Generación automática de datos de prueba mediante un enfoque que combina Búsqueda Dispersa y Búsqueda Local

Raquel Blanco[1], Javier Tuya[1] y Belarmino Adenso-Díaz[2]

[1] Departamento de Informática
[2] Departamento de Administración de Empresas
Universidad de Oviedo
Campus de Viesques s/n, Gijón, Asturias, 33204, España
{rblanco, tuya, adenso}@uniovi.es

Resumen. Las técnicas empleadas en la generación automática de datos de prueba tratan de encontrar de forma eficiente un conjunto pequeño de datos de prueba que permitan satisfacer un determinado criterio de suficiencia, contribuyendo así a la reducción del coste de la prueba del software. En este artículo se presenta un enfoque basado en la técnica Búsqueda Dispersa denominado TCSS-LS. Este generador combina la propiedad de diversidad de la Búsqueda Dispersa con la intensificación de una Búsqueda Local. La diversidad es empleada para extender la búsqueda de datos de prueba a todas las ramas de un programa bajo prueba con el fin de generar datos de prueba que las cubran, mientras que la búsqueda local intensifica la búsqueda en determinados puntos para los cuales la diversificación encuentra dificultades. Además se presentan los resultados comparativos de TCSS-LS frente a un enfoque basado únicamente en Búsqueda Dispersa denominado TCSS.

Palabras Clave: Prueba software, generación automática de datos de prueba, cobertura de ramas, Búsqueda Dispersa, técnicas de búsqueda metaheurísticas.

1 Introducción

La prueba del software es la fase del ciclo de vida de un desarrollo software cuyo objetivo es descubrir los fallos que se han producido en el resto de fases, de tal forma que puedan ser solucionados, mejorando así la calidad del producto y disminuyendo el coste derivado de dichos de fallos [25]. Sin embargo esta fase es muy costosa y se estima que suele consumir entre un 30% [15] y un 50% [4] como mínimo del coste total de un desarrollo software.

Una parte de esta fase que conlleva una labor intensiva es la generación de los datos de prueba utilizados en la construcción de los casos de prueba del producto software. Esta tarea es crucial para el éxito de la prueba, debido a que es imposible obtener un programa software completamente probado (el número de casos de prueba necesarios para probar un programa software es infinito [25]) y un diseño adecuado de los casos de prueba podrá detectar un elevado número de fallos. Además la creación de los datos de prueba, al ser principalmente manual en la actualidad, es tal vez la tarea más costosa de la prueba del software, ya que puede suponer aproximadamente el 40% de su coste total [35]. Por estas razones, los diversos métodos desarrollados para la generación automática de los datos de prueba tratan de encontrar de forma eficiente un conjunto pequeño de dichos datos que permitan satisfacer un determinado criterio de suficiencia. De esta forma se reduce el coste de la prueba del software [12][27][35], por lo que los productos software pueden ser probados más eficientemente.

Entre las técnicas más recientes que realizan esta automatización se encuentran las técnicas de búsqueda metaheurísticas. La aplicación de los algoritmos metaheurísticos a la resolución de

problemas en Ingeniería del Software fue propuesto por la red SEMINAL (Software Engineering using Metaheuristic INnovative ALgorithms) y se trata ampliamente en [10]. Una de esas aplicaciones es la selección de datos de prueba en el proceso de la prueba del software, la cual es tratada como un problema de búsqueda u optimización, como se muestra en varios trabajos que llevan a cabo una revisión de la literatura [20][21].

La técnica metaheurística más ampliamente utilizada en este campo son los Algoritmos Genéticos. Esta técnica ha sido empleada en diversos trabajos para generar datos de prueba bajo criterios de suficiencia como cobertura de ramas [3][16][24][27][29][34], cobertura de condición [2], cobertura de condición-decisión [23][35], cobertura de caminos [1][7][18][19] [33] o criterio all-uses [13]. Otras técnicas que también han sido empleadas en la generación automática de datos de prueba son la Programación Genética [31], el Recocido Simulado [19][30][32][35], la Búsqueda Tabú [11], los Algoritmos Evolutivos [9][22], la Repulsión Simulada [8] o la Búsqueda Dispersa [6][28].

En este artículo se presenta un algoritmo basado en Búsqueda Dispersa denominado TCSS-LS para la generación automática de datos de prueba bajo el criterio de suficiencia de cobertura de ramas. Para ello TCSS-LS combina la propiedad de diversidad de la Búsqueda Dispersa con la intensificación de una Búsqueda Local. La diversidad es empleada para extender la búsqueda de datos de prueba a todas las ramas de un programa bajo prueba con el fin de generar datos de prueba que las cubran, mientras que la búsqueda local intensifica la búsqueda en determinados puntos para los cuales la diversificación encuentra dificultades. Este algoritmo es una extensión del algoritmo basado en Búsqueda Dispersa denominado TCSS [5][6].

2 Búsqueda Dispersa

La Búsqueda Dispersa (Scatter Search) [14][17] es un método evolutivo que opera sobre un conjunto de soluciones, llamado Conjunto de Referencia (RefSet). Las soluciones presentes en este conjunto son combinadas con el fin de generar nuevas soluciones que mejoren a las originales. Así, el Conjunto de Referencia almacena las mejores soluciones que se encuentran durante el proceso de búsqueda, considerando para ello su calidad y la diversidad que aportan al mismo.

El funcionamiento general de la Búsqueda Dispersa consiste en la generación de un conjunto P de soluciones diversas, que serán utilizadas para crear el Conjunto de Referencia sobre el que se trabaja. De ese conjunto se seleccionarán una serie de soluciones para realizar las combinaciones que dan lugar a las nuevas soluciones, las cuales serán evaluadas de acuerdo a una función objetivo para determinar si mejoran a algunas de las presentes en el Conjunto de Referencia. En caso afirmativo el Conjunto de Referencia será actualizado, incluyendo esas nuevas soluciones y eliminando las peores. De esta forma la solución final al problema planteado estará almacenada en el Conjunto de Referencia. El algoritmo de Búsqueda Dispersa finaliza su ejecución cuando dicho conjunto no es actualizado.

3 La búsqueda de datos de prueba mediante Búsqueda Dispersa

En esta sección se describe la aplicación de la técnica metaheurística Búsqueda Dispersa a la generación automática de datos de prueba y su combinación con un método de Búsqueda Local. Esta combinación ha dado lugar al algoritmo generador de datos de prueba denominado TCSS-LS.

3.1 Planteamiento del problema

Durante el proceso de búsqueda de datos de prueba bajo el criterio de suficiencia de cobertura de ramas, TCSS-LS necesita conocer las ramas que han sido cubiertas por los datos de prueba

previamente generados y las que aún no lo han sido. Además necesita almacenar una serie de información que le sirva de base para la generación de nuevos datos de prueba que permitan aumentar la cobertura de ramas. Para ello utiliza el grafo de control de flujo asociado al programa software bajo prueba (SUT: Software Under Test), donde cada requisito de prueba a cumplir se representa mediante un nodo, es decir, cada evaluación cierta o falsa de una decisión del SUT dará lugar a un nodo. Con este grafo es posible determinar qué ramas han sido cubiertas debido a que el SUT es instrumentado para recordar el camino seguido por cada dato de prueba ejecutado en él.

Mediante el uso del grafo de control de flujo, el objetivo general de obtener datos de prueba que cubran todas las ramas del SUT se puede dividir en subobjetivos, consistiendo cada uno de ellos en encontrar datos de prueba que alcancen un determinado nodo del grafo de control de flujo.

Para alcanzar dichos subobjetivos, los nodos almacenan determinada información durante todo el proceso de generación de datos de prueba, la cual permite mantener el conocimiento de las ramas cubiertas y es utilizada para avanzar en el proceso de búsqueda. Cada nodo almacena esta información en un conjunto de soluciones, llamado Conjunto de Referencia o RefSet. A diferencia del algoritmo de Búsqueda Dispersa general, TCSS-LS trabajan con varios Conjuntos de Referencia. Cada uno de estos conjuntos se denomina S_k (donde k es el número de nodo) y contiene varios elementos $T_k^c = < \overline{x}_k^c, p_k^c, fb_k^c, fc_k^c > c \in \{1, 2, \ldots, B_k\}$, donde $\overline{x}_k^c$ es una solución (un dato de prueba) que alcanza el nodo k y está formada por los valores de las variables de entrada que hacen ciertas las decisiones de los nodos del camino que llega al nodo k , p_k^c es el camino recorrido por la solución, fb_k^c es la distancia que indica lo cerca que dicha solución está de pasar por su nodo hermano y fc_k^c es la distancia que indica lo cerca que está la solución de pasar por el nodo hijo que no ha sido alcanzado por dicha solución.

Las distancias son calculadas utilizando las decisiones de entrada a los nodos no alcanzados durante la ejecución de la solución en el SUT, es decir, decisiones que se evalúan a falso. Las funciones empleadas para su cálculo pueden consultarse en [11].

La estructura del grafo de control de flujo del SUT con la información almacenada en los Conjuntos de Referencia S_k puede verse representada en la Fig. 1.

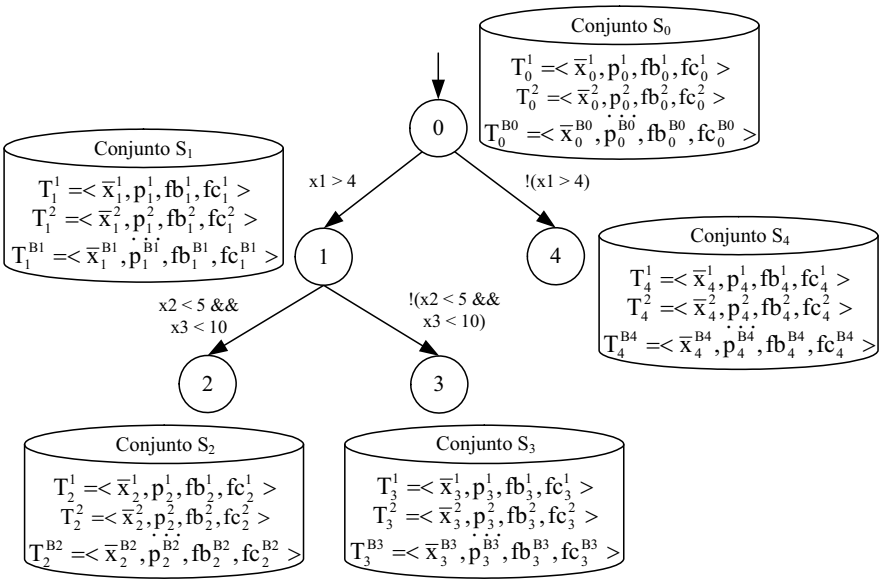

Fig. 1. Grafo de Control de Flujo de TCSS-LS

El conjunto de soluciones de un nodo n_k (S_k) tiene un tamaño máximo B_k. Este tamaño es diferente para cada nodo n_k y depende de la complejidad del código fuente situado por debajo del nodo n_k, de forma que las nuevas soluciones construidas a partir de las almacenadas en el conjunto

S_k puedan alcanzar los nodos que representan dicho código fuente. La forma en la que se determina cada tamaño B_k se puede consultar en [5].

TCSS-LS tratará de hacer los conjuntos S_k lo más diversos posibles para generar soluciones que puedan cubrir distintas ramas del programa.

3.2 Proceso de búsqueda

Como se ha comentado previamente, el objetivo de TCSS-LS es obtener la máxima cobertura de ramas, por lo que deben encontrarse soluciones (datos de prueba) que permitan cubrir todos los nodos del grafo de control de flujo. Puesto que dichas soluciones se almacenan en los nodos, el objetivo de TCSS-LS es, por tanto, que todos los nodos tengan al menos un elemento en su conjunto S_k. Sin embargo, este objetivo no puede ser alcanzado cuando el SUT posee ramas inalcanzables. Por ese motivo TCSS-LS también finaliza la búsqueda tras la generación de un número máximo de datos de prueba. Inicialmente los conjuntos de soluciones S_k están vacíos y serán rellenados en las sucesivas iteraciones.

En la Fig. 2 se muestra gráficamente el esquema del proceso de búsqueda de TCSS-LS. El proceso de búsqueda comienza con la generación de las soluciones aleatorias que serán almacenadas en el Conjunto de Referencia del nodo raíz (S_0), el cual actúa como el conjunto P del algoritmo estándar de Búsqueda Dispersa. Cada solución es ejecutada sobre el SUT, lo que da lugar al recorrido de un cierto camino. A continuación los Conjuntos de Referencia S_k (o RefSet) de los nodos pertenecientes al camino recorrido serán actualizados con aquellas soluciones que los alcancen, por lo que dichas soluciones además de ser diversas presentan un cierto grado de calidad. A partir de este momento comienzan las iteraciones del proceso de búsqueda, en las cuales se debe seleccionar un nodo del grafo de control de flujo (nodo en evaluación) para crear los subconjuntos de soluciones de su Conjunto de Referencia, que serán utilizados por las reglas de combinación para generar las nuevas soluciones. Estas nuevas soluciones, una vez han pasado por un proceso de mejora, son también ejecutadas sobre el SUT para realizar la posterior actualización de los Conjuntos de Referencia S_k de los nodos alcanzados, cerrando así el ciclo de ejecución.

Cuando el Conjunto de referencia del nodo seleccionado no posee suficientes soluciones para continuar con el proceso de búsqueda es necesario realizar un proceso de backtracking para encontrar otro conjunto más apropiado con el que trabajar. Si el proceso de backtracking llega a su fin, sin conseguir el objetivo perseguido, se lleva a cabo una regeneración de los Conjuntos de Referencia que se han visto afectados por dicho backtracking.

El final del proceso de búsqueda viene determinado por el cumplimiento de alguno de los criterios anteriormente expuestos: todos los nodos has sido alcanzados o se ha generado un número máximo de datos de prueba permitido.

La solución final de TCSS-LS está compuesta por los datos de prueba que cubren las ramas del SUT, los cuales están almacenados en los conjuntos S_k de los nodos del grafo de control de flujo, así como el porcentaje de cobertura de ramas alcanzado y el tiempo consumido en el proceso de búsqueda.

Los criterios utilizados en la selección del nodo del grafo de control empleado en cada iteración, la formación de los subconjuntos de soluciones a partir de un conjunto S_k, las reglas de combinación empleada para generar las nuevas soluciones, el proceso de mejora y los criterios de actualización de los conjuntos S_k pueden consultarse en [5][6].

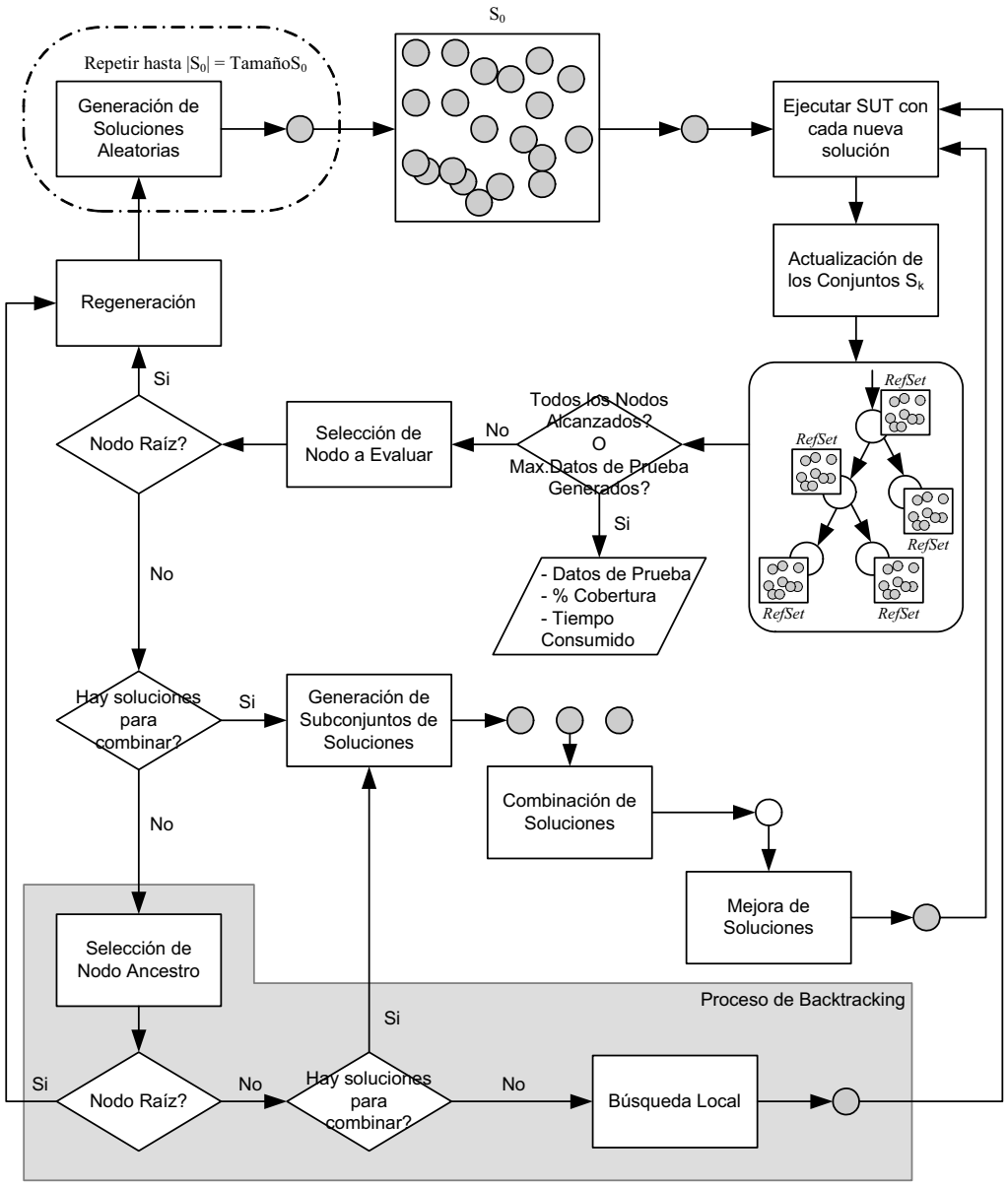

Fig. 2. Esquema de funcionamiento de TCSS-LS

3.3 El proceso de backtracking

La mayor dificultad del proceso de búsqueda surge cuando el nodo n_k en evaluación no posee al menos dos soluciones en su conjunto S_k que permitan realizar las combinaciones que dan lugar a las nuevas soluciones. Esto sucede cuando no existe ningún dato de prueba que cubra el nodo n_k (o sólo uno fue capaz de alcanzarlo) y además todos los demás candidatos están en la misma situación. Por tanto TCSS-LS no puede avanzar en el proceso de búsqueda y en consecuencia se debe realizar un proceso de backtracking. Este proceso trata de incrementar el tamaño del conjunto S_k del nodo n_k por medio de la generación de nuevas soluciones utilizando el nodo padre.

Para generar las nuevas soluciones durante el proceso de backtracking TCSS-LS tiene dos opciones:

a. Realizar combinaciones: esta opción se lleva a cabo cuando el nodo padre posee en su conjunto S_k soluciones que no han sido utilizadas en combinaciones previas. En este caso, las combinaciones se realizan utilizando dichas soluciones.
b. Utilizar un método de búsqueda local: esta opción se efectúa cuando el nodo padre ya ha utilizado todas sus soluciones en combinaciones previas. De este modo, TCSS-LS lleva a cabo un proceso de intensificación para intentar cubrir los nodos que no han sido alcanzados en el proceso de diversificación.

El SUT instrumentado es ejecutado con cada nueva solución y los conjuntos S_k de los nodos alcanzados durante dicha ejecución se actualizan. Si no se consigue generar ninguna solución en el nodo padre que permita llegar al nodo hijo, se deberá realizar nuevamente un backtracking para intentar generar soluciones en el nodo abuelo, llegando si fuera necesario hasta el nodo raíz. Si el backtracking llega al nodo raíz TCSS-LS lleva a cabo un proceso de regeneración puesto que las soluciones almacenadas hasta el momento no permitieron obtener la cobertura total de ramas. Durante dicha regeneración se limpian los conjuntos S_k afectados por el backtracking, pero se recuerda una de sus soluciones para que TCSS-LS no tenga que realizar un esfuerzo adicional buscando soluciones que lleguen a un nodo que previamente había sido alcanzado.

3.4 El proceso de Búsqueda Local

El método de Búsqueda Local trata de generar soluciones que cubran un determinado nodo n_k que provocó el backtracking. Para ello selecciona una solución del conjunto S_k del nodo ancestro seleccionado en cada iteración del backtracking (padre, abuelo, …), denominada solución original ($\overline{os}$), y lleva a cabo una serie de iteraciones a partir de ella para alcanzar el nodo n_k. La solución $\overline{os}$ seleccionada es aquella que posee el menor valor de distancia al nodo hijo n_k que se pretende alcanzar (fc_i^c), intentando así guiar la búsqueda de soluciones que cubran dicho nodo. El número de iteraciones a realizar a partir de una $\overline{os}$ tiene un límite máximo (MAX_INTENTOS) para evitar que se produzca un estancamiento de la búsqueda por el uso de una solución de que no da buen resultado.

