciones en los datos que maneja la herramienta. No obstante, el resto de los usuarios pueden hacer propuestas que serán revisadas por el administrador para su inclusión o no en la Base de Casos.

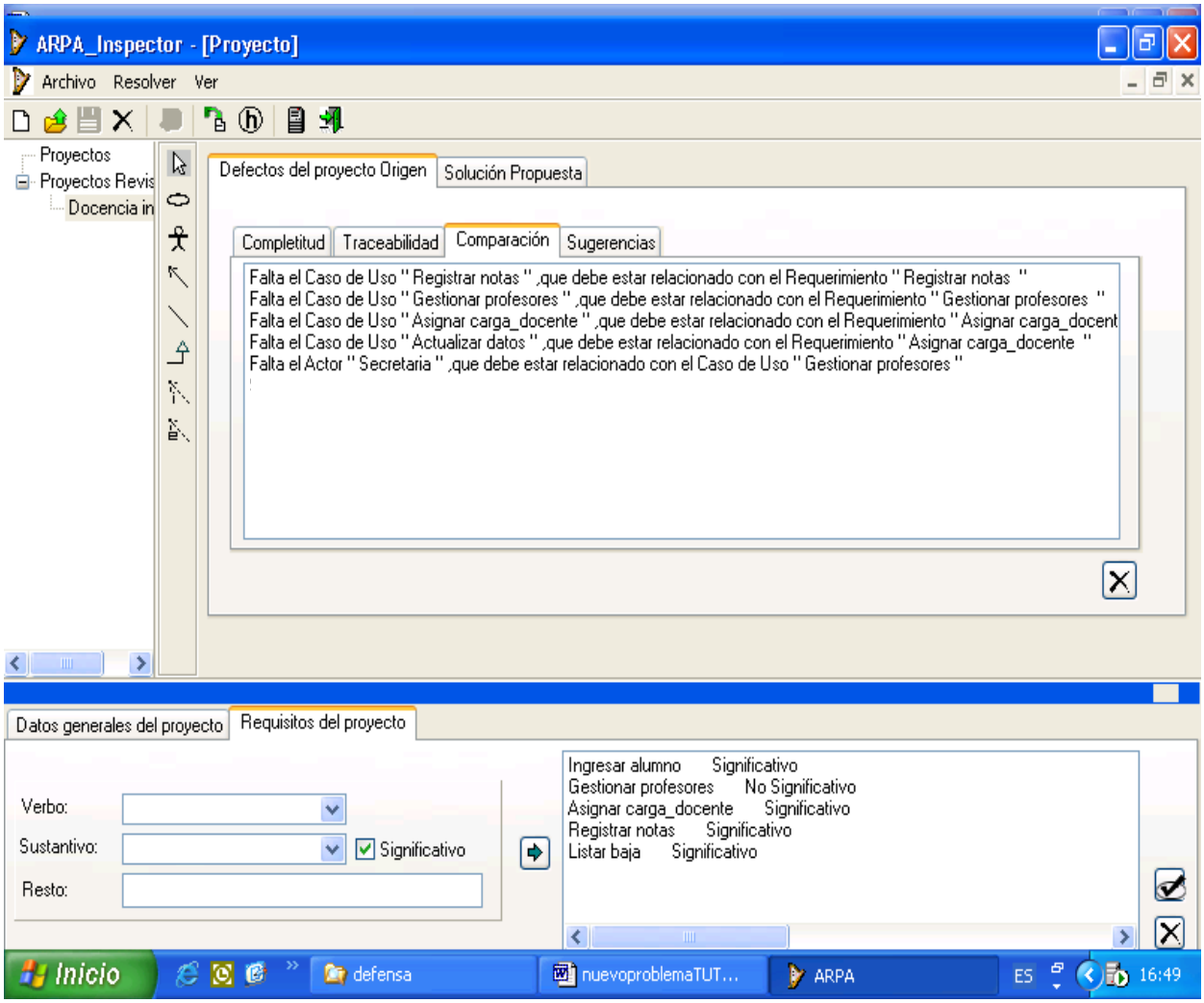

Figura 2. ARPA: Muestra de defectos detectados.