En cada iteración del proceso de Búsqueda Local se generan 2n nuevas soluciones (donde n es el número de variables de entrada de la solución) a partir de la mejor solución de la iteración anterior, denominada solución actual ($\overline{cs}$). Las nuevas soluciones se diferencian de $\overline{cs}$ en el valor de una sola variable de entrada y se obtienen de la siguiente forma:

- $\overline{ns1} = \overline{cs}^i + \delta_i$
- $\overline{ns2} = \overline{cs}^i - \delta_i$

donde el índice *i* recorre todas las variables de entrada y δ_i es definida de la siguiente forma:

$$\delta_i = \begin{cases} \dfrac{fc_{cs}}{reductor\_salto} & \text{si} \begin{cases} \text{TCSS - LS calcula } \overline{ns1} \text{ y } \delta_i \le max\_saltoderecho \\ \text{or TCSS - LS calcula } \overline{ns2} \text{ y } \delta_i \le max\_saltoizquierdo \end{cases} \\ \dfrac{max\_saltoderecho}{reductor\_salto} \cdot Ln\left(\dfrac{fc_{cs}\cdot(e-1)}{fc_{os}}+1\right) & \text{si TCSS - LS calcula } \overline{ns1} \text{ y } \delta_i > max\_saltoderecho \\ \dfrac{max\_saltoizquierdo}{reductor\_salto} \cdot Ln\left(\dfrac{fc_{cs}\cdot(e-1)}{fc_{os}}+1\right) & \text{si TCSS - LS calcula } \overline{ns2} \text{ y } \delta_i > max\_saltoizquierdo \end{cases}$$

donde *reductor_salto* es un parámetro que se incrementa para reducir el salto y los saltos máximos se definen como:

- max_saltoderecho = Rango Superior Variable i – $\overline{cs}^i$
- max_saltoizquierdo = $\overline{cs}^i$ – Rango Inferior Variable i

Antes de calcular las nuevas soluciones se deben determinar los saltos máximos que cada variable puede dar sin producir un desbordamiento del rango Así *max_saltoderecho* es el mayor salto que se puede sumar a una variable y *max_saltoizquierdo* es el mayor salto que se puede restar a una variable.

El salto δ_i se calcula a partir de la distancia al nodo hijo que se pretende alcanzar (fc). Esa distancia es dividida por un reductor que permite controlar la amplitud del salto. Si se está calculando $\overline{ns1}$, el salto no debe ser mayor que *max_saltoderecho*. En caso contrario el salto se recalcula teniendo en cuenta el salto máximo y un ratio obtenido en función de los valores de la distancia al nodo hijo de la solución original (fc_{os}) y de la solución actual (fc_{cs}). Este ratio se encuentra situado entre 0 y 1, por lo que suaviza el salto generado. Si se está calculando $\overline{ns2}$, el salto no debe ser mayor que *max_saltoizquierdo* y en caso contrario se sigue el proceso descrito para $\overline{ns1}$.

Posteriormente TCSS-LS ejecuta el SUT con las nuevas soluciones. Estas soluciones son evaluadas para determinar si alguna de ellas alcanza el nodo ancestro y disminuye el valor de la distancia fc de $\overline{cs}$, de modo que se utilice la mejor solución como nueva solución actual. Si ninguna solución mejora $\overline{cs}$ y la Búsqueda Local no ha llevado a cabo MAX_INTENTOS iteraciones a partir de la solución original $\overline{os}$, se reduce el salto utilizado en δi para generar soluciones más cercanas a $\overline{cs}$ en la próxima iteración. Si la Búsqueda Local ha realizado MAX_INTENTOS iteraciones a partir de $\overline{os}$ y el nodo hijo n_k no ha sido alcanzado, el método selecciona otra solución ($\overline{os}$) del conjunto S_k del nodo ancestro para efectuar la búsqueda. Cuando todas las soluciones del nodo ancestro han sido utilizadas por el método de Búsqueda Local, TCSS-LS realiza un backtracking y utiliza las soluciones de otro ancestro para alcanzar el nodo n_k. El proceso de Búsqueda Local finaliza cuando el nodo n_k se cubre o cuando todas las soluciones del nodo ancestro han sido utilizadas en la búsqueda.

4 Resultados

En esta sección se presenta la experimentación realizada con TCSS-LS y con el generador basado únicamente en Búsqueda Dispersa denominado TCSS [5][6]. Para ello se han empleando los benchmarks que se muestran en la

Tabla **1**. En esta tabla figura para cada benchmark su nombre y la abreviatura que lo representa, el número de ramas que contiene, su máximo nivel de anidamiento, su complejidad ciclomática y la referencia desde la cual fueron obtenidos.

Para llevar a cabo los experimentos se definen una serie de instancias, que especifican el SUT empleado por TCSS-LS y TCSS para generar los datos de prueba bajo el criterio de cobertura de ramas. Una instancia representa un benchmark con un determinado tipo de variables de entrada (C para variables de tipo char, F para variables de tipo float e I para variables de tipo entero), las cuales toman valores dentro de un cierto rango (L para rangos de 8 bits, M para rangos de 16 bits y H para rangos de 32 bits), así por ejemplo una instancia estaría definida por el benchmark TrianguloMyers con variables de entrada de tipo entero y rango de entrada de 8 bits. TCSS-LS y TCSS realizan 10 ejecuciones con cada instancia, de forma que los resultados obtenidos (porcentaje de cobertura alcanzada, datos de prueba generados y tiempo consumido) se corresponden con los valores medios de esas ejecuciones.

En todos los experimentos el criterio de parada utilizado por TCSS-LS y TCSS consiste en alcanzar el 100% de cobertura de ramas o generar 200.000 datos de prueba. La máquina utilizada para llevar a cabo la experimentación es un Pentium 1.50GHz con 512 MB de memoria RAM

Tabla 1. Lista de benchmarks empleados en la experimentación

Benchmark	Abr.	Nº de ramas	Nivel de Anidamiento	Complejidad Ciclomática	Referencia
Atof	AF	30	2	17	[34]
BisectionMethod	BM	8	3	5	[26]
ComplexBranch	CB	24	3	14	[34]
CalDay	CD	22	2	12	[2]
LineRectangle	LR	36	12	19	[11]
NumberDays	ND	86	10	44	[11]
QuadraticFormula	QF	4	2	3	[26]
QuadraticFormulaSthamer	QFS	6	3	4	[29]
RemainderSthamer	RS	18	5	10	[29]
TriangleMyers	TM	12	5	7	[25]
TriangleMichael	TMM	20	6	11	[23]
TriangleSthamer	TS	26	12	14	[29]
TriangleWegener	TW	26	3	14	[34]

En la Tabla 2 se resumen los resultados obtenidos por TCSS y TCSS-LS. Para cada instancia se muestra el porcentaje de cobertura alcanzada, el número de datos de prueba generados para lograr dicha cobertura y el tiempo consumido (en segundos). Para llevar a cabo una comparación correcta el número de datos de prueba creados y el tiempo consumido debe ser comparado cuando ambos generadores obtienen el mismo porcentaje de cobertura. Por este motivo cuando TCSS-LS alcanza mayor cobertura que TCSS se muestra entre paréntesis el número de datos de prueba generados y en tiempo consumido por TCSS-LS para alcanzar la misma cobertura obtenida por TCSS.

Los resultados obtenidos por ambos generadores indican que TCSS-LS siempre alcanza, al menos, la cobertura lograda por TCSS. De las 40 instancias TCSS alcanza el 100% de cobertura en 24 de ellas, mientras que TCSS-LS mejora la cobertura obtenida por TCSS al alcanzar el 100% en 33 instancias. En las 7 instancias restantes donde ambos generadores no alcanzan el 100% de cobertura, TCSS-LS incrementa la cobertura obtenida por TCSS en 4 de ellas.

Respecto al número de datos de prueba generados y al tiempo consumido para alcanzar la misma cobertura (27 instancias), TCSS-LS necesita menos datos de prueba en 22 instancias, en las cuales también consume menos tiempo (instancias 3, 6, 7, 15-22, 24-29, 32, 34-37). Sólo en 4 instancias TCSS-LS genera mayor número de datos de prueba y consume más tiempo que TCSS (instancias 9-11, 31).

De las 13 instancias en las que la cobertura alcanzada por TCSS es incrementada por TCSS-LS, éste genera menos datos de prueba para obtener mayor cobertura en 7 de ellas (instancias 1, 2, 4, 5, 12, 30, 33), consumiendo además menos tiempo en todas excepto en una (instancia 12). En las 6 instancias restantes TCSS-LS genera más datos de prueba para incrementar la cobertura, pero en 4 de ellas requiere menos datos de prueba y consume menos tiempo para obtener la máxima cobertura alcanzada por TCSS (instancias 8, 13, 14, 39). En las otras dos instancias los resultados de ambos generadores para obtener la máxima cobertura lograda por TCSS son iguales (instancias 38, 40).

Por otro lado, TCSS-LS obtiene mejores resultados (mayor cobertura, menor número de datos de prueba y menor tiempo) que TCSS en todos los benchmarks excepto en LR, donde obtiene peores resultados en los tres rangos enteros, y en TMM, donde obtiene peores resultados en el rango menor.

La mejora aportada por TCSS-LS también se puede ver analizando las diferencias entre los datos de prueba creados por cada generador, siendo éstas mayores cuando TCSS-LS genera menos datos de prueba que TCSS (por ejemplo, en la instancia 17) y son menores en caso contrario (por ejemplo, en la instancia 9). Por otro lado, cuando el rango de las variables de entrada se incrementa las diferencias también aumentan, por lo que la mejora aportada por TCSS-LS es mayor con rangos grandes.

Tabla 2. Resultados obtenidos por TCSS-LS y TCSS

Instancia	Benchmark	Tipo	Rango	Resultados para TCSS			Resultados para TCSS-LS		
				% Cob.	Datos Prueba	Tiempo (seg)	% Cob.	Datos Prueba	Tiempo (seg)
1	AF	C	L	97,19	101144	141,11	100,00	17133 (12167)	8,28 (4,69)
2	BM	F	H	88,00	52679	39,82	100,00	233 (194)	0,22 (0,22)
3	CB	I	L	100,00	6206	34,24	100,00	4880	19,67
4	CB	I	M	91,92	121223	608,09	100,00	5384 (3838)	15,18 (10,80)
5	CB	I	H	74,23	31210	41,21	100,00	26750 (472)	39,37 (0,82)
6	CD	I	M	100,00	38074	8,82	100,00	558	0,58
7	CD	I	H	100,00	39209	9,07	100,00	561	0,68
8	LR	F	H	97,49	4838	2,29	100,00	5939 (2601)	2,07 (1,21)
9	LR	I	L	100,00	1985	1,18	100,00	4213	1,37
10	LR	I	M	100,00	1580	1,03	100,00	3538	1,25
11	LR	I	H	100,00	2188	1,22	100,00	5817	1,69
12	ND	I	L	99,79	98173	193,19	100,00	76271 (76271)	269,13 (269,13)
13	ND	I	M	6,17	42410	17,91	100,00	115379 (2992)	361,66 (1,32)
14	ND	I	H	2,87	100912	43,32	99,04	148880 (1298)	164,52 (0,56)
15	QF	F	H	100,00	25839	3,75	100,00	50	0,02
16	QF	I	L	100,00	110	0,03	100,00	48	0,02
17	QF	I	M	100,00	20537	3,31	100,00	50	0,02
18	QF	I	H	100,00	26234	3,79	100,00	50	0,02
19	QFS	F	H	100,00	25159	4,25	100,00	2977	0,36
20	QFS	I	L	100,00	1176	0,31	100,00	1096	0,15
21	QFS	I	M	100,00	25052	4,24	100,00	2357	0,28
22	QFS	I	H	100,00	23838	4,06	100,00	1958	0,22
23	RS	I	L	100,00	100	0,09	100,00	100	0,09
24	RS	I	M	100,00	27676	4,79	100,00	482	0,28
25	RS	I	H	100,00	27743	4,82	100,00	484	0,28
26	TM	F	H	100,00	806	0,25	100,00	405	0,15
27	TM	I	L	100,00	367	0,16	100,00	260	0,13
28	TM	I	M	100,00	880	0,26	100,00	370	0,15
29	TM	I	H	100,00	951	0,22	100,00	607	0,15
30	TMM	F	H	96,00	9995	6,16	100,00	4518 (624)	2,35 (0,75)
31	TMM	I	L	100,00	1028	0,76	100,00	1568	0,79
32	TMM	I	M	100	26431	15,20	100,00	5147	2,64
33	TMM	I	H	92,00	117416	73,56	100,00	29419 (16395)	9,64 (5,22)
34	TS	F	H	88,46	1307	1,03	88,46	1121	0,90
35	TS	I	L	100,00	20028	8,83	100,00	18161	4,47
36	TS	I	M	88,46	1162	0,92	88,46	943	0,68
37	TS	I	H	88,46	1527	0,95	88,46	1046	0,77
38	TW	F	H	11,53	4	0,01	96,15	3692 (4)	2,45 (0,01)
39	TW	I	M	92,30	3561	2,25	96,92	17486 (1531)	7,84(1,56)
40	TW	I	H	11,53	4	0,01	96,15	3692 (4)	2,45 (0,01)

Para comprobar que las diferencias observadas entre TCSS-LS y TCSS son significativas se realiza un análisis estadístico con α=0,05. Las hipótesis a verificar son las siguientes:

- Hipótesis H1: El número de datos de prueba generados por TCSS-LS es significativamente menor que los datos de prueba generados por TCSS.
- Hipótesis H2: El tiempo consumido por TCSS-LS es significativamente menor que el tiempo consumido por TCSS
- Hipótesis H3: El número de veces que TCSS-LS obtiene mejores resultados que TCSS es significativo.

Las pruebas que permiten verificar la hipótesis H1 (prueba t para muestras pareadas o la prueba de los signos de Wilcoxon para muestras pareadas) dependen de la normalidad de la distribución. En este caso se trata de la normalidad de la variable $D_{DatosPrueba}$=$DatosPrueba_{TCSS-LS}$-$DatosPrueba_{TCSS}$. Por ello, en primer lugar se realiza la prueba de Kolmogorov-Smirnov. El p-value obtenido es muy pequeño (< 0,001) y no se puede asumir que siga una distribución normal.

Por tanto se aplica la prueba de los signos de Wilcoxon para muestras pareadas para verificar la hipótesis nula sobre la igualdad de medianas (H_0:m_D=0, H_1:m_D<0). El p-value obtenido en el análisis es más pequeño que 0,001<α y en consecuencia la hipótesis H_0:m_D=0 puede ser rechazada. Además la media de los datos de prueba generados por TCSS-LS es menor que la media de los datos de prueba generados por TCSS. Así pues se puede asumir que TCSS-LS genera menos datos de prueba que TCSS.

Para verificar la hipótesis H2 también es necesario comprobar la normalidad de la variable D_{Tiempo}=$Tiempo_{TCSS-LS}$-$Tiempo_{TCSS}$ para determinar la prueba a realizar. El p-value obtenido nuevamente es menor que 0,001 y no se puede asumir que siga una distribución normal.

Al aplicar la prueba de los signos de Wilcoxon para muestras pareadas, la hipótesis H_0:m_D=0 puede volver a ser rechazada ya que el p-value<0,001<α. Además la media del tiempo consumido por TCSS-LS es también menor que la media del tiempo consumido por TCSS. Por lo tanto se puede asumir que TCSS-LS consume menos tiempo que TCSS.

Para verificar la hipótesis H3 se emplea la prueba de McNemar que trata de validar la hipótesis nula que indica que los sujetos confrontados tienen la misma probabilidad de derrotar al otro, es decir,

$$H_0:\ \mathrm{Prob}\left[\frac{\text{vecesmejor}}{\text{vecesmejor}+\text{vecespeor}}\right] = \mathrm{Prob}\left[\frac{\text{vecespeor}}{\text{vecesmejor}+\text{vecespeor}}\right]$$

Para aplicar la prueba de McNemar se considera que un generador es mejor que otro cuando alcanza mayor porcentaje de cobertura o cuando crea menos datos de prueba si ambos generadores alcanzan el mismo porcentaje de cobertura.

En la Tabla 2 se puede observar que TCSS-LS obtiene mejores resultados que TCSS en 35 instancias y obtiene peores resultados en 4 instancias. Ambos generadores obtienen el mismo resultado en una instancia. El p-value obtenido es menor que 0,0001<α, por lo que la diferencia entre ambos generadores es significativa y TCSS-LS obtiene estadísticamente mejores resultados que TCSS.

4 Conclusiones

En este artículo se ha presentado el generador automático de datos de prueba TCSS-LS, que está basado en la técnica metaheurística Búsqueda Dispersa. TCSS-LS también utiliza un procedimiento de búsqueda local para intensificar la búsqueda de datos de prueba en determinados puntos. Este generador utiliza el grafo de control de flujo asociado al programa bajo prueba, el cual permite guiar el proceso de búsqueda, ya que almacena en los Conjuntos de Referencia que cada nodo posee tanto datos de prueba como el conocimiento adquirido durante dicho proceso.

Los resultados de los experimentos muestran que TCSS-LS consigue aumentar el porcentaje de cobertura obtenido por TCSS, y genera además menos datos de prueba y consume menos tiempo,

siendo la mejora aportada estadísticamente significativa. Así mismo, el análisis de los resultados indica que la combinación de varias técnicas, como la Búsqueda Dispersa y la Búsqueda Local, mejora la eficiencia de la generación de datos de prueba, puesto que se pueden aprovechar los principales beneficios de cada una de ellas.

A la vista de los resultados obtenidos, una línea de trabajo consiste en combinar la Búsqueda Dispersa con otra técnica que trabaje con una búsqueda local más especializada, la Búsqueda Tabú, con el fin de desbloquear más rápidamente aquellos puntos en los que la Búsqueda Dispersa encuentra dificultades. Otras líneas de trabajo futuras consisten en emplear TCSS-LS con otros criterios de suficiencia, como cobertura de caminos o MC/DC, así como la aplicación de la Búsqueda Dispersa a la generación de datos de prueba para sentencias de acceso a bases de datos y composiciones de servicios web.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Educación y Ciencia y los fondos FEDER dentro del Plan Nacional de I+D+I, Proyectos INT2TEST (TIN2004-06689-C03-02), Test4SOA (TIN2007-67843-C06-01) y RePRIS (TIN2005-24792-E, TIN2007-30391-E).

Referencias

1. Ahmed, M.A., Hermadi, I.: GA-based multiple paths test data generator. Computers and Operations Research 35(10), 3107-3124 (2008).
2. Alba, E., Chicano, F.: Observations in using parallel and sequential evolutionary algorithms for automatic software testing. Computers and Operations Research 35(10), 3161-3183 (2008).
3. Alshraideh, M., Bottaci, L.: Search-based software test data generation for string data using program-specific search operators. Software Testing Verification and Reliability 16(3), 175-203 (2006).
4. Beizer, B.: Software testing techniques. Second edition, Van Nostrand Reinhold (1990).
5. Blanco, R., Díaz, E., Tuya, J.: Algoritmo Scatter Search para la generación automática de pruebas de cobertura de ramas. In: Actas de IX Jornadas de Ingeniería del Software y Bases de Datos, pp. 375-386 (2004).
6. Blanco, R., Díaz, E., Tuya, J.: Generación automática de casos de prueba mediante Búsqueda Dispersa. Revista Española de Innovación, Calidad e Ingeniería del Software 2(1), 24-35 (2006).
7. Bueno, P.M.S., Jino, M.: Automatic test data generation for program paths using Genetic Algorithms. International Journal of Software Engineering and Knowledge Engineering 12(6), 691-709 (2002).
8. Bueno, P.M.S., Wong, W.E., Jino, M.: Improving random test sets using the diversity oriented test data generation. In: Proceedings of the Second International Workshop on Random Testing, pp. 10-17 (2007).
9. Bühler, O., Wegener, J.: Evolutionary functional testing. Computers and Operational Research 35(10), 3144-3160 (2008).
10. Clarke, J., Dolado, J.J., Harman, M., Hierons, R.M., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., Shepperd, M.: Reformulating software engineering as a search problem. IEE Proceedings – Software 150(3), 161-175 (2003).
11. Díaz, E., Tuya, J., Blanco, R., Dolado, J.J.: A tabu search algorithm for Software Testing. Computers and Operational Research 35(10), 3052-3072 (2008).
12. Ferguson, R., Korel, B.: The Chaining Approach for Software Test Data Generation. ACM Transactions on Software Engineering and Methodology 5(1), 63-86 (1996).
13. Girgis, M.R.: Automatic test data generation for data flow testing using a genetic algorithm. Journal of Universal Computer Science 11(6), 898-915 (2005).
14. Glover, F.: A template for Scatter Search and Path Relinking, Artificial Evolution, Lecture Notes in Computer Science 1363, Springer-Verlag, pp. 13-54 (1998).
15. Hartman, A.: Is ISSTA Research Relevant to Industry? ACM SIGSOFT Software Engineering Notes 27(4), 205-206 (2002).
16. Jones, B.F., Eyres, D.E., Sthamer, H.H.: A strategy for using Genetic Algorithms to automate branch and fault-based testing. The Computer Journal 41(2), 98-107 (1998).

17. Laguna, M., Martí, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers, Boston, MA, USA (2002).
18. Lin, J., Yeh, P.: Automatic test data generation for path testing using Gas. Information Sciences 131 (1-4) 47-64 (2001).
19. Mansour, N., Salame, M.: Data generation for path testing. Software Quality Journal 12, 121-136 (2004).
20. Mantere,T., Alander, J.T.: Evolutionary software engineering, a review. Applied Soft Computing 5(3), 315-331 (2005).
21. McMinn, P.: Search-based software test data generation: a survey. Software Testing Verification and Reliability 14(2), 105-156 (2004).
22. McMinn, P., Holcombe, M.: Evolutionary testing using an extended chaining approach. Evolutionary Computation 14(1), 41.64 (2006).
23. Michael, C., McGraw, G., Schatz, M.: Generating software test data by evolution. IEEE Transactions on Software Engineering 27(12), 1085-1110 (2001).
24. Miller, J., Reformat, M., Zhang, H.: Automatic test data generation using genetic algorithm and program dependence graphs. Information and Software Technology 48, 586-605 (2006).
25. Myers, G.: The art of software testing, Ed. John Wiley & Sons (1979).
26. Offut, A.J., Pan, J., Tewary, K., Zhang, T.: Experiments with Data Flow and Mutation Testing. Technical Report ISSE-TR-94-105, 1994.
27. Pargas, R.P., Harrold, M.J., Peck, R.R.: Test data generation using genetic algorithms. Journal of Software Testing Verification and Reliability 9, 263-282 (1999).
28. Sagarna, R., Lozano, J.A.: Scatter Search in software testing, comparison and collaboration with Estimation of Distribution Algorithms, European Journal of Operational Research 169, 392-412 (2006).
29. Sthamer, H.H.: The automatic generation of software test data using genetic algorithms. PhD Thesis, University of Glamorgan (1996).
30. Tracey, N., Clark, J., Mander, K.: Automated program flaw finding using simulated annealing. In: Proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 73-81 (1998).
31. Vergilio, S.R., Pozo, A.: A grammar-guided genetic programming framework configured for data mining and software testing. International Journal of Software Engineering and Knowledge Engineering 16(2), 245-267 (2006).
32. Waeselynck, H., Thévenod-Fosse,P., Abdellatif-Kaddour, O.: Simulated annealing applied to test generation: landscape characterization and stopping criteria. Empirical Software Engineering 12(1), 35-63 (2007).
33. Watkins, A., Hufnagel, E.M.: Evolutionary test data generation: a comparison of fitness functions. Software Practice and Experience 36, 95-116 (2006).
34. Wegener, J., Baresel, A., Sthamer, H.: Evolutionary test environment for automatic structural testing, Information & Software Technology 43(14), 841-854 (2001).
35. Xiao, M., El-Attar, M., Reformat, M., Miller, J.: Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques. Empirical Software Engineering 12(2), 183-239 (2007).

Diagnosing Business Processes Execution using Choreography Analysis

Diana Borrego, María Teresa Gómez-López, Rafael M. Gasca and Irene Barba

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Sevilla, Spain
{dianabn,maytegomez,gasca,irenebr}@us.es

Abstract. This work presents a proposal to diagnose business processes that form a global process using a choreography analysis. The diagnosis is based on distributed diagnosis since the business process is formed by a process orchestrations modelled by a set of activities. These business processes have two different types of activities, with internal and external interaction. In this paper the knowledge of the whole business process is divided in different processes. In means that each user has a local point of view of the information of the organization, it also happens in distributed system, where neither agent has global information of how the system is modelled. This work propose a methodology to diagnose the business processes, analyzing only the interactions between the activities of different processes. In order to perform the fault detection for business processes, an algorithm has been defined based on distributed diagnosis. Also some definitions about model-based diagnosis have been redefined to be adapted to business processes diagnosis.

Keywords: Choreography Analysis, Business Processes Diagnosis, Model-Based Diagnosis.

1 Introduction

Currently the business processes can be composed of different subprocess and a large number of activities that interact by means of a choreography with the same process or another.

Fault diagnosis permits to determine why a business process correctly designed does not work as it is expected. The diagnosis aim is to detect and to identify the reason of an unexpected behavior, or in other words, to identify the parts which fail in a process orchestration. Our proposal is based on DX community approaches [12], [9]. These works were proposed to find out the discrepancies between the observed and correct behavior of a system. In this paper we adapt the DX methodology to business processes fault detection. In the business processes, each user has a local point of view

The traditional diagnostic tools can be considered as a single diagnostic agent with a model of the whole system to be diagnosed, however, in some systems a single agent approach is not desirable. Moreover the integration of knowledge into one model of the system is infeasible if the system is too large, dynamic or distributed over different local entities. In some systems, the knowledge integration can proceed from different local diagnostic processes situated in different nodes (it is called spatially distributed) or from different fields of expertise (it is called semantically distributed [6]). When a business process is performed by different users, the full knowledge is unknown since is divided into different process, and the behavior of the process cannot be studied is a global way. Thereby, the business process diagnosis can be compared to the distributed diagnosis.

In previous works, the fault diagnosis for systems is classified as follows:

– **Centralized approach**. In general, traditional model-based diagnosis is centralized, and the diagnostic algorithm is run on an only system. This system captures all the

observations in order to perform the global diagnosis. Relating it to business process, it can be the diagnosis of a business process where all the activities that form the process are known.

- **Decentralized approach**. In order to obtain a global diagnosis of a system, a central coordination process and a local diagnoser for each subsystem that form the whole system are required. Some examples were presented in [4][10][7], where local diagnosers are communicated to a coordination process obtaining a global diagnosis. This type of knowledge distribution can be compared with the orchestration of a set of process to achieve an objective, where the central agent is in charge of the combination of the processes.
- **Distributed approach**. This type of systems represents the aim of this paper. It uses communication by a set of local diagnosers instead of requiring a global coordination process such as in a decentralized approach. In the bibliography, there are several proposals where there is no centralized control structure or coordination process [1][5][11][13][14]. Each local diagnoser is communicated directly with other diagnosers. In these systems the model is distributed, the diagnosis is locally generated and the inter-component consistency should be satisfied. Our solution is centered here, where each process that forms the global business process counts on a local agent which is in charge of the communication and the local diagnosis. This implies that the different process performs a part of the choreography to obtain the objective of the full process.

As it has been commented for each type of distribution of systems, this type of problematic is similar to business processes diagnosis, where the orchestration of the different processes and the different activities form an unknown process in a global way [15]. In the case of business processes, each business process does not know the rest of the business processes that take part in the orchestration to achieve a common objective. Therefore, in the solution given in this paper each business process counts on a local diagnoser to carry out the diagnosis tasks. These local diagnosers will communicate between them to carry out the diagnosis based on external interactions. An important difference of our solution with respect to previous ones is that it is not necessary to carry out a complete monitorization of the business processes to know what activities are failing, so that we do not need to perform the measurement of all the properties related to a running workflow instance. A previous solution to this problem [8] applies chronicles recognition to monitor and diagnose the behavior of software components.

The possible mistakes in a business process are: *(a)* the creation of a model that does not correspond to the real problem, *(b)* one or more than one activities are not executed as it was modelled, and *(c)* errors in the definition of the interactions between the activities of a process, or between the activities of different processes.

In order to develop the business process diagnosis, an architecture and an algorithm are proposed to obtain a precompiled structure that can be monitorizated depending on the external interactions and makes possible an improvement of the temporal efficiency, since this structure is prepared offline.

This work is organized as follows: Section 2 presents concepts related to centralized and distributed diagnosis adapted to business process diagnosis. Section 3 shows the description of the model of Business process related to distributed approach, and presents a motivating example to explain our proposal. Section 4 presents the distributed algorithm using the previous example. Finally, some conclusions and future work are presented.

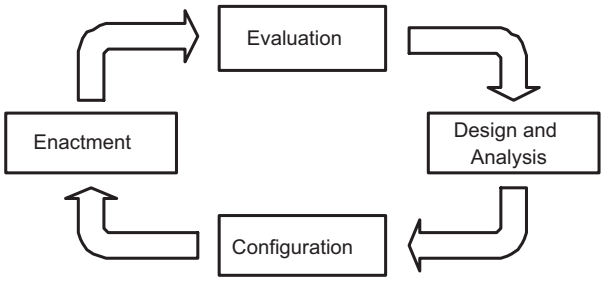

Fig. 1: Business process lifecycle.

2 Fault Diagnosis of Business Processes

Fault detection and identification of faulty activities of business processes are very important from the strategic point of view of the industries, since the economic demands and required environment conservation to remain in competitive markets.

This paper presents a new solution to diagnose business processes. In order to do that, the monitorization of a running workflow instance takes place. But this monitorization is only linked to some checkpoints, that must be determined previously. By means of this monitorization and the execution of a choreography analysis, an equivalent structure is obtained offline, which will be used to perform both local and global diagnosis, getting a high computational efficiency. This equivalent structure represents the relations among the different business processes and their external interactions, so that the local and global diagnosis can be performed without any information about the internal interactions of the different activities.

The diagnosis of the business processes will be carried out from information obtained from the checkpoints, indicating whether the events of the processes are correct or incorrect.

The business process lifecycle consists of four phases related to each other and organized in a cyclical way, where their dependencies can be detected. Fig. 1 represents the lifecycle of a business process. In the *Design and Analysis* phase, an analysis of the environment takes place, identifying needs and defining requirements, and finally designing and modelling the business process. In the *Configuration* phase, the business process is implemented, and the project preparation and realization is done. In the *Enactment* phase, an execution or deployment of the business process is carried out. Finally, in the *Evaluation* phase, the business process is monitored. The diagnosis process is included within the enactment phase, where after the execution of the different activities it is possible to detect an abnormal behavior of the choreography.

The kind of process choreography to diagnose S is formed by a finite set of business processes $\{BP_1, BP_2, ..., BP_n\}$, which are related among them using external interactions $\{s1, s2, ..., sp\}$. Also, the choreography has start events $\{i1, i2, ..., iq\}$ and end events $\{o1, o2, ..., or\}$. This implies that, in the business process under study there are three types of information: start and end events; internal interactions (only used within each business process) and external interactions (events among activities of different business processes). The external information is necessary to perform the local diagnosis.

Each business process (BP_i) is formed by a set of activities $(\{A_{i1}, A_{i2}, \ldots, A_{ik}\})$. The activities of a business process interchange incoming and outgoing interaction information. The incoming interaction information represent the inputs and the outgoing interaction information contains the results of the different activities. Both kind of interaction information are the internal and external interactions with activities in the same or in a different business process respectively.

In local and distributed diagnosis, the concepts of *Context Sets* and *Clusters* were defined [2], and in this paper these ideas are adapted for business process diagnosis:

Definition 2.1. *Context Set (CS)*: Any subset of activities of a business process. There are $2^{nactiv} - 1$ possible context sets, where *nactiv* is the number of activities of the business process.

Defining a business process as a directed graph, where nodes represent the different activities, and the directed edges are the internal and external interactions:

Definition 2.2. *Cluster (C)*: A connected component of the graph. The clusters of each business process will be taken into account in the local diagnosis process.

In our proposal, the process choreography (S) has an oracle which checks the end events, that are the checkpoints, finding out if they are correct (OK) or incorrect (KO).

Each business process involved in the process choreography has its own set of activities and requirements. Furthermore, each business process has associated an agent, which is in charge of the propagation and diagnosis tasks:

Definition 2.3. $SAgent_i$: Agent associated to BP_i that process incoming interaction information, performs local diagnosis and sends outgoing interaction information.

3 Motivating Example

In order to explain our proposal, we are going to use the example shown in Fig. 2, where there are four business processes, *(BP_1, BP_2, BP_3 and BP_4)*.

In the example, the different business processes with their respective information are:

- $\boldsymbol{BP}_1$ with the start event $\{i1\}$ and {s1, s2} as external interactions.
- $\boldsymbol{BP}_2$ with the start event $\{i2\}$ and {s1, s2, s3, s4, s5, s6, s7} as external interactions.
- $\boldsymbol{BP}_3$ with no start events and {s3, s4, s5, s6, s7, s8, s9, s10, s11} as external interactions.

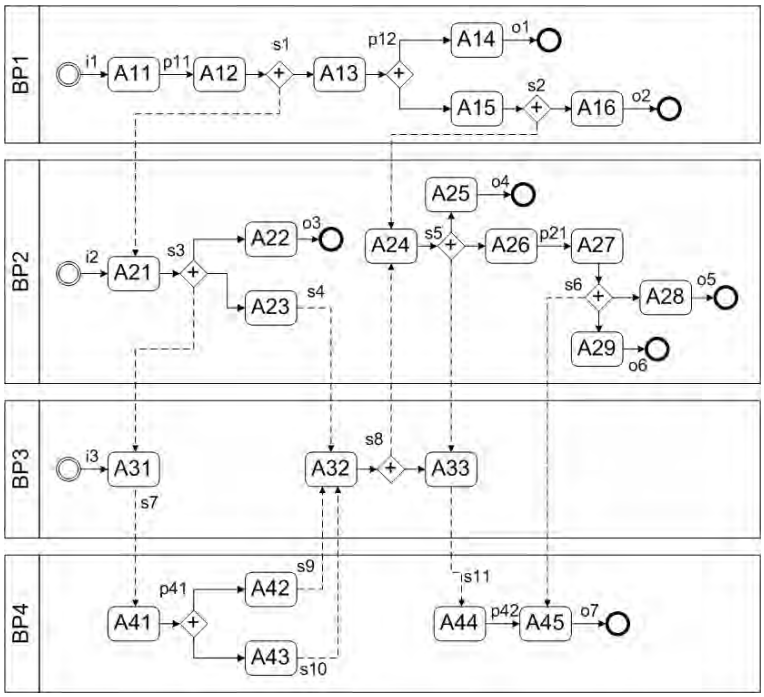

Fig. 2: Example of Business Processes.

– $\boldsymbol{BP_4}$ with the start event $\{i3\}$ and {s6, s7, s9, s10, s11} as external interactions.

The rest of the interactions in the processes and the activities are internal, and the are only known by the own process internally.

Each business process has a table where the relation between the non internal events and the rest of process (BP_1, BP_2, BP_3 and BP_4 for our example) is stored. That is, each business process knows how it is connected to other business processes by means of the different external interactions. Each business process does not know how the rest of the business processes are formed, only knows from what business processes have to receive incoming interaction information and towards what business processes have to send outgoing interaction information.

Once all the business processes know whether they are related to any incorrect output events, the diagnosis is solved locally in each business process (selfdiagnosis for each business process).

4 Logic and Distributed Algorithm

The *SAgent* of each business process connected to the end events of the receives information from an oracle about what end events are correct or incorrect. Based on this information, an algorithm which involves the process choreography takes place, and each *SAgent* can determine the local diagnosis that can be merged with the rest of the business processes. To carry out the steps of this algorithm, it is necessary to send interaction information among the different *SAgents*. For this reason, we suppose that the sending of information is always possible. In general, the distributed diagnosis process has four different phases to obtain the global diagnosis. Although we are going to use an example to explain all the relevant details, the main steps of the algorithm are:

– **Determining the local clusters:** In each business process involved in the process, an offline choreography analysis is executed by the *SAgent*. The results of this step are the different clusters (according to definition 2.2) of each business process, that form an equivalent structure which is used in the rest of the steps to improve the temporal efficiency of the diagnosis process.
– **Receiving the oracle:** Each *SAgent* related to the end events ($\{o1, o2, o3, o4, o5, o6, o7\}$ in the example shown in Fig. 2) receives incoming interaction information from the oracle with the information about the outputs that fail.
– **Propagation phase:** When a *SAgent* receives incoming interaction information with the information about what external interactions can be failing, an internal algorithm to decide which interactions could be correct or incorrect takes place. As a result of this algorithm, each *SAgent* sends outgoing interaction information to the *SAgents* of its neighbor business processes informing about the correct and incorrect interactions detected. This interaction information will travel from the business processes related to the end events ($\{o1, o2, o3, o4, o5, o6, o7\}$ in the example) to the business processes that receive the start events ($\{i1, i2, i3\}$ in the example).
– **Local diagnosis phase:** Depending on the received interaction information, each *SAgents* has to decide if the local diagnosis is necessary.

In order to improve the computational complexity, the first step of the algorithm to determine the local clusters is perform offline and only once, so that the next steps are based on the precompiled structure to carry out the diagnoses.

The steps of this algorithm are represented in Fig. 3, and will be explained in more detail using the example shown in Fig. 2.

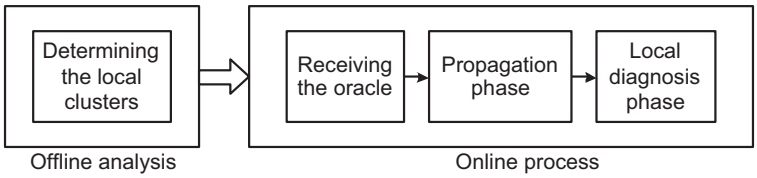

Fig. 3: Steps of the algorithm.

4.1 Determining the Local Clusters

In order to obtain the clusters of each business process, a choreography analysis takes place. Each *SAgent* derives the local clusters, which are formed by a set of activities linked through external and internal interactions (structural dependence), and not connected through internal interactions with the activities of a different cluster.

To know if an activity is working correctly is possible analyzing their outputs, inputs or other activities related to it by external or internal interactions. Within each cluster, there exist internal subsets (IS). The idea of an IS is to build a set of activities where the external and internal interactions are in more than one activity to derive the diagnosis using choreography analysis. In order to clarify it, new notations are used:

External(A_{ij}) are the external interactions (shared information) for the activity A_{ij}
Internal(A_{ij}) are the internal interactions (private information) for the activity A_{ij}

Definition 4.1. *Internal Subset (IS)* for a cluster in a business process BP_i is a set of activities, where for each one of its activities A_{ij}:

$$\begin{aligned}
&\forall\ \mathrm{v} \mid \mathrm{v} \in \mathit{Input}(A_{ij}){:}\ \mathrm{v} \in \{\mathrm{i}1, \ldots, \mathrm{i}q\} \\
&\quad \vee\ (\mathrm{v} \in \mathrm{Internal}(A_{ij}) \wedge \exists\ A_{ik} \neq A_{ij} \mid A_{ik} \in \mathrm{IS} \mid \mathrm{v} \in \mathrm{Output}(A_{ik})) \\
&\quad \vee\ (\mathrm{v} \in \mathrm{External}(A_{ij}) \wedge \exists\ A_{ik} \neq A_{ij} \mid A_{ik} \in \mathrm{IS} \mid \mathrm{v} \in \mathrm{Output}(A_{ik})) \\
&\quad \vee\ (\mathrm{v} \in \mathrm{External}(A_{ij}) \wedge \mathrm{v} \in \mathrm{External}(A_{jk}) \mid A_{jk} \in \mathrm{BP}_j \neq \mathrm{BP}_i) \\
&\forall\ \mathrm{v} \mid \mathrm{v} \in \mathit{Output}(A_{ij}){:}\ \mathrm{v} \in \{\mathrm{o}1, \ldots, \mathrm{o}r\} \\
&\quad \vee\ (\mathrm{v} \in \mathrm{Internal}(A_{ij}) \wedge \exists\ A_{ik} \neq A_{ij} \mid A_{ik} \in \mathrm{IS} \mid \mathrm{v} \in \mathrm{Input}(A_{ik})) \vee (\mathrm{v} \in \mathrm{External}(A_{ij}))
\end{aligned}$$

and for the set of activities that form each internal subset IS:

$$\begin{aligned}
&\forall\ \mathrm{v} \mid \mathrm{v} \in \mathit{Input}(IS){:}\ \mathrm{v} \in \mathrm{Input}(A_{ij}) \wedge A_{ij} \in IS \wedge (\ \mathrm{v} \in \{\mathrm{i}1, \ldots, \mathrm{i}q\} \vee \mathrm{v} \in \mathrm{External}(A_{ij})) \\
&\forall\ \mathrm{v} \mid \mathrm{v} \in \mathit{Output}(IS){:}\ \mathrm{v} \in \mathrm{Output}(A_{ij}) \wedge A_{ij} \in IS \\
&\quad \wedge\ (\ \mathrm{v} \in \{\mathrm{o}1, \ldots, \mathrm{o}r\} \vee (\ \mathrm{v} \in \mathrm{External}(A_{ij}) \wedge \nexists\ A_{ik} \neq A_{ij} \mid A_{ik} \in IS \wedge \mathrm{v} \in \mathrm{External}(A_{ik})))
\end{aligned}$$

For the example, the different clusters calculated by the *SAgents* are:

- $SAgent_1$: {$A11$, $A12$, $A13$, $A14$, $A15$ and $A16$}
- $SAgent_2$: {$A21$, $A22$ and $A23$} and {$A24$, $A25$, $A26$, $A27$, $A28$ and $A29$}
- $SAgent_3$: {$A31$} and {$A32$ and $A33$}
- $SAgent_4$: {$A41$, $A42$ and $A43$} and {$A44$ and $A45$}

For example, the activities {$A21$, $A22$, $A25$} form a new cluster with three internal subsets as it is shown in Fig. 4.

This new cluster works as a black box for the rest of the *SAgents*, and its activities can be connected with other clusters by external interactions.

In general, the different *SAgents* form a new system where the external interactions establish the local connection between the different business processes, as it is shown in Fig. 5. This obtained structure will be used in the rest of the algorithm to carry out the diagnosis process. This structure, customized for the diagnosis process, is prepared offline, and will be used to perform the local diagnosis phase, building a table with the relations between the activities and the external interactions that compose each business process.

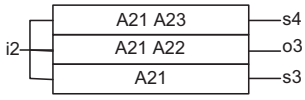

Fig. 4: A local cluster with three ISs.

4.2 Receiving the Oracle

The process choreography depends on an oracle which is able to determine the end events that can be failing.

Each *SAgent* related to the end events ($\{o1, o2, o3, o4, o5, o6, o7\}$ in the example shown in Fig. 2) receives interaction information from the oracle with the information about all the end events, indicating whether they are wrong or correct. In case of all the end events are correct, it is not necessary any propagation or diagnosis, because all the activities are working correctly. When at least an end event is wrong, the process to determine what activity does not work correctly starts. The diagnosis process starts in this step.

4.3 Propagation Phase

In order to explain next steps of the algorithm, these new definitions have to be introduced:

Definition 4.2. *Possible Incorrect Event (PIE):* It is an external interaction related to an incorrect end event. An interaction is related to an end event if when the interaction changes, the end event also changes.

Definition 4.3. *Correct Event (CE):* It is an external interaction related to a correct end event. If an interaction is related to a correct end event and an incorrect end event simultaneously, the interaction is defined as a *Correct Event*, since we suppose that two incorrect activities cannot generate a correct end event.