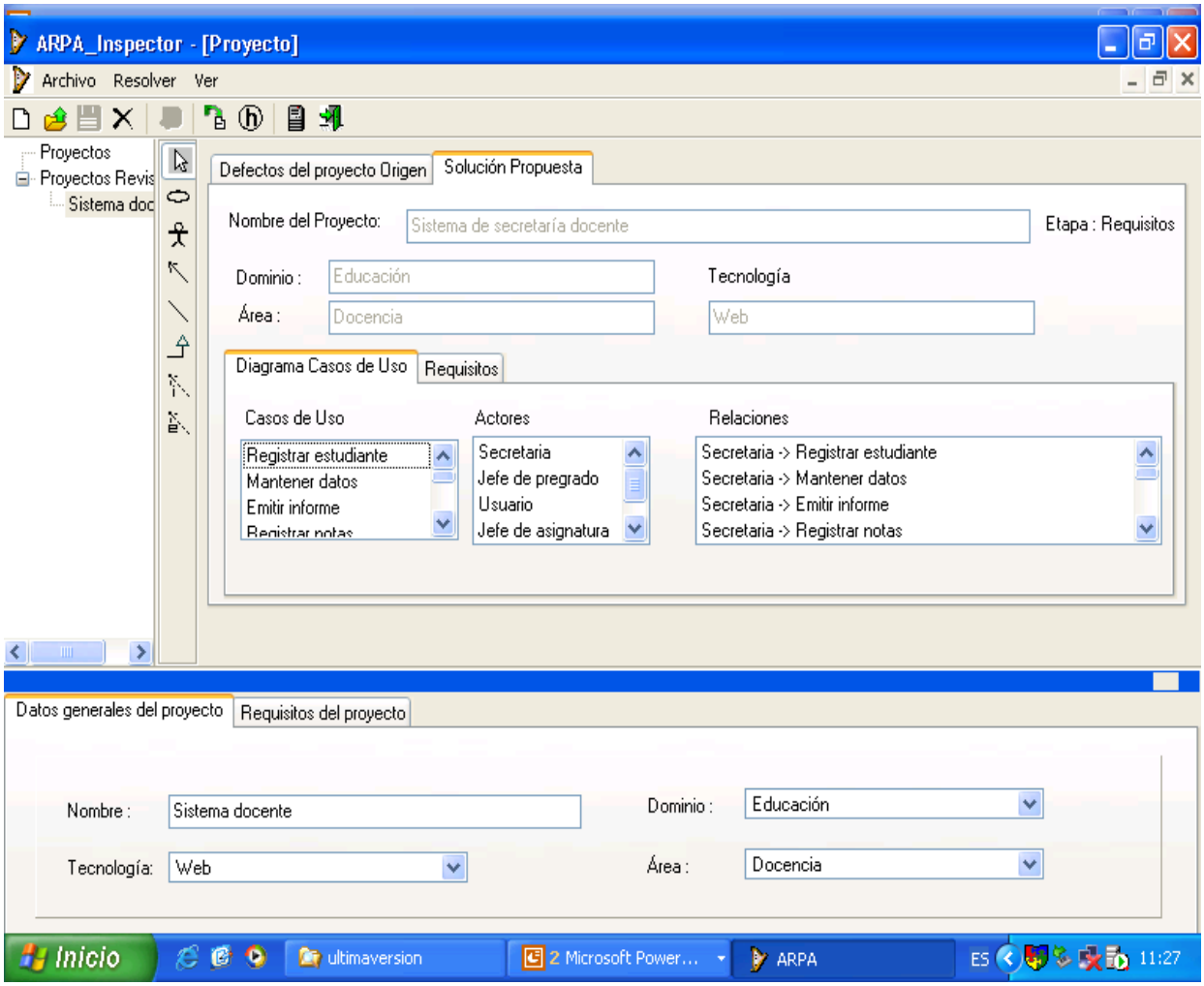

Figura 3. ARPA: Solución propuesta al revisor.