Depending on the end events, it is possible to know if the process choreography is working correctly. In order to know which activities are failing, the external interactions between the business processes are used to infer which interactions are failing. For example, for the local cluster shown in Fig. 4, being $s3$ a *PIE* and $s4$ a *CE*. It means that the activity $A21$ is failing, but it is not possible because if the external interaction $s4$ is correct, the activity $A21$ would be correct. It means that the external interaction $s3$ considered as a *PIE* actually

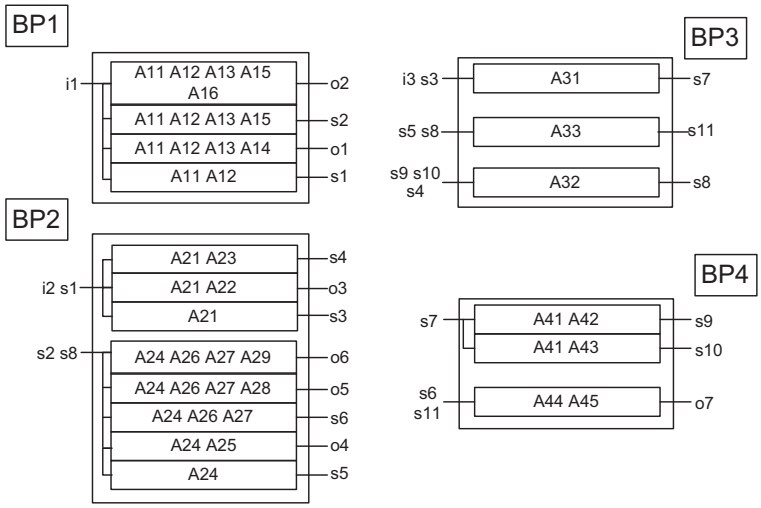

Fig. 5: Clusters and ISs of the process choreography.

it is not an incorrect event. If an interaction is correct all the activities associated to the IS of this interaction are correct. A PIE will be an IE if it has at least one activity that is not a correct activity. This idea is described in the following definition:

Definition 4.4. *Incorrect Event (IE):* Given the set *ExternalInteractions(IS_i)* that are output events of IS_i, `e` is an IE if it is a *PIE* and $\exists\ A_{kl} \in \texttt{IS}_i \mid \forall\ IS_j : j \in 1 \ldots n \wedge i \neq j \nexists\ \texttt{e'} : \texttt{e' is a}\ CE \wedge \texttt{v'} \in \texttt{ExternalInteractions(}IS_j\texttt{)} \wedge A_{kl} \in IS_j$

On this step of the algorithm, some interaction information is exchanged between the business processes. This is information about the *CE*s and *PIE*s, and the traffic direction goes from the clusters related to the end events {o1, ..., or} to the clusters which provide them the start events {i1, ..., iq}.

Therefore, according to the previous definition, the definition of *CE* is reinforced with this new concept, since a *CE* is also a event that is not an Incorrect Event.

In the previous step of the algorithm, the oracle sends the evaluation of the end events to the *SAgents* with clusters related to them, indicating which ones are correct (OK) or incorrect (KO). In this step these *SAgents* are able to infer what external interactions are *CE*s or *IE*s according to definition 4.4, and to send this interaction information to the business processes to which they are linked through their input events.

In order to do that, each *SAgent* needs to receive the information about all the outputs os a cluster before inferring whether the external interactions of that cluster are *CE*s or *IE*s. Therefore, the propagation will take place as the information of the outputs of the clusters is known by the *SAgents*. This is not a linear process since several business process can have clusters with a dependency between them. This is, for example, if a business process BPa has the clusters Cx and Cy, and another business process BPb has the cluster Cz, it is possible that Cx depends on Cz, that depends on Cy. In this example, to carry out the propagation, the *SAgent* of BPa waits for all the information about the outputs of Cx to propagate to Cz. In the same way, the *SAgent* of BPb will propagate from Cz to Cy when it has all the information about Cz. So, both *SAgents* have interchanged interaction information in both directions.

The interaction information sent by the *SAgents* are of the form shown in Fig. 6.

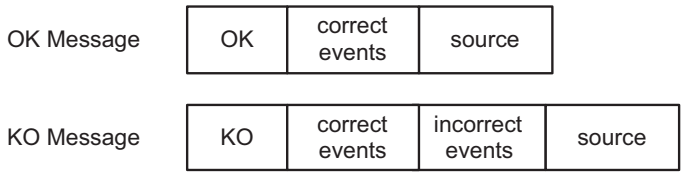

Fig. 6: Format of the interaction information.

These messages carry the following information:

- **OK/KO** field: type of information.
- **Correct events** field: the external interactions labelled as *CE*s by the *SAgent* of the business process source of the interaction information.
- **Incorrect events** field: the external interactions labelled as *PIE*s by the *SAgent* of the business process source of the interaction information.
- **Source** field: indicates the source of the interaction information. It can be a neighbor business process or the oracle to indicate the beginning of the diagnosis process.

The procedures in the algorithms 1, 2, 3 and 4 describe the behavior of a *SAgent* after receiving an OK or a KO interaction information.

Algorithm 1: Receiving a KO interaction information

```
Input: interactionInformation(KO, X_OK, X_KO, BP_i)
begin
    label as CEs the external interactions which are in X_OK;
    foreach external interaction e in X_OK do
        if e is not labelled as CE then
            label e as PIE;
    if the SAgent has received the incoming interaction information from all the
    business processes related to the outputs of the cluster C then
        checkEvents(C);
        propagateInformation(C);
    if the SAgent has not received the incoming interaction information from all the
    business processes related to its outputs then
        wait for the rest of the incoming interaction information;
end
```

Algorithm 2: Receiving an OK interaction information

```
Input: interactionInformation(OK, X_OK, BP_i)
begin
    label the external interactions which are in X_OK as CEs;
    if the SAgent has received the incoming interaction information from all the
    business processes related to the outputs of the cluster C_x then
        checkEvents(C_x);
        propagateInformation();
    if the SAgent has not received the incoming interaction information from all the
    business processes related to its outputs then
        wait for the rest of the incoming interaction information;
end
```

Algorithm 3: Procedure for labelling events as correct or incorrect

```
checkEvents(Cluster C):
begin
    /*definition 4.4*/
    foreach event e labelled as a PIE do
        if C ∈ IS_i and A_kl is an activity of IS_i and not exists another IS_j that
        contains A_kl with an event e' labelled as a CE and e' is different from e and e'
        belongs to Output(IS_j) then
            label e as an IE;
        else
            label e as a CE;
    foreach output event e labelled as IE do
        label as PIE the events which are inputs of the ISs where e is an output;
end
```

Algorithm 4: Procedure for propagating information

```
propagateInformation (Cluster C):
begin
    foreach BP_i related to C by the set of interactions X do
        if exists a subset of events X_i which are inputs of C and outputs of BP_i and
        are labelled as PIEs then
            send the interaction information (KO,X − X_i, X_i, this) to the SAgent_i;
        else
            send the interaction information (OK, X, this) to the SAgent_i;
end
```

The different clusters are connected by means of external interactions. For example, if $o5$, $o6$, and $o7$ are KO, and the rest of the end events are OK, the sent information is as follows:

- The oracle sends interaction information to $SAgent_1$, $SAgent_2$ and $SAgent_4$ indicating that $o5$, $o6$, and $o7$ are the incorrect end events.
- $SAgent_4$ has all the information about the outputs of one of its clusters. Therefore, it labels $o7$ as IE and propagates to $SAgent_3$ and $SAgent_2$ that s11 and s6 are PIEs.
- $SAgent_3$ has all the information about the outputs of one of its clusters ($s11$ is PIE). It propagates to $SAgent_2$ that $s5$ and $s8$ are incorrect.
- $SAgent_2$ has all the information about its biggest cluster ($o5$, $o6$, $s5$ and $s6$ are incorrect, and $o4$ is OK). According to definition 4.4, $s2$ and $s8$ are CEs, and this is the information propagated to $SAgent_3$ and $SAgent_1$.
- $SAgent_3$ had different kinds of information about $s8$ (CE and PIE simultaneously), therefore $s8$ is labelled as a CE. $SAgent_3$ can determine that $s4$, $s9$ and $s10$ are CEs.
- etc.

This process finishes when all the $SAgents$ have propagated all the interaction information about the inputs of all their clusters, so that the local diagnosis phase can start.

4.4 Local Diagnosis Phase

Using the information collected from the previous step, the local diagnosis is executed by the $SAgents$ which have any IE events.

The local diagnosis process has two phases (offline and online):

(i). **To build a signature matrix:** With the information obtained from Section 4.1, a signature matrix is created. This matrix relates each external interaction and end events of the business process with the activities of the ISs where this interaction participates. The matrix has *number of interactions and end events* rows and *number of activities* columns. This process is done only once, storing precompiled information. The construction of the matrix is an adaptation of the algorithm presented in [3] for business processes. An example is shown in Fig. 7(a) that represents the signature matrix for one of the clusters of $BP2$ in the example.

(ii). When the IEs are known, only the part of the matrix related to the possible wrong activities is analyzed. It means that only the rows of the IEs and the activities non related to any CE will participate in the local diagnosis. With this subset of relations among interactions, end events and activities, the set of activities which are not working correctly must be found. According to diagnosis theory, the best way to do that is calculating the hitting sets and minimal hitting sets, whose definition is as follows:
Definition 4.5 *Hitting Set (HS)* for a collection of components $\mathcal{C}$ is a set of components $\mathcal{H} \subseteq \bigcup_{\mathcal{S} \in \mathcal{C}} \mathcal{S}$ such that $\mathcal{H}$ contains at least one element for each $\mathcal{S} \in \mathcal{C}$. A HS of $\mathcal{C}$ is

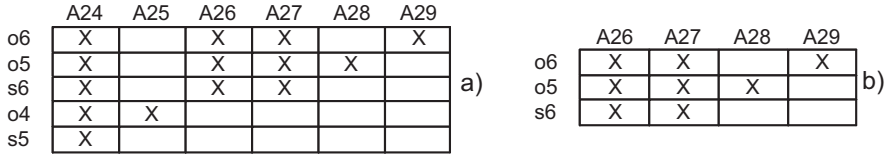

Fig. 7: Signature matrix of a cluster of $BP2$.

minimal iff no proper subset of it is a HS of $\mathcal{C}$. The minimal HSs for a set of sets are formed by $\{\mathcal{H}_1, \mathcal{H}_2, \ldots \mathcal{H}_n\}$, where $\mathcal{H}_i$ is a minimal HS of components. The cardinality of $\mathcal{H}_i$ ($|\mathcal{H}_i|$) is the number of components of $\mathcal{H}_i$. This definition can be adapted to activities instead of components.

In the case of the matrix, each set will be formed by the activities of each row of the matrix related to any IE, so that $\mathcal{C} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_n\}$, where each $\mathcal{S}_i$ is the set of software processes related to each IE. In Fig. 7(b) appears the part of the signature matrix of the cluster analyzed to perform the local diagnosis if $o5$ and $o6$ are KO and, therefore, $s6$ is IE.

Finally, the minimal hitting sets are the diagnosis of the business processes. For the example in Fig. 7(b) there are two minimal hitting sets, which are $\{A26\}$ and $\{A27\}$.

(iii). The local diagnosis can be improved storing all the final diagnosis according to the IEs. It means that if an IE has already been analyzed, the diagnosis will be very efficient. Each *SAgent* counts on a local list where it stores the fault signature from the local diagnosis executed until that moment, so that when a *SAgent* has to diagnose its activities, previously it checks whether it has already done the same diagnosis before.

Each element of the lists contains two fields with the next information:

(a) External interactions labelled as IEs.

(b) Activities which can fail according to the IEs detected.

The local diagnosis results depend on the subset of external subsets which are IEs, since two different analysis with the same set of IEs will obtain the same minimal hitting sets of activities. Using a local list to store previous results makes possible a faster process of diagnosis, because of the reutilization of the information obtained so that it is not necessary to carry out the same analysis twice.

Finally, the global diagnosis D_g can be seen as the union of all the local diagnosis: $D_g = D_1 \cup D_2 \cup ... \cup D_n$, being n the number of business processes.

According to our example, Table 1 shows some example results of the self-diagnosis on each business process, presenting what activities can be failing.

Table 1: Diagnosis Results for each Business Process

KO end events	BP1	BP2	BP3	BP4
{o5, o6, o7}	-	{A23} {A26} {A27}	{A31} {A32}	{A41} {SA44} {A45}
{o2, o4}	{A16}	{A25}	-	-
{o3, o5, o7}	-	{A21} {A28}	{A31} {A32} {A33}	{A41} {SA44} {A45}
{o1, o3, o4}	{A14}	{A22} {A25}	-	-
{o1, o2, o6}	{A14, A15} {A14, A16}	{A29}	-	-

5 Conclusions and Future Work

In this work how to perform the diagnosis of business processes is presented. These business processes work with public and private information (external and internal interactions), and the diagnosis process can be performed without needing to know neither all the information nor the business process model. By means of the use of previously compiled knowledge, a high temporal efficiency is obtained, since the preparation of an equivalent structure is performed offline.

Our proposal of fault diagnosis is an innovative solution to the diagnosis of business processes, although it does not find the global minimal diagnosis, since the set of activities found as a solution is not a global minimal hitting set.

As future work, we will work in the detection of other kinds of failures in business processes, such as the errors derived from the creation of a model that does not correspond to the real problem, or errors in the definition of the interaction between the activities of a process, or between the activities of different processes.

Acknowledgment

This work has been partially funded by the Ministry of Education and Science of Spain (DPI2006-15476-C02-01) and European Regional Development Fund(ERDF/FEDER).

References

1. R Bianchini and R Buskens, Implementation of on-line distributed system-level diagnosis theory.
2. D Borrego, M T Gómez-López and R M Gasca, Diagnosing distributed systems using only structural and qualitative information. International Transactions on Systems Science and Applications (to appear), 2008.
3. R Ceballos, M T Gomez-Lopez, R M Gasca and C del Valle, A compiled model for faults diagnosis based on different techniques. AI Commun., Vol. 20, No. 1, 2007, pp. 7–16.
4. R Debouk, S Lafortune and D Teneketzis, Coordinated decentralized protocols for failure diagnosis of discrete-event systems.
5. E Fabre, A Benveniste and C Jard, Distributed diagnosis for large discrete event dynamic systems. 15th IFAC World Congress, Barcelona 2002.
6. P Frohlich, I de Almeida Mora, W Nejdl and M Schroeder, Diagnostic agents for distributed systems. ModelAge Workshop 1997, pp. 173–186.
7. M T Gomez-Lopez, R M Gasca, C D Valle and S Pozo, Distributed model-based diagnosis using object-relational constraint databases.. AINA (2) 2006, pp. 866–870.
8. X L Guillou, M O Cordier, S Robin and L Rozé, Chronicles for on-line diagnosis of distributed systems. Research Report. ftp://ftp.irisa.fr/techreports/2008/PI-1890.pdf 2008.
9. J D Kleer, A Mackworth and R Reiter, Characterizing diagnoses and systems. Artificial Intelligence 56, Vol. 2-3, 1992, pp. 197–222.
10. Y Pencole, Decentralized diagnosier approach: application to telecommunication networks. 13th International Workshop on Principles of Diagnosis 2000, pp. 185–192.
11. G Provan, A model-based diagnosis framework for distributed systems.
12. R Reiter, A theory of diagnosis from first principles. Artificial Intelligence 32, Vol. 1, 1987, pp. 57–96.
13. I Roychoudhury, G Biswas, X Koutsoukos and S Abdelwahed, Designing distributed diagnosers for complex physical systems.. 16th International Workshop on Principles of Diagnosis 2005, pp. 31–36.
14. R Su and W M Wonham, A model of component consistency in distributed diagnosis.. 15th International Workshop on Principles of Diagnosis 2004, pp. 62–68.
15. M Weske, Business Process Management: Concepts, Languages, Architectures, Springer-Verlag, 2007.

Estimación del Esfuerzo Software: Factores vinculados a la aplicación a desarrollar

Joseba Esteban López
José Javier Dolado Cosín

Departamento de Lenguajes y Sistemas Informáticos
U.P.V./E.H.U.

Resumen. Durante la historia del software se han desarrollado multitud de métodos de estimación del esfuerzo de desarrollo en proyectos software. Cada uno de estos métodos basa su estimación en diferentes aspectos, también denominados *factores*. Este artículo tiene como objetivo listar y catalogar los principales factores utilizados en la estimación del esfuerzo software y que están relacionados con la aplicación a desarrollar . Para esto se han revisado los métodos de estimación más mencionados en la literatura, como son COCOMO, SLIM o analogía.

1. Introducción

La estimación del coste de desarrollo software, y por tanto la exactitud de esta, se ha convertido en un aspecto crucial tanto para la compañía desarrolladora del proyecto como para el cliente, el cual espera que el coste del software coincida lo más posible con la estimación. Una infravaloración en la estimación del coste software puede suponer que la planificación del proyecto acordado acabe sufriendo más problemas de los esperados inicialmente y una disminución de la calidad o retrasos en la fecha de entrega. Por el contrario, una sobrevaloración puede suponer un exceso de recursos reservados al proyecto, lo que conlleva un presupuesto excesivo del proyecto y, por tanto, una reducción de la competitividad de la compañía ante la asignación de nuevos proyectos. Una estimación lo suficientemente exacta permite a la compañía desarrolladora de software una mejor planificación de los proyectos que maneja, así como una mejor asignación de los recursos necesarios para cada proyecto. Además permite valorar el impacto ante cambios en el plan de desarrollo del proyecto [6]. En consecuencia, la exactitud de las estimaciones toma un cariz cada día más relevante dentro de la Ingeniería del Software.

Los modelos de estimación del coste de desarrollo software se basan en un conjunto de variables. Estas variables se denominan *factores*, como el coste, el esfuerzo o el tamaño entre otros. Cada modelo basa sus estimaciones en un conjunto propio de factores que pueden estar relacionados entre sí. La naturaleza de las relaciones causales entre los factores implica cierta incertidumbre: no son deterministas. Esto quiere decir que, suponiendo que exista una relación entre el esfuerzo de desarrollo y la calidad del software, no es necesariamente cierto que aumentando el esfuerzo se consiga una mejoría en la calidad, aunque si es probable que suceda. [8]

Hasta la fecha no hay un completo entendimiento sobre las relaciones causales entre factores ni de su influencia en el resultado final del proyecto software [4]. Las causas que impiden un mejor discernimiento sobre estas relaciones pueden deberse a que los métodos de estimación fallan en la representación de las relaciones causales y su incertidumbre. Asimismo se ve afectado por la gran cantidad de datos necesarios y la dificultad de obtenerlos [8]. Esta falta de información se ve acentuada con las estimaciones en las etapas iniciales del proyecto software debido a que la aplicación a desarrollar no está definida del todo. Por otro lado,

según va avanzando el desarrollo del proyecto, se conocen más datos sobre la aplicación y las relaciones entre los factores, y por tanto la exactitud de la estimación aumenta [5].

Los factores se pueden agrupar en factores del grupo de desarrollo, de la aplicación a desarrollar o relacionados con metodología de desarrollo. Este artículo muestra los factores relacionados con la aplicación a desarrollar más relevantes en la estimación del esfuerzo software. Comienza con una breve descripción de los métodos de estimación analizados. A continuación se analiza qué factores vinculados a la aplicación a desarrollar son utilizados por los métodos de estimación y cómo son valorados.

2. Métodos de estimación analizados

La estimación del coste software es el proceso de predecir el esfuerzo requerido para el desarrollo de un sistema software. La mayor parte del coste de desarrollo software es debido al esfuerzo humano [10], por lo que la mayoría de los métodos de estimación proporcionan sus estimaciones de esfuerzo en términos de persona-mes de los programadores, analistas y gestores de proyecto. La estimación del esfuerzo puede ir acompañada de otras unidades, como son la duración del proyecto (en días en el calendario) y el coste (basado en el coste medio en unidades de moneda de la plantilla implicada en el desarrollo). [6]

Históricamente la estimación del coste de desarrollo software ha supuesto la parte más compleja del desarrollo de software. Las principales causas de esta complejidad son:

- Ausencia de bases de datos históricas adecuadas de medidas de coste
- Los factores implicados en el desarrollo software están interrelacionados y sus relaciones no están bien entendidas
- Falta de práctica y habilidad del personal encargado de realizar las estimaciones

En este apartado se muestran los principales métodos de estimación sobre los que se ha basado el estudio de los factores de estimación del esfuerzo software. De cada uno de los métodos se incluye una breve descripción. Los métodos están clasificados en dos grupos: métodos algorítmicos y métodos no algorítmicos.

2.1. Métodos de estimación no algorítmicos

En este apartado se exponen brevemente los métodos de estimación que no se basan en modelos matemáticos. Estos incluyen la estimación experta y el uso de analogías entre otros.

Estimación experta El método de estimación más extendido es el juicio experto. Está basado en la experiencia del experto y su conocimiento de las normas dominantes de la industria como base para la estimación del coste software. Se puede considerar [5] como experto a aquel que cuenta con experiencia como supervisor o gestor, o tiene experiencia haciendo estimaciones. Los expertos suelen hacer sus estimaciones basándose en su experiencia y en analogías de proyectos anteriores.

A pesar de ser el método más extendido, las estimaciones basadas en el juicio experto no están libres de problemas, ya que este enfoque no es ni repetible ni explícito. Además de que contar con un experto experimentado para cualquier nuevo proyecto es muy complicado, sus estimaciones están sujetas a un alto grado de inconsistencia. Según el estudio realizado por Stein G. y Magne J. en , dos estimaciones de un experto sobre el mismo proyecto pueden variar de media hasta un 71 %, con una mediana del 50 %. Un intento de minimizar la inconsistencia de las estimaciones de los expertos, consiste en consultar a varios expertos. Como ejemplos de técnicas de consenso entre expertos están Delphi y PERT [6]. Por otro lado, los gestores de los proyectos software intentan maximizar la productividad y minimizar

el coste, y así hacer coincidir las estimaciones con el objetivo fijado con el cliente. Por tanto, los expertos basan sus estimaciones en estas manipulaciones de los presupuestos de proyectos anteriores de la compañía, con lo que su experiencia queda en entredicho.

Los factores que manejan los expertos a la hora realizar sus estimaciones no son muy conocidos. Por tanto, en este artículo no se hablará de este método en las siguientes secciones donde se analizan los factores empleados por los métodos de estimación.

Uso de analogías La utilización de analogías a la hora de estimar el coste de desarrollo software, consiste en comparar el nuevo proyecto a estimar con el coste real de proyectos anteriores de la compañía. La comparación entre proyectos se puede hacer comparando el proyecto en su totalidad o por partes. Comparar los proyectos basándose en su descomposición tiene la ventaja de ser un proceso más detallado que si se comparan proyectos en su totalidad. Por contra, comparar los proyectos en su totalidad lleva implícita la ventaja de incluir todos los componentes relativos al coste, incluso los componentes de los que no se tiene constancia.

La ventaja de este método radica en que las estimaciones se hacen en base al coste real de los proyectos de la compañía. Esto implica que la compañía debe disponer de una cantidad de datos de proyectos anteriores suficiente. Aún así, no queda muy claro que proyectos anteriores sean representativos de los nuevos proyectos. [6]

Una implementación de este método es la herramienta ANGEL [9] que se basa en la minimización de la distancia euclídea según las variables consideradas y que soporta la recolección, almacenaje e identificación de los proyectos más similares a la hora de estimar el esfuerzo.

Otros métodos En este apartado se comentan métodos menos extendidos dentro de los métodos de estimación no algorítmicos.

Por un lado encontramos el principio de Parkinson. Este principio determina el coste de desarrollo basandose en la máxima de que el trabajo se expande hasta completar todo el volumen disponible. Por ejemplo, si un proyecto debe estar terminado en 12 meses y se dispone de 5 personas, el esfuerzo quedará determinado como 60 personas/mes. [6]

Price-To-Win es un método consistente en determinar el coste del proyecto en función del precio que está dispuesto a pagar el cliente. Con este método es habitual que el encargado de realizar la estimación se vea obligado a ajustarla en función de lo que quiere pagar el cliente y, así, conseguir que le asignen el proyecto a la compañía. La aplicación de este método puede derivar en retrasos en el desarrollo o en que la plantilla se vea obligada a hacer horas extra. Al igual que el método de Parkinson, no se trata de un método recomendable a la hora de estimar el esfuerzo.[6]

Los enfoques Bottom-up y Top-down generan estimaciones basándose en la descomposición del software en componentes. Con Bottom-up se estima por separado cada componente del sistema. Posteriormente se unen todas las estimaciones para generar la estimación del proyecto en su totalidad. El único requerimiento para este enfoque radica en conocer perfectamente cómo se descompone el sistema, algo complicado en las etapas iniciales del desarrollo. Por el contrario, con el enfoque Top-down primero se realiza una estimación global del proyecto, utilizando el método de estimación que se prefiera, y a continuación se divide en componentes. Este enfoque resulta más útil en fases tempranas del desarrollo [6]. Ambos enfoques no son métodos de estimación como tales, sino estrategias para facilitar la tarea de estimar el esfuerzo de desarrollo software.

2.2. Métodos de estimación algorítmicos

Los métodos de estimación algorítmicos realizan sus estimaciones de coste en función de un conjunto de variables y parámetros. Estas variables están consideradas como los principales factores de coste. Los diferentes métodos de estimación algorítmicos se diferencian tanto por los factores que emplean en su modelo, como por la forma de la función que utilizan. Dependiendo del tipo de formula matemática que utiliza un modelo, se puede clasificar en modelos lineales, multiplicativos o con función de rendimiento (Power Function Models). Los métodos algorítmos se dividen en dos grupos: analíticos y empíricos [6].

Métodos de estimación algorítmicos empíricos Los métodos de estimación empíricos se basan en la experiencia a la hora de realizar una formulación matemática que modele el esfuerzo de desarrollo software. En este apartado se describe brevemente el método de estimación COCOMO.

COCOMO (COnstructive COst MOdel) Este método de estimación fue ideado por B. W. Boehm [1] a principios de la década de los 80. Está basado en el método de Walston-Felix [10]. Dado que en este artículo se estudian los factores relacionados con el esfuerzo y la aplicación a desarrollar, de ahora en adelante se considerará COCOMO y sus factores.

Se trata de un método de estimación basado en una función de rendimiento de la forma: $Effort = a \times S^b \times m(X)$

Donde S es el tamaño del código en miles de líneas (KLOC). Las otras dos variables, a y b, son coeficientes dependientes de la complejidad del software. Con $m(X)$ se representa un multiplicador dependiente de 15 factores. En COCOMO los factores de coste se dividen en cuatro grupos: factores de producto (fiabilidad requerida, tamaño de la base de datos y complejidad del producto), factores de computación (restricciones de tiempo de ejecución, de almacenaje, inestabilidad de la máquina virtual y de tiempo de respuesta del equipo), factores de la plantilla (capacidad del analista, capacidad del programador, experiencia con la aplicación a desarrollar, con el lenguaje de desarrollo y la máquina virtual) y factores del proyecto (plan de desarrollo requerido, herramienta de desarrollo y la utilización de prácticas modernas de programación).

Este método basa la calibración de sus coeficientes en 63 proyectos finalizados, para los que consigue resultados moderados, aunque no se puede concluir que sea válido para todos los casos.

El modelo de COCOMO reflejaba las prácticas de desarrollo de software de la época de los 80. En la década y media siguiente estas técnicas cambiaron radicalmente, por lo que la aplicación del modelo original empezó a resultar problemática y se acabó por desarrollar un nuevo modelo: COCOMO II [2]. En lo referente a este estudio, los factores que afectan a la estimación de esfuerzo software, se produce un cambio entre el modelo viejo y el más reciente. Del COCOMO original se desecharon varios factores (VIRT, TURN, VEXP, LEXP y MODP) y se aportaron otros nuevos (DOCU, RUSE, PVOL, PEXP, LEXT, PCON y SITE). En este artículo se tratan los factores de ambos modelos y, por tanto, no se hace diferencia entre versiones.

Métodos de estimación algorítmicos analíticos Los métodos de estimación analíticos se basan en una comprensión del problema, descomponiéndolo para así comprender mejor su comportamiento. A a partir de aquí se pueden desarrollar ecuaciones matemáticas que modelen el problema de estimar el esfuerzo de desarrollo software. Este artículo expone brevemente dos métodos de estimación algorítmicos analíticos: SOFTCOST y SLIM.

SOFTCOST [3] El modelo SOFTCOST asume una relación lineal entre el esfuerzo y el tamaño de la aplicación mediante esta fórmula matemática:

$E = S/P_1$ con $P_1 = P_0 A_1 A_2$

Donde E es el esfuerzo, S es el tamaño estimado en KLOC y P_1 es la productividad media en KLOC/Personas-Mes. Para definir P_1 se utilizan varios factores. P_0 es una constante, A_1 engloba 6 factores de ajuste de productividad combinados mediante una fórmula matemática. A_2, al igual que A_1, es una combinación de otros 29 factores.

El método SOFTCOST engloba un total de 68 parámetros. La implementación del SOFTCOST es interactiva y recopila la información de una serie de 47 preguntas con las que deduce los valores de los factores. Como salida ofrece el esfuerzo y la duración del desarrollo software descompuesto en una estructura estándar. También ofrece una estimación del nivel de la plantilla, tamaño en páginas de la documentación y requerimientos del procesador.

SLIM Este método de estimación se apoya en el modelo teórico de Norden-Rayleigh sobre la forma de la curva de los desarrollos de los proyectos y se supone que es aplicable a proyectos con un tamaño superior a 70.000 líneas de código. El método de estimación SLIM se basa en el punto de vista de que la cantidad de trabajo asociado a cualquier producto se puede ver como una proporción del producto entre en esfuerzo realizado y tiempo necesario para conseguirlo:

$$Producto = Parmetro\,de\,Productividad \cdot \left(\frac{Esfuerzo}{B}\right)^{\frac{1}{3}} \cdot Tiempo^{\frac{4}{3}}$$

Donde *Producto* representa cierta medida sobre la funcionalidad del mismo y se suele medir en SLOC. El *Esfuerzo* viene medido en personas-año o personas-mes. El *Tiempo* representa la cantidad de tiempo empleada en el desarrollo y se mide en meses o años. El *Parámetro de Productividad* (PP) es una constante que engloba los factores del entorno. PP expresa la productividad de la compañía. Esta constante permite igualar las otras tres variables. De manera que la cantidad de *Producto* con el mismo *Esfuerzo* y *Tiempo* dedicado puede variar dependiendo del entorno de trabajo. El parámetro B representa la destreza en función del tamaño del sistema. Las potencias de la fórmula matemática corresponden a sucesivas validaciones. La forma que se muestra en este artículo la adquirió a mediados de os 80, cuando el método se validó con 750 sistemas.

Este método de estimación merece mención aparte en cuanto a los factores en los que se basa, que es el objetivo de análisis de este artículo. Aparte del tamaño de la aplicación a desarrollar, en este método se tienen en cuenta el conjunto de factores relacionados con el entorno (*Parámetro de Productividad - PP*) y la tasa de contratación de personal (*Manpower Build Parameter - MBP*), pero no de manera explícita o desglosada. Ambas variables se definen a partir de proyectos ya realizados. Dentro de un mismo proyecto ya finalizado, tanto *PP* como *MBP* se comportan como una constante. Así, una vez establecidos los parámetros de esfuerzo, tiempo y tamaño de proyectos anteriores, se pueden definir ambos parámetros. *PP* se comporta como un parámetro de proporcionalidad dentro la fórmula de SLIM. De esta manera se establece a grado de productividad asignada a la compañía en un momento dado. El parámetro *MBP* se establece de manera análoga.

El *Parámetro de Productividad* constituye una macromedida del entorno general de desarrollo. *PP* engloba el conjunto de factores que afectan a toda la organización, como la gestión del proyecto, la utilización de buenos requerimientos, diseños, codificaciones, inspecciones y pruebas. También incluye aspectos como el nivel del lenguaje de programación, el estado de la tecnología, la experiencia de los miembros del grupo y la complejidad de la aplicación. Es por esto que no se puede determinar qué valor o peso se asigna a cada uno de los aspectos que engloba. Se obtuvieron una serie de valores representativos de *PP* a partir de la base de datos QSM. A cada valor representativo de *PP* se le asignó un índice (*PI*). De esta manera, una compañía obtendrá un valor bajo de *PI* cuando trabaje en entornos elementales con herramientas inadecuadas, o cuando el producto tenga un alto grado de complejidad (como

microcódigo o firmware). Los valores altos de *PI* se asocian a compañías con un buen entorno y personal experimentado, o cuando se trate de productos de baja complejidad que se comprenden bien.

El parámetro *MBP* se obtiene por calibrado de proyectos anteriores. Representa el estilo de gestión de la empresa o la tasa de contratación de personal. Se calcularon seis valores representativos del parámetro *MBP* a partir de la base de datos QSM. A cada valor representativo se le asignó un índice (*MBI*). Estos índices representan desde una tasa de contratación lenta (*MBI* =1) a una tasa extremadamente rápida (*MBI* = 6). Con este parámetro se ha observado que acumular personal moderadamente reduce el esfuerzo de desarrollo, en comparación con hacerlo más intensamente. En otras palabras, se puede reducir el tiempo de desarrollo aumentando la tasa de contratación (*MBI*), y por consiguiente el esfuerzo, pero sólo se consigue con un gasto elevado en comparación con la reducción en el tiempo de desarrollo.

3. Factores de la aplicación a desarrollar

A la hora de estimar el esfuerzo empleado en el desarrollo de un software, hay que estimar el esfuerzo de una aplicación determinada, con una plantilla concreta y un método de desarrollo. Tanto la aplicación a desarrollar como la plantilla y el método de desarrollo empleado cuentan con unas características propias, denominados factores. Los atributos o factores relacionados con el producto software tienen gran importancia dentro de la ingeniería del software. Esta idea se suele resumir mediante el lema "Una buena estructura interna supone una buena calidad externa". Con lo cual, es evidente la importancia de estos factores en el desarrollo del software y el esfuerzo dedicado. [7]

En esta sección se exponen los factores más habituales en la estimación del esfuerzo software dentro del grupo de factores propios de la aplicación a desarrollar. Se da una visión de cada factor, con qué otros factores está relacionado y cómo es tratado por cada uno de los métodos de estimación expuestos en la sección anterior.

3.1. Tamaño (*Size*)

El factor de tamaño, o *size*, se suele medir en líneas de código (LOC), en miles de líneas de código (KLOC) o en líneas de código fuente (SLOC). También se puede medir mediante puntos de función [7] o teniendo en cuenta el espacio de almacenamiento (bytes o cualquiera de sus múltiplos).

Comúnmente se ha considerado el factor del tamaño como el más relacionado con el esfuerzo software [7], aunque contabilizar líneas de código no es un proceso claro. Entre el código que se desarrolla se pueden encontrar líneas en blanco o comentarios. Estas líneas no están relacionadas directamente con el esfuerzo de desarrollo, aunque sí influyen en la legibilidad del código, útiles a la hora de modificar una aplicación existente. Hay que tener en cuenta que no todas las líneas de código que se escriben, y que por tanto suponen un esfuerzo, llegan a ser entregadas y que la utilización de generadores automáticos (pantallas, formularios o interfaces de usuario) generan miles de líneas de código con un esfuerzo mucho menor. Ante una aplicación orientada a objetos, establecer el tamaño de la misma en líneas de código no apropiado. En consecuencia, se puede decir que el tamaño en relación al esfuerzo es significativo pero no determinante.

COCOMO basa su estimación del esfuerzo en la posibilidad de conocer el tamaño, medido en KLOC o puntos de función, del software a realizar. En la ecuación del modelo COCOMO se puede ver una proporción del esfuerzo software y el tamaño de este, ajustando con los coeficientes (a y b) y la función $m(X)$, que considera el peso, en relación al coste, del resto de atributos del proyecto software. Es decir, traslada la estimación del esfuerzo a la estimación

del tamaño. En consecuencia, el factor del tamaño del software adquiere un carácter mucho más determinante en la estimación del esfuerzo que el resto de factores.

El método de estimación SLIM también otorga gran relevancia al tamaño (denominado *producto* o *tamaño*) del software a la hora de estimar el esfuerzo. El tamaño del software en SLIM es proporcional al producto del la "productividad" por el esfuerzo y el tiempo. Se mide en líneas de código fuente (SLOC).

3.2. Lenguaje de desarrollo

Con este factor se tiene en cuenta el lenguaje que se utiliza a la hora de desarrollar la aplicación software. En la actualidad existen multitud de lenguajes. Cada lenguaje permite desarrollar un tipo de software de manera óptima dependiendo de su nivel de abstracción (desde lenguaje máquina a BASIC), según su forma de ejecución (compilados o interpretados) o el paradigma de programación donde podemos encontrar lenguajes imperativos (como C#, Java o Perl), lenguajes funcionales (como Lisp), lenguajes lógicos (Prolog) o lenguajes orientados a objetos (como Java, Visual Basic .Net, PHP o Ada entre otros). Unos son más adecuados que otros a la hora de realizar ciertas aplicaciones. Por ejemplo, para desarrollar una aplicación para un dispositivo móvil se puede utilizar un lenguaje de nivel bajo como ensamblador, pero el esfuerzo empleado sería muy alto. Por el contrario, si se utiliza un lenguaje de alto nivel u orientado a objetos como Java, parte del esfuerzo de desarrollo se emplearía en adaptar la funcionalidad de los diferentes paquetes a un dispositivo sin una capacidad de procesado elevada. En cambio, si se utiliza un lenguaje adecuado, como Java Mobile, el esfuerzo se minimiza ya que los paquetes de este lenguaje están preparados para este tipo de dispositivos. Aparte de este aspecto, el factor del lenguaje de desarrollo debe tener en cuenta la complejidad intrínseca de cada lenguaje (programar en un lenguaje de alto nivel es más sencillo e intuitivo que uno de nivel bajo).

Este factor lo tienen en cuenta varios métodos de estimación. El método de estimación SLIM lo denomina *NLENG* y lo incluye dentro del parámetro de Productividad calibrándolo con los proyectos ya realizados.

El método de B.W. Boehm (COCOMO) no tiene en cuenta el lenguaje de desarrollo, al menos directamente. En este método se considera que el lenguaje de desarrollo y la experiencia de los desarrolladores afectan de manera conjunta al esfuerzo. Cuanta más experiencia acumulan los desarrolladores, programan de manera más segura y con menor número de errores, reduciendo el esfuerzo.

3.3. Herramientas de desarrollo

Aparte de lo adecuado que sea el lenguaje de programación con la aplicación que se quiere desarrollar, la herramienta de desarrollo también juega un papel importante en el esfuerzo. Desarrollar un sistema complejo con una herramienta elemental, multiplica el esfuerzo dedicado o lo convierte en una tarea casi imposible. Por el contrario, desarrollar una aplicación muy simple con una herramienta muy compleja, aunque potente, requiere un esfuerzo extra dedicado al manejo de la herramienta. Por tanto, emplear la herramienta adecuada para cada tipo de aplicación implica mejorar la productividad y ajustar el esfuerzo de desarrollo. Cada herramienta de desarrollo lleva implícito un periodo de adaptación por parte del grupo de desarrolladores para aprender a utilizar dicha herramienta y sacarle todo el partido. En consecuencia, un cambio de herramienta en un proyecto conlleva un retraso y un aumento del esfuerzo.

Dentro de los factores que se tienen en cuenta en COCOMO, *TOOL* es el factor con el que se plasma el uso adecuado de las herramientas de desarrollo. Los valores de *TOOL* varían desde "muy bajo" cuando sólo se utilizan herramientas básicas, a "muy alto" cuando se utilizan herramientas específicas.

3.4. Fiabilidad del software

La fiabilidad del software se define como la probabilidad de ejecutar un software y que no cause ningún fallo en el sistema durante el tiempo especificado y bajo unas condiciones específicas. También se puede ver como la probabilidad de que exista un fallo en el software y sea ejecutado. Un fallo en el sistema a realizar puede suponer desde un problema de fácil solución y sin importancia, hasta la pérdida de vidas humanas (por ejemplo en el software de control aéreo).

Existen diversas técnicas para mejorar la fiabilidad del software. Con la prevención de fallos se intentan corregir los fallos durante la etapa de desarrollo. Una vez el producto software ha sido desarrollado se pueden descubrir sus posibles fallos con la aplicación de técnicas de detección de fallos. Con las técnicas de tolerancia a fallos el objetivo es proporcionar una respuesta controlada ante fallos no detectados.

Se puede ver que este factor está estrechamente ligado a la fase de pruebas dentro del desarrollo del software. Por tanto, cuanto mayor sea la necesidad de fiabilidad del software a desarrollar, más concisa será la fase de pruebas y, consecuentemente, el esfuerzo necesario para el desarrollo software se verá incrementado.

Este factor se tiene en cuenta en el método de estimación COCOMO. Se denomina *RELY* e indica las posibles consecuencias para el usuario en el caso de que existan defectos en el producto. El factor puede tomar cinco posibles valores, desde "muy bajo" a "muy alto". Un valor "muy bajo" indica que el efecto de un posible fallo sólo tendría como consecuencia el inconveniente de solucionarlo. Con un valor "bajo" las consecuencias supondrían una perdida fácilmente recuperable para los usuarios. Cuando la perdida para los usuarios es moderada y permite salvar la situación sin demasiada dificultad, *RELY* tomaría el valor "nominal". Ante un valor "alto" las consecuencias del fallo pueden suponer una perdida financiera o una molestia a un gran número de usuarios. El valor más extremo, "muy alto", supondría la perdida de vidas humanas o grandes perdidas financieras.

3.5. Almacenamiento del software

Las necesidades de almacenamiento requeridas por el software, principalmente el tamaño de la base de datos, pueden suponer esfuerzo extra en el desarrollo del software. Hoy en día el tamaño del software no suele ser un aspecto tan preocupante como lo fue a mediados de los 70. Actualmente el precio por megabyte de almacenaje se ha reducido considerablemente, además de que se han mejorado las técnicas de compresión. Aun así, ante un software que tenga una restricción de espacio (como pueden ser los sistemas para dispositivos móviles o empotrados), ajustar el tamaño a ese espacio puede suponer un incremento del esfuerzo importante.

Hoy en día el espacio necesario para un sistema o una base de datos no muy grande no supone un gran inconveniente. No ocurre lo mismo con las grandes bases de datos. Si el software a realizar necesita apoyarse en una base de datos de un tamaño importante (por ejemplo una aplicación bancaria), el esfuerzo se puede ver incrementado por la necesidad gestionar la infraestructura de almacenaje necesaria. Por norma general, en proyectos de tamaño medio, el espacio de almacenamiento tanto del código como de los datos, no suele suponer ningún esfuerzo extra.

COCOMO tiene en cuenta el aspecto del almacenamiento del software mediante dos factores: *DATA* y *STOR*. Con la medida *DATA* se captura cuanto se ve influenciado el desarrollo del producto software por unos requerimientos de almacenamiento elevados. Este factor es importante en cuanto al esfuerzo de generar datos que posteriormente se utilizaran para probar el sistema, introducir nuevos datos en la base de datos o adaptarlos. *DATA* viene definido por el coeficiente:

$\frac{D}{P} = \frac{TBD}{TP}$

Donde *TBD* representa el tamaño de la base de datos (la cantidad de datos a ser articulada y almacenada en memoria secundaria hasta la entrega del producto software) y *TP* el tamaño del código en SLOC. El valor del factor se discretiza en cuatro segmentos (0-10, 10-100, 100-1000. +1000) que determinan los valores de *DATA*, que son "bajo", "nominal", "alto" o " muy alto".

Parte del almacenamiento principal del software se utiliza para guardar el código. Si el software a realizar tiene como requisito la obligación de correr en un volumen menor al del almacenamiento principal, el esfuerzo de programación se verá incrementado. Este echo lo captura COCOMO mediante el factor *STOR*, asignándole un valor desde "nominal", cuando la reducción del almacenamiento principal es del 50 %, hasta "extremadamente alto", cuando la reducción es del 95 %.

3.6. Complejidad del software

A la hora de establecer el esfuerzo asociado a la realización o mejora de un determinado software, resulta esencial conocer la dificultad intrínseca al propio software. Evidentemente, abstrayéndose de las ventajas o dificultades que presente el lenguaje con el que se programe o la aplicación de desarrollo, una aplicación sencilla siempre tendrá asociado un esfuerzo de desarrollo menor que otra más compleja de realizar.