A continuación se muestra un ejemplo de la información contenida en la base de casos; así como el proyecto a revisar. En las figuras 2 y 3 se muestran algunos de los defectos mostrados al revisor y la solución que ofrece la herramienta para este ejemplo particular.

Ejemplo de proyecto a revisar
Nombre del proyecto: Docencia integral.
Dominio: Educación
Área: Pregrado
Requisitos preliminares:

1. Ingresar alumno. (significativo)
2. Gestionar profesores.
3. Asignar carga. (significativo)
4. Registrar nota. (significativo)
5. Listar Bajas. (significativo)

Modelo de Casos de Uso en la figura 4.

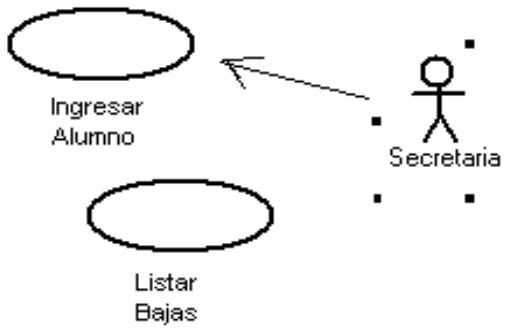

Figura 4. Diagrama de Casos de Uso del Sistema para el proyecto "Docencia Integral".

Ejemplo de información de la Base de Casos empleada por ARPA
Nombre del proyecto: Gestión del área de pregrado.
Dominio: Educación
Área: Pregrado
Requisitos preliminares:

1. Gestionar profesores.
2. Gestionar plan metodológico.
3. Asignar carga.
4. Programar actividad.
5. Mostrar asignaturas.
6. Mostrar profesores.

Modelo de Casos de Uso en la figura 5.

Nombre del proyecto: Secretaría Docente.
Dominio: Educación
Área: Pregrado
Requisitos preliminares:

1. Actualizar estudiantes.
2. Reportar promoción.
3. Registrar nota.
4. Registrar estudiante.
5. Buscar estudiante.
6. Mostrar Bajas.

Modelo de Casos de Uso en la figura 6.

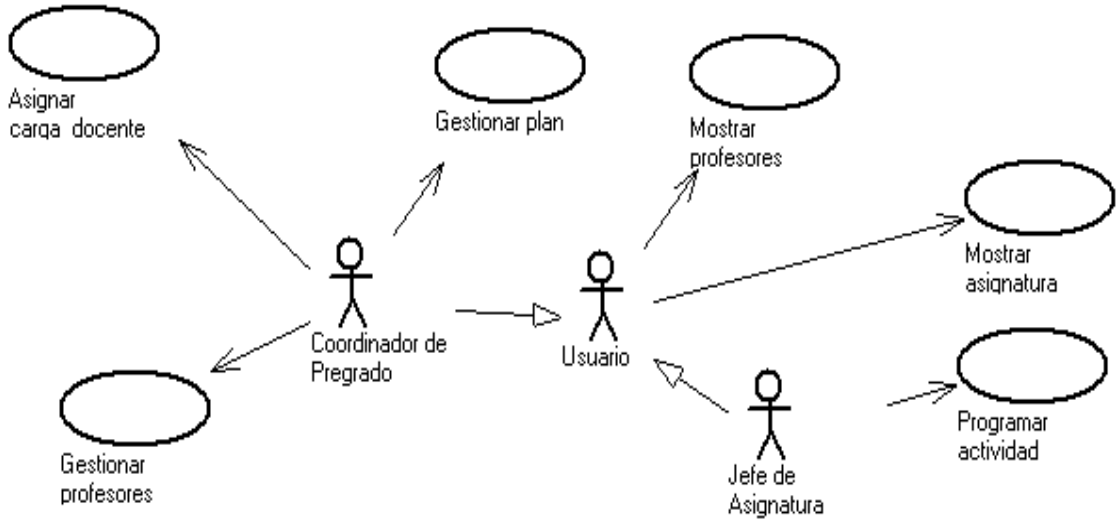

Figura 5. Diagrama de Casos de Uso del Sistema para el proyecto "Gestión del área de pregrado".