La complejidad o dificultad asociada a un software depende del dominio del problema (gran cantidad de requisitos incluso contradictorios, dificultad en la comunicación con el cliente), de los impedimentos en la gestión del proceso de desarrollo (manejar grandes cantidades de código, la necesidad de gestionar un gran equipo de desarrollo, gran esfuerzo en el análisis y diseño, gestionar un ciclo de vida largo, etc.), el nivel de detalle exigido o los límites de la capacidad humana. Por otro lado, la complejidad en el desarrollo de una aplicación se reduce si la plantilla asociada al proyecto cuenta con la experiencia suficiente en el desarrollo de aplicaciones similares.

El método de estimación COCOMO captura la complejidad del software mediante el factor: *CPLX*. Con *CPLX* se indica la complejidad de cada módulo y que, una vez determinada la complejidad de todos los módulos, se utiliza para determinar la complejidad compuesta del sistema. *CPLX* toma valores desde "bajo", cuando se trata de módulos compuestos por expresiones matemáticas simples, hasta "extremadamente alto" para módulos con muchos recursos de planificación. Con la segunda versión de este método, el factor *CPLX* se ha mantenido pero ha cambiado la escala de valores. La complejidad pasa a decidirse evaluando cinco tareas que caracterizan el software a desarrollar. Las tareas a evaluar son las operaciones de control, operaciones computacionales, operaciones dependientes de dispositivos, operaciones de gestión de datos y operaciones de gestión de interfaz de usuario. Cada una de estas tareas se valora de manera subjetiva mediante una escala de seis valores que van desde "muy bajo" a "extremadamente alto".

3.7. Factores de plataforma

En esta sección se exponen los factores que capturan el esfuerzo extra de ajustar la aplicación a desarrollar al entorno donde se va a implementar o ejecutar. El esfuerzo de desarrollo de una aplicación que se ejecute en modo local (sin acceso a la red) será menor que si ese mismo sistema accediese a la red. La misma diferencia se puede encontrar con el acceso a una base de datos así como la adaptación del software a uno o varios sistemas operativos. También influye el echo de tener que adaptar el sistema al hardware de la máquina, ya sea por estar limitada en cuanto a su capacidad de computo (por ejemplo un dispositivo móvil), como por la necesidad de gestionar un hardware específico (por ejemplo un dispositivo haptico para medicina). El esfuerzo de desarrollo también se puede ver afectado por los

requisitos de la máquina en la que se desarrolla el software: el tiempo de respuesta. Así, cuanto mayor sea el tiempo de respuesta, más alto será el esfuerzo humano en la fase de desarrollo. Un ejemplo que ilustra este aspecto puede verse en el desarrollo de aplicaciones de minería de datos. Si se va a ejecutar el software en una máquina con poca capacidad de computo, que tarde en mostrar los resultados, el esfuerzo del desarrollador será mucho mayor que si dispusiera de una máquina más potente que le permitiese visualizar los resultados de la ejecución más rápidamente.

En el método de estimación COCOMO se tienen en cuenta estos aspectos mediante los factores *TURN*, *VIRT*, *PVOL* y *TIME*. El factor *TURN* cuantifica el tiempo de respuesta del ordenador desde el punto de vista del programador. *TURN* puede variar desde "bajo" para un sistema interactivo, a "muy alto" cuando el tiempo medio de respuesta es de más de 12 horas. Este factor desaparece en la nueva versión de COCOMO II debido a que la mayoría de los desarrolladores tienen acceso casi inmediato a los recursos computacionales de su estación de trabajo. Aún así, este factor ha sido tenido en cuenta en este estudio dado que, si la aplicación con la se desarrolla el software tiene un tiempo de respuesta alto, el esfuerzo puede verse incrementado. Igualmente, si el propio sistema a desarrollar tiene un tiempo de respuesta significativamente largo, la fase de pruebas se verá afectada aumentado su duración. Por tanto, actualmente, este factor sólo es significativo para el esfuerzo en los casos en que el tiempo de respuesta sea notablemente alto.

TIME es el factor con el que cuantifica la restricción del tiempo de ejecución del sistema a desarrollar. Se expresan en términos de porcentaje de tiempo de ejecución disponible que se espera que sea utilizado por la aplicación. Este factor puede ser considerado como un factor complejidad, ya que se basa en el echo de que para un programador siempre será más exigente desarrollar una aplicación con una restricción en el tiempo de ejecución. Los valores van desde "nominal" (menos del 50 % de recursos de tiempo de ejecución utilizados) hasta "extra alto" (95 % del recurso de tiempo de ejecución consumido).

La volatilidad de la plataforma (hardware y software) en el método COCOMO se refleja mediante dos factores: *PVOL* y *VIRT*. Ambos hacen referencia a lo posibles cambios que se pueden producir tanto en la plataforma de desarrollo (*VIRT*) como en la que se ejecutará el sistema a desarrollar (*PVOL*). Los valores que pueden tomar van desde "bajo", donde cada 12 meses se produce un cambio importante, a "muy alto", donde se produce un cambio importante cada dos semanas. En la segunda versión del método el factor *VIRT* desaparece, con lo que no se tiene en cuenta la volatilidad de la plataforma de desarrollo.

3.8. Requerimientos no funcionales

Con este apartado se engloban aquellas características que se exigen a la aplicación a desarrollar, como la posibilidad de reutilización o una documentación exhaustiva.

El método de B. W. Boehm utiliza el driver de coste *RUSE* para contabilizar el esfuerzo extra necesario para el desarrollo de componentes pensados para ser reutilizados en otros proyectos. La mayoría de este esfuerzo se atribuye en la creación de un diseño del software más genérico, una documentación más detallada y pruebas más extensas, así se asegura que los componentes están listos para utilizarse en otras aplicaciones. *RUSE* puede tomar cinco valores que van desde "bajo" cuando el componente no se va a reutilizar, a "extremadamente alto" cuando se va a reutilizar a lo largo de múltiples líneas de producto. El nivel de documentación requerida, COCOMO lo analiza con el driver de coste *DOCU*. Este factor se evalúa en términos de adecuación de la documentación del proyecto a las necesidades del ciclo de vida del mismo. *DOCU* puede tomar valores desde "muy bajo" cuando la documentación deja muchas necesidades del ciclo de vida sin cubrir, hasta "muy alto" cuando la documentación resulta excesiva para las necesidades del ciclo de vida.

4. Conclusiones

Este artículo ha estudiado los factores más importantes relacionados con la "aplicación a desarrollar". Uno de los principales inconvenientes que se han presentado a la hora de hacer este estudio ha sido la falta de información en cuanto a este grupo de factores, su grado de influencia en el esfuerzo de desarrollo y las relaciones con el resto de factores.

Teniendo en cuenta el grado en el que estos factores inciden en el esfuerzo, consideramos que los que afectan de manera más directa al esfuerzo son la fiabilidad y la complejidad. Esto es debido a que ante unos valores altos de estos factores, el esfuerzo de desarrollo aumenta significativamente tanto en la fase de desarrollo como en las fases de análisis y pruebas. Estos dos factores pueden ver reducido sus efectos sobre el esfuerzo. Si la compañía desarrolladora cuenta con una gran experiencia en proyectos de similar dificultad y fiabilidad requerida, es lógico pensar que el esfuerzo de desarrollo se verá reducido aún manteniéndose alto.

En este sentido los factores del lenguaje y la herramienta de desarrollo así como los requisitos no funcionales, afectan al esfuerzo de igual manera, viendo reducida su relevancia sobre el esfuerzo si el equipo de desarrollo cuenta con experiencia en proyectos similares o con la herramienta y el lenguaje de desarrollo empleados. En este caso, con experiencia con la herramienta y el lenguaje de desarrollo, el esfuerzo tendría una relación más directa con el tamaño de la aplicación a desarrollar. Es por esto, que el factor del tamaño tiene una relación significativa con respecto al esfuerzo, pero no determinante. Si además se tiene en cuenta los problemas de medición del tamaño del software, tanto en líneas de código como con los puntos de función, y los resultados no muy buenos que se han obtenido por parte de los métodos de estimación basados en este factor (como COCOMO, SOFTCOST y SLIM), el factor del tamaño no debería de tener tanta importancia en relación al esfuerzo software.

Por último, los factores de plataforma y de almacenamiento, tanto del código como de la base de datos, son los que tienen una relación menos directa con el esfuerzo software. Hoy en día el problema de almacenamiento y la de las restricciones de computo no suponen un inconveniente que requiera de un esfuerzo extra. En casos extremos, como puede ser el desarrollo de una aplicación para un sistema móvil con restricciones de computo importantes y plataforma (pantalla reducida, teclado, etc.), el esfuerzo no se verá incrementado siempre que se utilicen las herramientas y el lenguaje de desarrollo apropiados para la plataforma. Ante un sistema que requiera una base de datos muy grande se debe tener en cuenta el factor del tamaño, ya que incide en el esfuerzo de manera significativa tanto en la gestión de la base de datos como en la introducción de datos para las pruebas.

En este artículo no se han comentado otros grupos de factores como los factores humanos o los relacionados con la metodología de desarrollo. Estos factores pueden tener la misma importancia que los factores comentados en este estudio y quedan como trabajo futuro.

Referencias

1. Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.
2. Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, August 2000.
3. S. D. Conte, H. E. Dunsmore, and Y. E. Shen. *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986.
4. J. J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43(1):61–72, 2001.
5. Stein Grimstad and Magne Jorgensen. Inconsistency of expert judgment-based estimates of software development effort. *Journal of Systems and Software*, 80(11):1770–1777, nov 2007.
6. Zhang Fan Hareton Leung. Software cost estimation.

7. Jose Javier Dolado Cosin Luis Fernandez Sanz. *Medicion para la Gestion en Ingenieria del Software*. feb 2000.
8. Emilia Mendes. The use of a bayesian network for web effort estimation. In *ICWE 2007, Web Engineering, 7th International Conference, Como, Italy, July 16-20, 2007, Proceedings*, volume 4607 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2007.
9. M. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. In *Software Engineering, 1996, Proceedings of the 18th International Conference on*, pages 170–178, 1996.
10. Claude E. Walston and Charles P. Felix. A method of programming measurement and estimation. *j-IBM-SYS-J*, 16(1):54–73, 1977. See letters \citeHalstead:1977:FLC,Walston:1977:FAR.

Uso de Meta-Análisis para Integrar Resultados Experimentales

Mª Esperanza Manso[1], José A. Cruz-Lemus[2], Marcela Genero[2], Mario Piattini[2]

[1]Grupo de investigación GIRO, Departamento de Informática, Universidad de Valladolid.
Campus Miguel Delibes, E.T.I.C., 47011, Valladolid, España.
manso@infor.uva.es
[2]Grupo de investigación ALARCOS, Departamento de Tecnologías y Sistemas de Información.
Universidad de Castilla-La Mancha, Paseo de la Universidad, 4. 13071 Ciudad Real, España.
{JoseAntonio.Cruz, Marcela.Genero, Mario.Piattini}@uclm.es

Resumen. El objetivo principal de este artículo es presentar diferentes técnicas estadísticas, en especial el meta-análisis, que permiten integrar los resultados obtenidos en familias de experimentos y realizar revisiones de estudios con hipótesis comunes de una forma objetiva y sistemática. Aunque estas técnicas son mejores que las revisiones subjetivas de estudios no están exentas de riesgos, ni son aplicables en todas las condiciones. Además veremos cómo se está utilizando el meta-análisis en la Ingeniería del Software empírica, y como caso práctico aplicaremos esta técnica en una familia de cinco experimentos controlados realizados en entornos académicos, cuyas hipótesis investigan por un lado la correlación entre la complejidad estructural y el tamaño de los diagramas de clases UML y su complejidad cognitiva y, por otro, la correlación entre la complejidad cognitiva y la entendibilidad y modificabilidad de dichos diagramas. Los resultados obtenidos tienen implicaciones tanto desde el punto de vista del modelado como el de la enseñanza, ya que muestran evidencia empírica sobre cuáles son los constructores UML que tienen más influencia cuando los modeladores tienen que comprender y modificar los diagramas de clases UML.

Palabras Clave: Meta-análisis, Ingeniería del Software Empírica, experimentación.

1. Introducción

Si consideramos la ciencia como un medio para acumular y refinar tanto la información como el conocimiento que ésta contiene, entonces es esencial establecer líneas directrices objetivas y fiables de cómo integrar y sintetizar los resultados procedentes de diferentes investigaciones, pero con objetivos similares. Existen métodos subjetivos, dependientes del conocimiento y la experiencia particular de los experimentadores, para decidir qué información es relevante incluir y con qué peso se valorará. Frente a ellos existen técnicas cuantitativas que permiten sintetizar objetivamente investigaciones individuales [35]. El problema de la experimentación es que difícilmente los resultados de un solo estudio son definitivos, por lo que es necesario hacer réplicas [1; 23]. Cuando el investigador quiere sintetizar los resultados de experimentos con hipótesis similares, los métodos cuantitativos que proporciona la estadística definen un marco para obtener una estimación del efecto del tratamiento (tamaño del efecto) a través de los experimentos, conociendo el tamaño del efecto de cada uno de ellos, de forma objetiva y rigurosa. Los beneficios que proporcionan estas técnicas es que permiten detectar los efectos del tratamiento aunque sean pequeños. Si nos referimos a la Ingeniería del Software Empírico (ISE), detectar una disminución mínima de tiempo en la realización de una tarea que se repite varias veces al día por muchas personas puede suponer un ahorro importante. Desde ese punto de vista el meta-análisis es una bala de plata para el investigador que le permite obtener conclusiones del esfuerzo realizado en decenas de experimentos por otros investigadores.

Pero estas técnicas no son la panacea, puede haber problemas por no aplicarlas debidamente que invalidan los resultados. Los más críticos con estas técnicas hablan de "comparar manzanas y naranjas", y Glass [13] contesta que tienen parte de razón en eso, pero que efectivamente comparar manzanas con manzanas es trivial. El uso de estas técnicas en ISE está aún pendiente, queda camino por recorrer en cuanto a réplicas de experimentos que aporten información sobre las hipótesis con las que se trabaja en la ingeniería del software [19;25].

El resto del documento está estructurado de la siguiente forma, en la sección 2 veremos cuales son las principales técnicas cuantitativas para sintetizar diferentes experimentos, en la sección 3 se describen las limitaciones de estas técnicas y las amenazas a la validez de los resultados cuando no se utilizan adecuadamente. En la sección 4 revisaremos los estudios de meta-análisis realizados en ISE, y en la 5 aplicaremos el meta-análisis en una familia de 5 experimentos que estudia la relación entre la complejidad estructural y el tamaño de los diagramas de clases UML y su complejidad cognitiva y, entre ésta y la entendibilidad y modificabilidad de dichos diagramas. Para finalizar en la sección 6 expondremos las conclusiones.

2. Técnicas Estadísticas para Sintetizar Experimentos

" El meta-análisis se refiere al análisis de análisis… al análisis estadístico de una colección de resultados procedentes de estudios individuales, y cuyo propósito es integrar dichos resultados. Supone una alternativa rigurosa frente a las discusiones meramente narrativas sobre los resultados de una colección de estudios …" [12]

Nos referiremos aquí a algunos de los métodos estadísticos que permiten acumular e interpretar resultados a través de diferentes experimentos, relacionados por investigar hipótesis similares, acumulando así su conocimiento [15; 12; 28; 35; 33].

Existen fundamentalmente tres modos de abordar la síntesis de resultados a través de experimentos individuales, utilizados en otras áreas como medicina o psicología: meta-análisis, combinación de niveles de significación y recuento de votos. A continuación describiremos estas técnicas, las dos primeras con algo más de detalle pues son las que se utilizan con frecuencia en ISE.

2.1. Meta-Análisis

El Meta-análisis tiene como objetivo obtener un tamaño del efecto global, entendiendo como tamaño del efecto "El grado en que el efecto del tratamiento está presente en la población" [5], concepto relevante cuando se habla de potencia de los test estadísticos. Por eso esta técnica se basa en obtener medidas estandarizadas del tamaño del efecto en cada experimento para después combinarlas, obteniendo así una estimación media del tamaño del efecto global. Existen diferentes modos de realizar esa combinación, el más utilizado consiste en ponderar las medidas estandarizadas por una función del tamaño muestral de cada estudio o bien de las varianzas [15; 25]. Una vez calculada la media del tamaño del efecto se interpreta ese valor bien proporcionando un intervalo de confianza, o un p-valor, que permite decidir sobre las hipótesis del meta-análisis.

El meta-análisis comprende tres pasos principales:

1. Decidir qué estudios incluir en él.
2. Estimar el tamaño del efecto para cada estudio.
3. Combinar los tamaños del efecto de cada estudio para estimar el tamaño del efecto global, que puede tener una expresión genérica como en la ecuación 1, y contrastar con él las hipótesis de interés.

$$efecto\_global = \sum_i w_i efecto_i \quad \textbf{(1)}$$

Los dos primeros pasos pueden ser causa de problemas que revisaremos en la sección 3, y en cuanto al tercero, primero hay que elegir el Modelo del Tamaño del Efecto, si es fijo o aleatorio, pues en este último caso hay que añadir una fuente más de variabilidad [24; 8]. Existen dos tipos fundamentales de estimadores del tamaño del efecto, relacionados con la hipótesis que interesa investigar: diferencias entre dos grupos (diferencias de medias) o grado de asociación entre dos variables (coeficiente de correlación). Desde el punto de vista teórico algunos de estos estimadores son equivalentes conceptualmente, pues hay una forma de obtener cada uno a partir de los otros [15; 12;35].

En el ejemplo de la sección 5 vamos a utilizar la *g de Hedges* definida en [15], que puede adoptar diferentes expresiones, todas con las mismas propiedades asintóticas, pero con pesos (w_i) diferentes para ajustar el sesgo cuando se trabaja con muestras pequeñas.

Uno de los mayores problemas que se pueden encontrar al trabajar con estos estimadores tiene que ver con las suposiciones teóricas que se hacen sobre los tipos de distribución (Normal) y la homogeneidad de las varianzas, aunque existen estimadores robustos para soslayar algunos, hay que tenerlos en consideración. La herramienta con la que hemos trabajado [2] permite elegir las diferentes opciones que hemos mencionado.

2.2. Combinación de Niveles

La Combinación de niveles de significación, como su propio nombre indica, permite obtener una medida o estadístico que resume los diferentes niveles de significación (α) de los experimentos. Según sea la estimación obtenida podemos extraer ciertas conclusiones sobre la significación de la hipótesis a través de todos los experimentos. Existen diferentes estadísticos que combinan los niveles de significación, entre ellos los de Fisher basados en una ji-cuadrado, el de Winer basado en una t-student o el de Stouffer similar al anterior pero basado en una distribución Normal [25; 28; 35].

La ventaja es que en esta técnica no hay dependencia del tipo de datos de cada experimento ni de su distribución, pues considera el tamaño del efecto. A cambio su interpretación puede ser difícil o errónea, pues si el nivel de significación global es significativo eso no implica que se acepta la alternativa en cada estudio, sólo que en alguno sí se acepta. Luego realmente no proporciona información sobre la existencia de efecto a través de los estudios.

2.3. Recuento de Votos

El Recuento de votos es una técnica que se basa en obtener una información global sobre los resultados "positivos", "negativos" o "neutros" de cada experimento que se está considerando. Necesitamos conocer, además de las hipótesis, al menos el p-valor o el nivel de significación (α) de los experimentos para poder clasificarlos como positivos si el resultado es significativo, negativo si no los es y neutro en otro caso. La técnica se basa en evaluar la proporción de positivos, rechazando la hipótesis nula, de no efecto del tratamiento, si esa proporción sobrepasa un nivel [15; 25; 33].

3. Amenazas y limitaciones de la síntesis de experimentos

Los resultados obtenidos cuando se sintetiza la información procedente de diferentes experimentos pueden carecer de validez y fiabilidad si no se tienen en cuenta los riesgos y las limitaciones de los mismos, detallados, entre otros, en [21; 35]. Podemos clasificar las amenazas a la validez de estos métodos en cuatro categorías:

1. Los estudios individuales difieren en cuanto a técnicas de medida, definición de variables y tipos de sujetos.
2. Los estudios incluyen experimentos con diseños de alta y baja calidad.

3. Los resultados se pueden sesgar si no se tienen en cuenta estudios cuyos resultados son no significativos, pues éstos no suelen publicarse, lo que afectará sobre todo cuando se hacen revisiones de publicaciones..
4. Cuando del mismo estudio se extraen múltiples resultados, las conclusiones pueden estar sesgadas y parecer más fiables de lo que son porque no hay independencia entre los resultados.

Cuando disponemos de los datos de cada experimento podemos plantearnos trabajar con todos ellos juntos, para así extraer conclusiones "mejores" que estudiándolos por separado, sin embargo eso no es siempre cierto, como lo pone de manifiesto la paradoja de Simpson. Esta paradoja se produce porque la dirección del efecto del tratamiento en cada experimento individual se invierte cuando las muestras se mezclan sin tener en cuenta una variable que influye en los resultados. Más información sobre los problemas del uso y abuso de estas técnicas se pueden encontrar en [4; 13; 24], y en [19; 22; 23] donde se abordan los problemas propios de su aplicación en la ISE. A modo de ejemplo, en [25] abordan el problema de generalizar resultados en ISE, debido a la forma en que usualmente se seleccionan los sujetos y objetos, sin aleatorizar, de la población "general" correspondiente.

4. Uso del Meta-análisis en ISE

En Ingeniería del Software estas técnicas se han utilizado bien para extraer conclusiones en familias de experimentos perfectamente planificados, lo que evita una gran parte de amenazas a la validez de las conclusiones, o bien para realizar revisiones sistemáticas [19] de experimentos publicados sobre ciertos temas:

1. Uso en Familias de experimentos: En [22] se sintetizan los resultados de un estudio sobre el efecto que tenía una herramienta sobre la eficacia y eficiencia de las inspecciones; los tamaños del efecto se obtuvieron a partir de los coeficientes de correlación. Para sacar conclusiones sobre experimentos que evaluaban diferentes técnicas de inspección se utilizó esta técnica en [20;26]. En [14] también se utiliza esta técnica para sintetizar los resultados de un experimento y 4 réplicas que evaluaban técnicas de inspección. En [7] se realiza un meta-análisis para estudiar el efecto de la Programación por Parejas (*Pair Programming*) en la calidad, duración y esfuerzo, el tamaño del efecto se mide con una diferencia de medias, y en [29] se trata de sintetizar diferentes experimentos sobre técnicas de lectura.
2. Uso en revisiones sistemáticas: algunas en [18] se revisan publicaciones de ISE para estudiar el uso que se hace de la potencia de los test estadísticos; en [30] se hace una revisión de 100 proyectos en JAVA para estudiar y sintetizar los resultados individuales sobre el estudio de correlaciones entre diferentes métricas.

5. Aplicación del meta-análisis a una familia de experimentos

Vamos a presentar en esta sección una aplicación del meta-análisis a una familia de experimentos que describiremos en la sección 5.1, en la sección 5.2 presentaremos los resultados de aplicar dicha técnica.

5.1. Una familia de experimentos

Los cinco experimentos controlados se realizaron teniendo en cuenta algunas recomendaciones propuestas en [17, 34]. La Tabla 1 resume las principales características de contexto de los cinco experimentos.

Tabla 1. Características de la familia de experimentos

	#Sujetos	Universidad	Fecha	Curso

E1	72	Universidad de Sevilla (España)	Marzo 2003	4º
R1	28		Marzo 2003	
E2	38	Universidad de Castilla-La Mancha (España)	Abril 2003	3º
R21	23	Universidad de Sannio (Italia)	Junio 2003	4º
R22	71	Universidad de Valladolid (España)	Septiembre 2005	3º

5.1.1 Planificación de los experimentos

En esta subsección definiremos las características comunes a todos los experimentos, consistentes en:

Preparación. La familia tiene un doble objetivo, definido como:

- Objetivo 1: analizar la complejidad estructural y el tamaño de los diagramas de clases UML con respecto a su relación con la complejidad cognitiva desde el punto de vista de los modeladores y diseñadores de software en un contexto académico.
- Objetivo 2: analizar la complejidad cognitiva de los diagramas de clases UML con respecto a su relación con la entendibilidad y modificabilidad desde el punto de vista de modeladores o diseñadores software en un contexto académico.

Definición del contexto: en estos estudios hemos utilizado estudiantes como sujetos experimentales. Las tareas a realizar no requerían altos niveles de experiencia industrial, así que creímos que estos sujetos podían considerarse apropiados, como se apunta en varios trabajos [3, 16]. En resumen, trabajar con estudiantes implica una serie de ventajas, como el hecho de que el conocimiento previo es bastante homogéneo, la disponibilidad de un elevado número de sujetos, y la posibilidad de probar diseños experimentales e hipótesis iniciales [31]. Una ventaja más es el uso de principiantes como sujetos en experimentos en entendibilidad es que la complejidad cognitiva del sujeto bajo estudio no es tan alta como la de sujetos con experiencia.

Material: los materiales experimentales consisten en un conjunto de diagramas de clases UML que cubren un amplio rango de valores las medidas presentadas en la Tabla 5 del Apéndice A. Así, obtuvimos tres tipos de diagramas: difíciles de mantener (D), fáciles de mantener (F) y de una dificultad mediana (M). Algunos fueron diseñados específicamente para los experimentos y otros se obtuvieron de aplicaciones reales. Cada diagrama tiene documentación añadida que contiene, además de otras cosas, cuatro tareas de comprensión y cuatro de modificación.

5.1.2 Conducción de los experimentos individuales

Las variables consideradas para medir la complejidad estructural y el tamaño fueron un conjunto de 11 medidas presentadas en la Tabla 5 (Apéndice A). La medida *CompSub* es la percepción subjetiva de los sujetos sobre la complejidad de los diagramas con los que trabajaron durante la tarea experimental. Nosotros consideramos *Compsub* como una medida de la complejidad cognitiva. Los posibles valores de esta variable son: Muy sencillo, Moderadamente sencillo, Medio, Moderadamente complejo y Muy complejo. Para medir la entendibilidad y modificabilidad de los diagramas consideramos el tiempo (en segundos) utilizado por cada sujeto para completar las preguntas de entendibilidad y modificabilidad. Hemos llamado a estas medidas Tiempo de entendibilidad y modificabilidad.

Utilizamos un diseño balanceado inter-sujetos, así cada sujeto trabajó con un único diagrama. Los diagramas se asignaron aleatoriamente y cada diagrama fue asignado al mismo número de sujetos.

Formulamos las siguientes hipótesis, que se derivan de los objetivos de la familia, previamente presentados:

- $H_{0,1}$: la complejidad estructural y el tamaño de los diagramas de clases UML no está correlacionado con la complejidad cognitiva. $H_{1,1}$:$\neg H_{0,1}$
- $H_{0,2}$: La complejidad cognitiva de los diagramas de clases UML no está correlacionada con su entendibilidad y modificabilidad. $H_{1,2}$:$\neg H_{0,2}$

Todos los experimentos fueron supervisados y limitados en el tiempo. Se pueden encontrar más detalles en [9, 10].

Utilizamos SPSS [32] para realizar todos los análisis estadísticos y la herramienta Comprehensive Meta Analysis [2] para realizar el meta-análisis.

5.1.2.1 Experimento 1 (E1) y réplica (R1)

Hemos estudiado las hipótesis utilizando el coeficiente de correlación de Spearman, obteniendo las siguientes conclusiones relacionadas con los dos objetivos de la familia de experimentos:

- Objetivo 1: la complejidad estructural y la complejidad cognitiva presentan una correlación positiva significativa para todas las medidas, con la excepción de NM, NGEN y MAXDIT en R1.
- Objetivo 2: la complejidad cognitiva parece estar correlacionada positivamente con el esfuerzo necesario para comprender los diagramas de clases UML, pero los resultados son significativos sólo para E1. Al mismo tiempo, no hay correlación con el esfuerzo necesario para modificar los diagramas. Una posible explicación para esto puede ser que los sujetos basen su percepción en la dificultad de la primera prueba que realizan, que en este caso es la de comprensión.

5.1.2.2 Experimento 2 (E2) y sus réplicas (R21 y R22)

En estos estudios, los objetivos y variables fueron los mismos que en los descritos previamente, pero los diagramas utilizados fueron diferentes, y el contexto y diseño fueron mejorados. Se puede encontrar información más detallada en [10].

De nuevo, hemos utilizado a sujetos elegidos por conveniencia, pero en este caso se ha mejorado la selección bloqueando la experiencia de los sujetos. Realizamos un test previo, y con los resultados obtenidos dividimos a los sujetos en dos grupos. Cada diagrama fue asignado al mismo número de sujetos en cada grupo. Se pueden encontrar más detalles de este proceso en [10].

Las medidas de entendibilidad y modificabilidad sólo se incluyeron cuando la realización de las tareas tenía un nivel de calidad mínimo (corrección y completitud). Los sujetos que estuvieron por debajo del 75% en corrección y completitud fueron excluidos del estudio.

Después de comprobar las hipótesis formuladas y relacionándolas con los objetivos de la familia de experimentos, concluimos que:

- Objetivo 1: tenemos resultados favorables que admiten una correlación entre la complejidad estructural y la complejidad cognitiva de los diagramas de clases UML. Muchas de las medidas están significativamente correlacionadas con la complejidad subjetiva en diferentes estudios, especialmente en aquellos relacionados con las jerarquías de herencia (ver Tabla 2).
- Objetivo 2: los resultados también están a favor de la hipótesis que relaciona la complejidad cognitiva con la entendibilidad de los diagramas de clases UML.

Tabla 2. Medidas correlacionadas en E1, R21 y R22

Estudio	Medidas correlacionadas significativamente
E2	NC, NAssoc, NGen, NGenH, MaxDIT (5 sobre 11 medidas)
R21	Todas excepto NM, NGenH y MaxAgg (8 sobre 11 medidas)
R22	Todas excepto NM (10 sobre 11 medidas)

5.2 Estudio de Meta-análisis

Hemos utilizado el meta-análisis para extraer conclusiones globales de la familia de experimentos presentada anteriormente. Como ya hemos mencionado, necesitamos estandarizar los tamaños de los efectos y para ello debemos elegir un estadístico que los mida, en este caso hemos utilizado coeficientes de correlación para, a partir de ellos, obtener la métrica del efecto global medida con la *g de Hedges* adecuada cuando los tamaños muestrales no son grandes. Además en [18] se clasifica el tamaño del efecto global

en pequeño, medio o grande basándose en una revisión de estudios en ISE. En este ejemplo, aunque los diseños experimentales no son exactamente los mismos, consideramos que los 5 estudios cumplen todas las condiciones para incorporarlos en el meta-análisis salvando las amenazas para que los resultados sean válidos, así pues trabajaremos con 5 estimaciones para construir el tamaño del efecto global.

Primeramente, realizamos un meta-análisis para estudiar la correlación entre cada par Medida-*CompSub*. En la Tabla 3 presentamos la estimación global de los coeficientes de correlación, con un intervalo de confianza del 95%, el *p-valor* y el valor de la *g de Hedges*, incluyendo una clasificación del tamaño del efecto como grande (G), mediano (M) o pequeño (P).

Los resultados obtenidos están a favor de la existencia de una correlación positiva entre la complejidad cognitiva y las 11 medidas que miden la complejidad estructural y el tamaño de los diagramas de clases UML. De hecho, muchos de los tamaños del efecto son medios o grandes, exceptuando el de NM que es pequeño. Las medidas de tamaño que tienen más influencia en la complejidad cognitiva son NC y NA, mientras que las medidas de complejidad que tienen más influencia en la complejidad cognitiva son las relacionadas con las asociaciones (NAssoc) y generalizaciones (NGen y MaxDIT). Así que los diagramas de clases UML con mayor número de clases y atributos tienen mayor complejidad cognitiva; además, cuanto mayor número de asociaciones, generalizaciones o profundidad de herencia tenga un diagrama de clases mayor será su complejidad cognitiva.

Respecto de las hipótesis derivadas del objetivo 2, la Tabla 4 muestra que podemos admitir la existencia de correlación entre la complejidad cognitiva y la entendibilidad y la modificabilidad de los diagramas de clases UML. Los tamaños del efecto son medios en ambos casos, pero la estimación de la correlación de la entendibilidad es mayor que la correlación de la modificabilidad. Podemos, por tanto, concluir que cuanta mayor complejidad cognitiva contiene un diagrama, más difícil será comprenderlo o modificarlo.

Tabla 3. Meta-análisis para las correlaciones entre Medidas-*CompSub*

H0: ρ<=0	Correlación (ρ) Tamaño del Efecto Global	Límite inferior	Límite superior	p-valor	g de Hedges
NC	0.566	0.464	0.653	0.0000	1.322(G)
NA	0.541	0.435	0.632	0.000	1.219(G)
NM	0.177	0.040	0.307	0.012	0.339(P)
NAssoc	0.566	0.465	0.653	0.000	1.318(G)
NAgg	0.481	0.368	0.581	0.000	1.051(M)
NDep	0.484	0.371	0.584	0.000	1.060(M)
NGen	0.484	0.371	0.584	0.000	1.018 (G)
NGenH	0.422	0.302	0.529	0.000	0.903 (M)
NAggH	0.393	0.270	0.504	0.000	0.814 (M)
MaxDIT	0.492	0.379	0.590	0.000	1.080 (G)
MaxHAgg	0.360	0.233	0.474	0.000	0.734 (M)

Tabla 4. Meta-análisis para las correlaciones *CompSub*-Tiempo de Entendibilidad y Modificabilidad

H0: ρ<=0	Correlación (ρ) Tamaño del Efecto Global	Límite inferior	Límite superior	p-valor	g de Hedges
Tiempo de Entendibilidad	0.330	0.200	0.449	0.000	0.684 (M)
Tiempo de Modificabilidad	0.186	0.044	0.320	0.011	0.368(M)

En la Figura 2 aparece el meta-análisis de la correlación entre un par de medidas y la medida *CompSub*, y entre su entendibilidad y su complejidad cognitiva, obtenido con la herramienta utilizada [4].

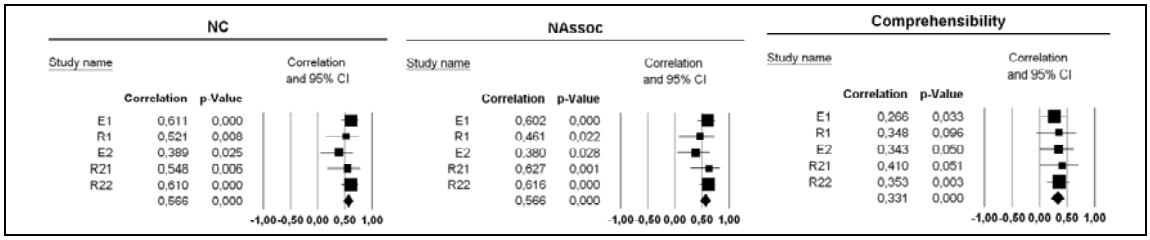

Fig. 2. Meta-análisis para NC-*CompSub*, NAssoc-*CompSub* y *CompSub*-Tiempo de entendibilidad

Los resultados del meta-análisis son relevantes porque podrán utilizarse como medios para controlar el nivel de ciertos atributos de calidad de los diagramas de clases UML durante la fase de modelado. Además, tienen implicaciones tanto prácticas como pedagógicas, proporcionando información sobre cuales son los constructores de UML que tienen más influencia sobre el esfuerzo para comprender y modificar los diagramas de clases UML. Por ello cuando existen diseños alternativos de un diagrama de clases UML, este conocimiento puede ser útil para seleccionar el que minimice estos constructores.

6 Conclusiones

Hemos presentado algunas de las técnicas que permiten sintetizar los resultados de varios experimentos de forma objetiva, facilitando así la acumulación y el refinamiento del conocimiento que se quiere conseguir cuando se propone una hipótesis de estudio. En áreas como la medicina y la psicología la experimentación y la síntesis de resultados se está utilizando desde hace décadas, luego en esas áreas ya se han enfrentado a problemas que los investigadores de ISE podemos aprovechar [4;24]. En [25] se muestra uno de los riesgos propio de ISE: la generalización de resultados es más que conflictiva, pues la elección de sujetos, en general, es por conveniencia, y los objetos proceden sólo de ciertos dominios.

En Ingeniería del Software el meta-análisis no se ha utilizado demasiado, y sobre todo el uso ha sido para comparar diferentes técnicas de revisión de software y para hacer revisiones sistemáticas de algunos temas [20; 22; 23; 14; 29], por lo que aún quedan ámbitos e hipótesis por explorar con esta técnica: efecto del uso de herramientas y nuevas tecnologías en atributos de calidad del software, o sobre la productividad en ciertas fases de desarrollo del software, entre otros. Nuestra intención es seguir utilizando esta técnica en experimentos realizados con diagramas de estado [6] y las expresiones OCL [27], pues estamos convencidos de que el meta-análisis es una herramienta útil si se realiza en determinadas condiciones.

Agradecimientos

Esta investigación forma parte del proyecto ESFINGE (TIN2006-15175-C05-05) financiado por el Ministerio de Educación y Ciencia (España), y del proyecto IDONEO (PAC08-0160-6141) financiado por la Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha (España).

Referencias

1. Basili, V., Shull, F. y Lanubile, F.: Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering, 25, pp. 456-473. 1999.
2. Biostat, Comprehensive Meta-Analysis v2, 2006. http://www.meta-analysis.com/.
3. Briand, L., Bunse, C. y Daly, J.: A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. IEEE Transactions on Software Engineering, 27(6), pp. 513-530. 2001.
4. Brooks, A. Meta Analysis- A silver Bullet- for Meta-Analysts. Empirical Software Engineering, 2 333-338. 1997.
5. Cohen J. Statistical power analysis for the behavioural sciences second ed. Lawrence Erlbaum Associates, Publishers London. 1988.
6. Cruz-Lemus, J. A.., Genero, M. y Piattini, M.: Metrics for UML Statechart Diagrams. In: Metrics for Software Conceptual Models, Imperial College Press, UK., 2005.
7. Dybå, T., Arisholm, E., Sjøberg, D. I. K., Hannay, J. E. y Shull, F.: Are Two Heads Better than One? On the Effectiveness of Pair Programming. IEEE Software, 24(6), pp. 10-13. 2007.
8. Field, A.P. Meta-Analysis of Correlation Coefficients. Phsychological Methods, 6 (2) 161-180. 2001.
9. Genero, M., Manso, M. E. y Piattini, M.: Early Indicators of UML Class Diagrams Understandability and Modifiability. ACM-IEEE International Symposium on Empirical Software Engineering, pp. 207-216. 2004.
10. Genero, M., Manso, M. E., Visaggio, A., Canfora, G. y Piattini, M.: Building Measure-Based Prediction Models for UML Class Diagram Maintainability. Empirical Software Engineering, 12, pp. 517-549. 2007.
11. Genero, M., Poels, G., Manso, M. E. y Piattini, M.: Defining and Validating Metrics for UML Class Diagrams. in Metrics for Software Conceptual Models, Imperial College Press, UK., 2005.
12. Glass, G. V., McGaw, B., and Smith, M. L. Meta-Analysis in Social Research. Sage Publications. 1981.
13. Glass, G.V. http://glass.ed.asu.edu/gene/papers/meta25.html. (16-7-2008)
14. Hayes, W.: Research in Software Engineering: a Case for Meta-Analysis. 6th IEEE International Symposium on Software Metrics (METRICS'99), pp. 143-151. 1999.
15. Hedges, L. V. y Olkin, I.: Statistical Methods for Meta-Analysis, Academia Press, 1985.
16. Höst, M., Regnell, B. y Wohlin, C.: Using Students as Subjects - a Comparative Study of Students & Proffesionals in Lead-Time Impact Assessment. 4th Conference on Empirical Assessment & Evaluation in Software Engineering (EASE 2000), pp. 201-214. 2000.
17. Juristo, N., y Moreno, S.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers, 2001.
18. Kampenes, V., Dybå, T., Hannay, J. E. y Sjoberg, D. I. K.: A Systematic Review of Effect Size in Software Engineering Experiments. Information and Software Technology, 49(11-12), pp. 1073-1086. 2007.
19. Kitchenham, Barbara. Procedures for Performing Systematic Reviews, Joint Technical Rreport, Keele University TR/SE-0401 and NICTA 0400011T.1, July 2004.
20. Laitenberger, O., El-Emam, K. y Harbich, T.: An Internally Replicated Quasy-Experimental Comparison of Checklist and Perspective-based Reading of Code Documents. IESE, 006.99/e, 1999.
21. Lipsey, M. y Wison, D.: Practical Meta-Analysis, Sage, 2001.
22. Miller, J. y McDonald, F.: Statistical Analysis of Two Experimental Studies. University of Strathclyde, EFoCS-31-98, 1998.
23. Miller, J. Applying Meta-Analytical Procedures to Software Engineering Experiments. Journal of Systems and Software, 54: 29-39. 2000.
24. Pai, Madhukar, McCullovch, Michael, Gorman, Jennifer D., Pai, Nitika, Enanoria, Wayne, Kennedy, Gail, Tharyan, Prathap, Colford, John M. Jnr. Systematic reviews and meta-analysis: An illustrated, step-by-step guide. The National medical Journal of India, 17(2) 2004, pp 86-95.
25. Pickard, M.: Combining Empirical Results in Software Engineering. University of Keele, T-R V1, 2004.
26. Porter, A. y Johnson, M.: Assessing Software Review Measurement: Necessary and Sufficient Properties for Software Measures. Information and Software Technology, 42(1), pp. 35-46. 1997.
27. Reynoso, L., Genero, M. y Piattini, M.: Measuring OCL Expressions: An approach based on Cognitive Techniques. in Metrics for Software Conceptual Models, Imperial College Press, UK., 2005.

28. Rosenthal, R. Meta-Analytic Procedures for Social Research. Sage Publications. 1986..
29. Shull, F., Carver, J., Maldonado, J., Conradi, R., Basili, V. Replicated Studies: Building a Body of Knowledge about Software Tecniques. In N. Juristo y A. Moreno (Eds). Lecture Notes on Empirical Software Engineering (pp 39-84). Singapore. World Scientific. 2003.
30. Succi, G. , Spasojevic, R., Hayes, J.J., Smith, M.R., Pedrycz, W. Application of Statistical Meta-Analysis to Software Engineering Metrics Data. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, vol 1 709-714. 2000.
31. Sjoberg, D. I. K. , Hannay, J. E., Hansen, O., Kampenes, V., Karahasanovic, A., Liborg, N. K. y Rekdal, A. C.: A Survey of Controlled Experiments in Software Engineering. IEEE Transactions on Software Engineering, 31(9), pp. 733-753. 2005.
32. SPSS, SPSS 12.0, Syntax Reference Guide, 2003.
33. Sutton, J. A., Abrams, R. K., Jones, R. D., Sheldon, A. T., and Song, F. Methods for Meta-Analysis in Medical Research. John-Wiley & Sons. 2001.
34. Wohlin, C., Runeson, P., Hast, M., Ohlsson, M. C., Regnell, B. y Wesslen, A.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publisher, 2000.
35. Wolf, F. M. Meta-Analysis: Quantitative Methods for Research Synthesis. Sage Publications. 1986.

Apéndice A

Después de estudiar el metamodelo UML y revisar la literatura existente sobre medidas, propusimos un conjunto de medidas para la complejidad estructural de los diagramas de clases UML [11] (ver Tabla 5). Las medidas propuestas están relacionadas con el uso de las relaciones de UML, como asociaciones, dependencias, agregaciones y generalizaciones.