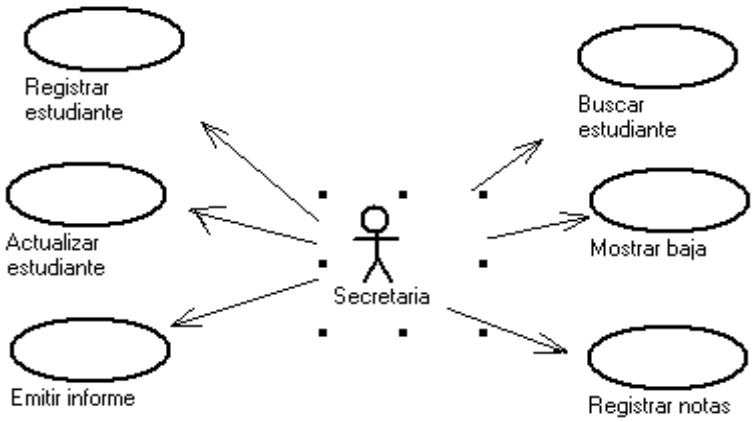

Figura 6. Diagrama de Casos de Uso del Sistema para el proyecto "Secretaría Docente".

4. Validación

Para validar la **función de semejanza entre proyectos** se realizó una ronda para conocer el criterio de los expertos con respecto a la semejanza entre un grupo de proyectos reales y posteriormente se compararon estos valores con los arrojados por la función de semejanza propuesta en el modelo.

Para ello se seleccionaron un grupo de 22 Trabajos de Diploma de los cursos 2002-2003 y 2003-2004 cuyo listado de requisitos preliminares fue presentado a los expertos junto con una matriz de 22x22. Los expertos debían asignar un valor de semejanza (entre cero y uno) para cada par de proyectos representado por el elemento S_{ij} de la matriz.

El procesamiento de los datos ofrecidos por los expertos y su comparación con los valores arrojados por la función de semejanza propuesta para cada par de proyectos dio como resultado una diferencia promedio del valor experto con el de la propuesta (sistema) de 0,12 y una varianza promedio de 0,01.

Además se solicitó a los expertos que ubicaran los valores de semejanza entre cero y uno en las categorías Ninguna, Baja, Media, Alta y Total semejanza. Con estos valores se compararon nuevamente las categorías de los expertos con el sistema y los resultados evidenciaron que existe una alta coincidencia, pues en alrededor de un 60% existe coincidencia dentro de la misma categoría y alrededor de un 39% la diferencia entre la categoría donde ubican los expertos y la categoría donde ubica el sistema es de distancia 1, tal y como se muestra en la Figura 7.

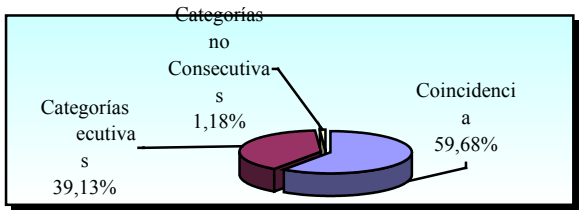

Figura 7. Comparación entre los valores arrojados por el sistema y el promedio de los expertos.

Con el objetivo de **comprobar la coincidencia o no entre los defectos detectados** por los expertos y los propuestos por los mecanismos de recuperación y determinación de defectos que se incluyen en la herramienta ARPA se realizó la siguiente comparación. Se solicitó a los profesores de la asignatura de Ingeniería de software 1, curso 2004-2005, que señalaran los defectos encontrados en una muestra de 32 proyectos reales desarrollados por los estudiantes en la asignatura.

Posteriormente se suministró a la herramienta ARPA cada uno de los 32 proyectos y se compararon los defectos propuestos por la herramienta con los encontrados por los profesores. De los 30 tipos de defectos detectados por los profesores la herramienta detectó 14 tipos, que son los 14 posibles según los mecanismos implementados hasta el momento en ARPA.
Del total de defectos relacionados con omisiones de artefactos y alcance del proyecto que la herramienta propone como posibles defectos en los proyectos que se analizaron, aproximadamente el 90% fueron considerados como válidos (o procedentes) para los proyectos particulares, en el chequeo posterior que se realizó con los profesores. Alrededor de un 16% de los defectos propuestos por ARPA no fueron detectados por los profesores. De ellos, cerca de un 10% se consideraron como procedentes (Figura 8).

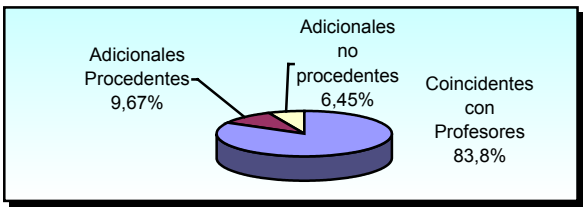

Figura 8. Comparación de los defectos detectados por ARPA y por los profesores.

Conclusiones

Este trabajo hace una propuesta de solución a la detección automática de defectos en los requisitos de los proyectos de software utilizando el Razonamiento Basado en Casos. Se describe una herramienta que asiste la detección automática de defectos, que puede constituir una importante ayuda a los desarrolladores, inspectores y directivos de la

empresa. De esta forma se aporta una solución para la gestión del conocimiento que en materia de Revisiones irá acumulando la industria de software y que será aprovechada por especialistas con menos experiencia. Además se ha presentado parte de la validación de la herramienta empleando métodos de expertos, lo que demuestra la factibilidad de empleo de esta en las pruebas realizadas hasta el momento.

Referencias

[Abits05] Abits. Software México-Latinoamérica, <http://www.abits.com/>, (2005).

[Althoff01] K. Althoff. `Case-Based Reasoning. Handbook of Software Engineering and Knowledge Engineering´. kaiserslautern, Alemania. Fraunhofer Institute for Experimental Software Engineering (IESE). (2001).

[August04] S. August. `Ending Requirements Chaos´. IBM Rational. IBM Corporation 2004. http://www-136.ibm.com/developerworks/rational/library/4853.html (2004).

[Basili01] V. Basili y otros. `Software Defect Reduction Top 10 List´. Computer, No. 34, vol. 1, pág. 135-137, January, (2001).

[Bello02] P. Bello. `Aplicaciones de la Inteligencia Artificial´. Universidad de Guadalajara, Centro Universitario de Ciencias Económico Administrativas, México, (2002).

[Bergmann99] R. Bergmann. `Developing Industrial Case Based Reasoning Applications. The INRECA Methodology´. Berlín, Alemania. Springer-Verlag, (1999).

[Biffl00] S. Biffl. `Analysis of the impact of reading technique and inspector capability on individual inspection performance´. Seventh Asia-Pacific Software Engineering Conference (APSEC'00), IEEE Inc., Diciembre (2000).

[CMMI02] CMMI Product Team. `Capability Maturity Model Integration (CMMISM). Version 1.1´. CMU/SEI-2002-TR-011, ESC-TR-2002-011, Pittsburgh PA SEI, Carnegie Mellon University, March (2002).

[Delgado04] M. Delgado, S. Álvarez, A. Roseta. `Una propuesta de introducción de las revisiones en el proceso de desarrollo de software´. Investigación Operacional, número 3, (2004).

[Durán97] A. Durán y otros. `Identificación de Patrones de Reutilización de Requisitos de Sistemas de Información´. Proyecto MENHIR de la CICYT, TIC97-0593-C05-01, Ministerio de Educación y Ciencia de España, (1997).

[Febles04] A. Febles. `Un modelo de referencia para la gestión de configuración en la pequeña y mediana empresa de software´. Tesis presentada en opción del grado de Doctor en Ciencias Técnicas. Instituto Superior José Antonio Echeverría, (2004).

[Ganesan00] K. Ganesan y otros. `CASE-BASED SOFTWARE QUALITY PREDICTION´, International Journal of Software Engineering and Knowledge Engineering, Vol. 10, No. 2, pág:139-152, (2000).

[GarcíaF00] F. García. `Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos´. PhD thesis, Universidad de Salamanca, (2000).

[GarcíaF01] F. García y otros. `CBR Applied to Development with Reuse Based on mecanos´. Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering, Buenos Aires, (2001).

[InrecaII03] Information and Knowledge Reengineering for Reasoning from Cases. AcknoSoft S.A., <http://www.cordis.lu/esprit/src/22196.htm>, (2003).

[Jacobson00] I. Jacobson. ´El Proceso Unificado de Desarrollo de Software´. Addison Wesley Longman Inc, (2000).

[Manjares01] A. Manjares. `Razonamiento basado en casos´. Universidad Nacional de Educación a Distancia, Departamento de Inteligencia Artificial, Madrid, España, (2001).

[Markosian03] L. Markosian. `Hosted Servicesfor Advanced V&V Technologies: An Approach to Achieving Adoption with out the Woes of Usage´. 3rd

International Workshop on Adoption-Centric Software Engineering, ICSE 2003 IEEE/ACM International Conference on Software Engineering, Portland, Oregon, May, (2003).

[Markus00] M. Markus. `¿Fallidos proyectos de software?, Ya no más´. Quality Progress, no.11, vol.33, p. 116-117, nov. (2000).

[McEwen04] S. McEwen. `Requirements: An introduction´, IBM Rational, IBM Corporation 2004, http://www-136.ibm.com/developerworks/rational/library/4166.html, (2004).

[OOReuse03] Research Projects. OOReuse. CBR-WEB & MLNET. http://www.ai-cbr.org, (2003).

[OMG03] Object Management Group. `Unified Modeling Language. Version 1.5´, Object Management Group Inc., (2003).

[O'Neill00] D. O'Neill. `National Software Quality Experiment: Results 1992-1999´. Software Technology Conference, Salt Lake City, (2000).

[Rational01] Rational Corporation. `Rational Unified Process´. Version 2001A.04.00, Copyright 1987-2001, (2001).

[Rational05] Rational Corporation. `IBM Software-Rational´, Rational Software 2005, http://www.ibm.com/py/ products/ software/rational.html, (2005).

[Reitzig03] R. Reitzig. `Using Rational Software Solutions to Achieve CMMI Level 2´, Rational Software 2003, http://www.therationaledge.com/content/jan_03/f_CMMI_rr.jsp, (2003).

[ROSA03] Research Projects. ROSA. CBR-WEB & MLNET. http://www.ai-cbr.org, (2003).

[Rumbaugh99] J. Rumbaugh. `The Unified Modeling Language. Reference Manual´, Addison Wesley Longman Inc, (1999).

[Schulmeyer97] G. Schulmeyer. `Handbook of Software Quality Assurance´, Prentice may, (1997).

[Shepperd03] M. Shepperd. ´Case-based Reasoning and Software Engineering´, Empirical Software Engineering Research Group, Universidad de Bournemouth, UK, (2003).

[Turban98] E. Turban. `Decision Support Systems and Intelligent Systems´. Prentice Hall Inc, Quinta edición, New Jersey, Estados Unidos. (1998).

[Vicinanza91] S. Vicinanza, `Software effort estimation: an exploratory study of expert performance´, Information Systems Research 2 (4), p. 243-262, (1991).

[Watson97] I. Watson. `Applying Case-Based Reasoning: Techniques for Enterprise Systems´. Morgan Kaufmann Publishers Inc. (1997).