Tabla 5. Medidas para diagramas de clases UML

	Nombre	Descripción
Medidas de tamaño	Número de clases (NC)	El número total de clases en un diagrama de clases.
	Número de atributos (NA)	El número de atributos definidos en todas las clases en un diagrama de clases (no incluye el número de atributos heredados o atributos definidos en métodos). Esto incluye atributos definidos en una instancia.
	Número de métodos (NM)	El número total de métodos definidos en todas las clases de un diagrama de clases, no incluyendo métodos inherentes (porque podrían ser contados doblemente). Esto incluye métodos definidos en una clase instanciada.
Medidas de complejidad estructural	Número de asociaciones (NAssoc)	El número total de relaciones de asociación en un diagrama de clases.
	Número de agregaciones (NAgg)	El número total de relaciones de agregación (cada pareja de elementos en una relación de agregación).
	Número de dependencias (NDep)	El número total de relaciones de dependencia.
	Número de generalizaciones (NGen)	El número total de relaciones de generalización (cada pareja padre-hijo en una relación de generalización).
	Número de jerarquías de generalización (NGenH)	El número total de jerarquías de generalización, esto es, contar el número total de estructuras con relaciones de generalización.
	Número de jerarquías de agregación (NAggH)	El número total de jerarquías de agregación, esto es, contar el número total de estructuras todo-partes en un diagrama de clases.
	Máximo DIT (MaxDIT)	El máximo valor DIT obtenido para cada clase en un diagrama de clases. El valor DIT para una clase en una jerarquía de generalización es el mayor camino desde la raíz de la jerarquía (Chidamber y Kemerer, 1994).
	Máximo HAgg (MaxHAgg)	El máximo valor HAgg obtenido para cada clase en un diagrama de clases. El valor HAgg de una clase en una jerarquía de agregación es el mayor camino de la clase a las hojas.

Pai, Madhukar, McCullovch, Michael, Gorman, Jennifer D., Pai, Nitika, Enanoria, Wayne, Kennedy, Gail, Tharyan, Prathap, Colford, John M. Jnr. Systematic reviews and meta-analysis: An illustrated, step-by-step guide. The National medical Journal of India, 17(2) 2004, pp 86-95.

Hacia un modelo híbrido de simulación de la producción de software en un entorno multiproyecto

Javier Navascués Fernández-Victorio[1], Isabel Ramos Román[2] y Miguel Toro Bonilla[2]

[1] Departamento de Organización Industrial y Gestión de Empresas, Universidad de Sevilla;
[2] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla
[1]jnavascues@us.es

Abstract. La simulación del ciclo de vida de los proyectos software o de partes de éste es un activo campo de investigación en la ingeniería del software. El presente informe analiza la literatura sobre modelos de simulación que puedan se útiles para estudiar las organizaciones de desarrollo de software. La finalidad del trabajo es construir un modelo de simulación en el que se puedan implementar diversas políticas de asignación de recursos basadas en las técnicas de gestión de proyectos propias de la investigación operativa en un contexto de mejora de procesos y teniendo en cuenta el carácter multiproyecto.

La simulación de proyectos software

La simulación de proyectos software es un campo de investigación que registra un importante nivel de actividad. En la tesis de Mercedes Ruiz Carreira (18) se recoge una relación de trabajos de simulación del proceso software entre los años 1991 y 2000. En el origen de todos ellos está el trabajo de Abdel-Hamid y Madnick (2) que supuso un exhaustivo intento de modelar el proceso de inspección formal con un grado de detalle muy elevado. Reúne las actividades relacionadas con la gestión de recursos humanos, la producción, la planificación y el control en un modelo de Dinámica de Sistemas que se ha convertido en una referencia en este campo de investigación. A partir de ahí en la misma fuente se recogen referencias a una serie de trabajos que han ido enfocando diferentes perspectivas y problemas dentro del marco general antes citado.

Con posterioridad a la publicación de este trabajo, la producción no ha cesado, enriqueciéndose el campo de la producción de modelos y acometiéndose esfuerzos de modelado híbrido dentro del campo de la simulación e integrándolo con métodos y herramientas de otros campos, contemplándose:

- El empleo simultáneo de varios paradigmas de modelado (continuo y discreto, fundamentalmente)
- La integración entre la simulación y la aplicación de métodos y modelos de optimización del campo de la Investigación Operativa y de la Inteligencia Artificial

- La integración de la simulación dentro de herramientas de ayuda a la decisión en tiempo de ejecución básándose en la metodología de agentes
- La integración de la simulación en modelos generales de representación y explotación del conocimiento

Desde el punto de vista académico y profesional la referencia fundamental es la serie de talleres PROSIM (*Workshop of Software Process Simulation and Modeling*) iniciada en 1998 y fusionada a partir de 2006 con el taller internacional SPW (*Software Process Workshop*), que se celebra en el marco de las conferencias ICSE (*International Conference on Software Engineering*) organizadas por la SCM y el IEEE (7,15,16,20).

El entorno multiproyecto

El entorno multiproyecto, con sus elevadas dependencias, está naturalmente inclinado a multiplicar los efectos cruzados de los diferentes bucles de realimentación que gobierna la evolución del estado de la organización de desarrollo. Es significativo, por tanto, que el esfuerzo por la simulación de este tipo de entorno sea relativamente reducido en comparación con el caso de un único proyecto. El propio Abdel Hamid (1) llama la atención sobre lo inapropiados que pueden ser los métodos y modelos de estimación cuando no tienen en cuenta las implicaciones a escala de toda la organización.

En el artículo citado (1) se presenta un trabajo en el que se modela específicamente la transferencia de personal entre varios proyectos que se están desarrollando en paralelo. La transferencia obedece a una serie de reglas relacionadas con la presión del plazo. Se realizan dos experimentos: uno de aceleración de los plazos y otro en el que se emula la regla MINSLACK para asignar recursos de la organización a los proyectos que compiten por ellos.

En (17) se presenta un modelo de dinámica de sistemas en el que se modela un ciclo de vida incremental en términos que conlleva relaciones de concurrencia entre diferentes paquetes de trabajo que, a otra escala, equivalen a la competencia por recursos de varios proyectos simultáneos, con la complicación adicional que dicha interrelación se establece no sólo por esa competencia por los recursos sino también por la información que fluye de un paquete de trabajo a otro, acoplándolos por tanto también a través de ese mecanismo.

Los autores proponen un modelo sencillo basado en una estructura simplificada del modelo de Abdel Hamid semejante a la recogida en (19), que relaciona recursos, tiempo y esfuerzo con el paquete de trabajo. Esta sencilla estructura se denomina el *triángulo del proceso*. Este triángulo puede definirse, modularmente, a diferentes niveles jerárquicos: paquetes de trabajo, fases, entregas, proyectos y organización en su conjunto.

Para acoplarlos diferentes módulos se emplea tanto la asignación de recursos a módulos como el acoplamiento de los trabajos. El primero es una regla de reparto de recursos entre paquetes de trabajo (o módulos de nivel superior) en base a prioridades y presiones de plazo. El acoplamiento de los trabajos implica tratar el trabajo de *corrección de errores* y las *transiciones*. La corrección de errores puede aparecer

dentro de un mismo paquete de trabajo o en otro subsiguiente. Trasladando la corrección de errores a una entrega posterior se *externalizan* estas tareas. Las transiciones se producen cuando una fase madura lo suficiente como para iniciarse la siguiente. Las transiciones pueden producirse de diversas formas: mediante *hitos* o por *unidades* de trabajo que al concluirse se convierten en *inputs* de las siguientes. Esto permite modelar la superposición de fases. El objetivo último del modelo es analizar las estrategias de desarrollo incremental, tanto a nivel previo (planificación) como su implementación y su modificación en tiempo de ejecución.

La asignación de los recursos a los proyectos

En la literatura revisada aparecen básicamente tres enfoques:

1. Generar una solución a partir de un modelo de investigación operativa y simular luego las consecuencias de su aplicación, generalmente por medio del método de Montecarlo.
2. Generar el espacio de estados a través del proceso de simulación y luego aplicar un heurístico de la investigación operativa para optimizar el estado del sistema.
3. Modelar el decisor explícitamente como un agente dotado de las reglas heurísticas; en este caso se trata de una herramienta de apoyo a la dirección del proyecto para que evalúe las consecuencias de las decisiones adoptadas de forma dinámica

Los heurísticos aplicados son de diferentes tipos:

- reglas simples de prioridad
- programación dinámica (métodos aproximados)
- algoritmos de búsqueda \emph{greedy} (miopes)
- algoritmos genéticos
- *sequeaky wheel optimizaton*; un algoritmo de búsqueda en el espacio de las estrategias de prioridad

Antonio *et al*, (4), presentan una aproximación al problema basada en la teoría de colas y la simulación estocástica para el apoyo a la planificación, gestión y control de la asignación de personal a un proyecto de mantenimiento multifase. Se trata de la adaptación al efecto 2000, *Y2K*, de un gran sistema de información de una entidad financiera desarrollado en COBOL/JCL que requiere la constitución de diferentes equipos trabajando en varios centros geográficamente distribuidos. Se estudia una configuración centralizada desde el punto de vista del cliente así como otras configuraciones más desagregadas. Los métodos de teorías de colas y la simulación estocástica se emplean para evaluar las políticas de asignación de personal a los equipos y la posibilidad de alcanzar los hitos previstos para la ejecución del proyecto.

En otro segundo artículo (3) se emplean los *algoritmos genéticos* para generar soluciones para el problema de teoría de colas teniendo en cuenta la necesidad de corregir errores y de reprogramar y abandonar algunas de las tareas inicialmente previstas. La simulación de Montecarlo de las colas junto con los algoritmos genéticos permite generar secuencias y asignar personal a los equipos. El impacto posible del trabajo de corrección de errores, el abandono de tareas y los errores e

incertidumbres en las estimaciones iniciales se caracteriza con funciones de distribución que se utilizan como entradas para la simulación.

Padberg, en varios artículos(9, 10, 11, 12, 13, 14) presenta su trabajo en el que se modelan como problemas *markovianos* de decisión en los que se tienen en cuenta, además de la variación estocástica de la duración de las actividades, la posibilidad de vueltas atrás y retrasos. Para ello se generan políticas de secuenciación con un algoritmo aproximado de *programación dinámica*.

Con un conjunto de problemas similares pero que se diferencian en la necesidad mayor o menor de recursos especializados y el grado de integración entre los componentes del producto, se simula la aplicación de la política óptima generada por el algoritmo con las políticas clásicas de listas. Se comprueba que cuanto mayor es la necesidad de recursos especializados y cuanto menor es el grado de integracion entre los componentes, más destaca la solución propuesta sobre las reglas de listas.

Browning *et al*, (21) presentan un modelo se basa en la *design structure matrix*. La varianza en la duración y el coste en los proyectos de desarrollo de producto son atribuibles según su criterio en gran medida a las iteraciones, algo que es común también en el proceso software. Dichas iteraciones pueden o no ocurrir dependiendo del comportamiento de una serie de variables que el modelo simula como variables estocásticas cuya probabilidad de ocurrencia depende de la existencia de determinados paquetes de información que activan un proceso de reelaboración. Un proceso de reelaboración puede, a su vez, desencadenar otros sucesivos afectando así al proyecto en su totalidad. Para hacer frente a este problema utilizan el método de Montecarlo para simular políticas de secuenciación robustas generadas por un algoritmo genético.

Özdamar (8) presenta una propuesta de aplicación de reglas de prioridad en proyectos definidos con el formalismo *fuzzy*. Presenta un heurístico para la programación de las actividades de un modelo de desarrollo en cascada que se modela utilizando la lógica borrosa. la duración de las actividades depende del modo de ejecución seleccionado representado por una función de pertenencia trapezoidal. También se modela la disponibilidad de los recursos diferenciando entre dos categorías, una de personal más cualificado que comparte varias tareas y - por tanto - asignable de forma continua, y otra categoría de personal que no realiza multitareas y que se presenta en unidades discretas.

El objetivo del modelo es minimizar el tiempo de realización. Para ello se emplea un algoritmo de generación de secuencias en serie y se ensayan diferentes reglas de prioridad entre las más sencillas empleadas en los problemas de secuenciación. El modelo permite la resecuenciación dinámica para escoger las variaciones en las estimaciones a lo largo de la ejecución del proyecto.

En (5) Joslin presenta un modelo basado en agentes que simula de forma dinámica la reasignación de personal a las tareas en un proyecto con diferentes funcionalidades que deben entregarse en un plazo fijo.

Se trata de un sistema de ayuda a la decisión (DSS) basado en un heurístico denominado *Squeaky Wheel* que explora en el espacio de las estrategias en lugar de en el espacio de las soluciones. El simulador puede implementar diferentes reglas de asignación si bien la versión que se presenta en el artículo sólo presenta los primeros resultados para reglas muy sencillas.

La mecánica básica es la de reasignar el personal entre tareas conforme se aproximan los plazos y se detecta el riesgo de no cumplirlos. El modelo contempla el efecto aprendizaje cuando se produce un cambio de tarea y también el efecto de la sobrecarga de comunicación por lo que incorpora no linealidades el tipo de las que se presentan en los modelos clásicos de dinámica de sistemas. Los autores anuncian la futura implementación de reglas más sofisticadas, como las basadas en algoritmos genéticos.

Propuesta de un modelo híbrido multiproyecto

En el contexto descrito hasta aquí se encuentra a faltar una propuesta capaz de integrar los siguientes aspectos:

- El carácter multiproyecto de la producción de software
- El impacto de las actividades de medición y mejora de procesos en el comportamiento de los proyectos y del conjunto
- La implementación de estrategias de asignación de recursos sofisticadas

La propuesta es desarrollar un modelo dinámico de simulación jerárquico capaz de integrar estas facetas. El modelo de simulación será de tipo híbrido por la necesidad de capturar con la granularidad necesaria procesos y sucesos de naturalezas muy diferentes. Se pretende con ello poder modelar simultáneamente:

- El carácter discreto de:
  - El ciclo de vida de un producto software caracterizado por la sucesión de fases definidas y caracterizadas cada una de ellas por sus correspondientes entregables e hitos
  - Las decisiones de asignación (y reasignación) de recursos y programación (y reprogramación) de actividades
  - El proceso de registro para la obtención de métricas necesarias para la adopción de decisiones
  - Los eventos externamente determinados o derivados de las decisiones del punto anterior que pueden afectar al ciclo de vida o a cualquiera de sus fases
- El carácter continuo de:
  - El proceso de trabajo con el correspondiente consumo de esfuerzo y la eventual generación de errores
  - El entorno general de la empresa (movimiento laboral, ciclos de capacitación del personal, desarrollo de infraestructuras de soporte a los procesos de producción, etc.)
  - La mejora de las capacidades de los procesos

Si bien el problema específico que se desea modelar en esta investigación no tiene una correspondencia clara en la literatura, sí que hay en las referencias revisadas elementos suficientes con los que construir un modelo híbrido que permita representarlo. Los componentes serían:

- **Los modelos reducidos como ``átomos''.** El modelo presentado en (19) puede servir para constituir el núcleo básico de la simulación de la evolución de las tareas. Su integración en un marco más amplio obligará a redefinir los bucles de

realimentación que se cierran en el propio átomo y cuales traspasan las frontreas de estos.

- **Los triángulos de procesos**. La estructura de integración jerárquica de los \emph{triángulos de procesos} descritos por Powell permite a la vez modelar la jerarquía tareas - fases - proyectos - cartera y representar, a través de los tres primeros niveles un modelo de desarrollo incremental, que puede reducirse al modelo en cascada con facilidad.
- **La mejora del proceso como tarea del proceso**. La incorporación de las actividades de ingeniería y soporte como tareas específicas del proyecto como se propone en (23) permite modelar explícitamente el *trade-off* entre costes y beneficios de la mejora de procesos.
- **La orientación a agentes**. El modelo presentado por Joslin, orientado a la búsqueda en el espacio de las estrategias de priorización de las actividades permite simular las reglas heurísticas de asignación de recursos.

El formalismo DEVS, *discrete event system specification*, propuesto por Ziegler (22) posibilita representar con naturalidad un sistema híbrido al gestionar conjuntamente una simulación continua con un manejo muy flexible de la lógica de eventos y los flujos de información asociados al inicio y finalización de tareas y a la monitorización de las mismas. Este formalismo presenta además la ventaja de que el modelo básico, basado en un sistema muy simple denominado el DEVS *atómico* y en un acoplamiento elemental de dos átomos, es escalable y muy fácilmente modelable con la tecnología orientada a objetos. Esto permite generar una clase de modelos fácilmente parametrizables y reutilizables en el que se pueden diferenciar, y a la vez mantener coherentes, los procesos continuos del entorno de la organización y del propio desarrollo de las tareas elementales incluyendo los efectos de realimentación, y los procesos discretos tanto de flujos materiales (entregables, módulos, ...) como de medida, control y decisión.

Este trabajo se ha realizado dentro del proyecto “Calidad Predecible y gestionada mediante simulación y técnicas de prueba en etapas tempranas” (TIN 2007-67843-C06-03)

Bibliografía

(1) AbdelHamid TK. A Multiproject perspective of single-project dynamics. J.Syst.Software 1993 sep,;22(3):151-165.

(2) AbdelHamid TK, Madnick SE. Software Project Dynamics An Integrated Approach. Englewood Cliffs NJ: Prentice Hall; 1991.

(3) Antoniol G, Cimitile A, Di Lucca GA, Di Penta M. Assessing staffing needs for a software maintenance project through queuing simulation. IEEE Trans Software Eng 2004;30(1):43-58.

(4) Antoniol G et al, A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. Proceedings - 10th International Symposium on Software Metrics, METRICS 2004; 14 September 2004 through 16 September 2004; ; 2004.

(5) Joslin, D.., Poole, W., Agent-based simulation for software project planning. Winter Simulation Conference; 2005.

(6) Meier C, Yassine AA, Browning TR. Design Process Sequencing With Competent Genetic Algorithms. Transactions of the ASME 2007;129:566.
(7) Münch J, Pfahl D. Special issue on ProSim 2005, The 6th international workshop on software process simulation and modeling, St. Louis, Missouri, USA may 2005. Software Process: Improvement and Practice 2006;11(4):339-343.
(8) Özdamar L, Alanya E. Uncertainty Modelling in Software Development Projects (With Case Study). Annals of Operations Research 2001;102:157-178.
(9) Padberg F. On the potential of process simulation in software project schedule optimization. Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International 2005;2:127-130 Vol. 1.
(10) Padberg F. Computing optimal scheduling policies for software projects. Software Engineering Conference, 2004. 11th Asia-Pacific 2004:300-308.
(11) Padberg F. Linking software process modeling with Markov decision theory. Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International 2004;2:152-155 vol.2.
(12) Padberg F. A software process scheduling simulator. Software Engineering, 2003. Proceedings. 25th International Conference on 2003:816-817.
(13) Padberg F. Using process simulation to compare scheduling strategies for software projects. Software Engineering Conference, 2002. Ninth Asia-Pacific 2002:581-590.
(14) Padberg F. A study on optimal scheduling for software projects. Software Process: Improvement and Practice 2006;11(1):77-91.
(15) Pfahl D, Raffo DM, Rus I, Wernick P editors. ProSim 05. The 6th International Workshop on Software Process Simulation and Modeling. Proceedings : May 14-15, 2005, Adams Mark Hotel, St. Louis, Missouri, USA. : Fraunhofer IRB Verlag, Stuttgart; 2005.
(16) Pfahl D, Rus I. Special issue on ProSim 2004, The 5th International Workshop on Software Process Simulation and Modeling, Edinburgh, Scotland, May 2004. Software Process: Improvement and Practice 2005;10(3):251-253.
(17) Powell A, Mander K, Brown D. Strategies for lifecycle concurrency and iteration – A system dynamics approach. Journal of Systems and Software, 1999 4/15;46(2-3):151-161.
(18) Ruiz Carreira M, Ramos I, Toro M. Modelado y simulación del proceso de desarrollo de software: Una técnica para la mejora de procesos. Técnicas cuantitativas para la gestión en la ingeniería del software Oleiros, La Coruña: Netbiblo SL; 2007. p. 69-90.
(19) Ruiz M, Ramos I, Toro M. A simplified model of software project dynamics. Journal of Systems and Software 2001 12/15;59(3):299-309.
(20) Wernick P, Scacchi W. Special Issue on ProSim 2003, The 4th International Workshop on Software Process Simulation and Modeling, Portland, OR May 2003. Software Process: Improvement and Practice 2004;9(2):51-53.
(22) Zeigler BP, Kim TG, Praehofer H. Theory and practice of modeling and simulation. New York: Academic Press; 2000.

QoS-Aware Services composition using Tabu Search and Hybrid Genetic Algorithms

Jose Antonio Parejo
Pablo Fernandez
Antonio Ruiz Cortés

Dept. Computer Languages and Systems.
University of Sevilla

Abstract. In a distributed services oriented environment, having a myriad of functionally equivalent services, Quality of Service(QoS) emerges as the key differential factor. In this scenario organizations can dynamically select partners for their core business processes expressed as Composite Web Services (CWS). As a consequence, QoS-aware composition should drive an effective selection by optimizing different factors and meeting constraints according to preferences of organizations. QoS-aware composition can be formulated as a NP-hard optimization problem. In order to deal with this hard problem, different heuristic techniques (such as genetic algorithms with different solution encodings or simulated annealing) had been proposed in the literature. In this paper we apply metaheuristic optimization techniques to this problem, specifically tabu search and an hybrid genetic algorithm. We compare these techniques with other proposals using experimental results, showing that our proposals provide improvements.

1 Introduction

In a distributed environment of services, organizations can dynamically select partners for their core business processes. Those business processes are implemented as Composite Web Services (CWS), typically expressed using BPEL. In this context, QoS properties (such as cost, performance and reliability) are key factors for selecting and composing services. QoS-aware selection of concrete services to support the composition can be formulated as an optimization problem [2] [4].
In this paper we afford the QoS-aware service selection problem using a framework solution that boosts the adoption of different metaheuristic techniques using a common runtime. Main contributions of the paper are shown below:

1. We pioneer the application of some heuristic techniques applied to this novel problem, such as tabu search (TS) and a hybrid variant of genetic algorithm (HGA).
2. We present the results of a empirical study of the performance of these techniques compared with other heuristics, such as a basic genetic algorithm (GA), and iterative steepest descent method (ISD). The results shown:

- Our implementation of the hybrid genetic algorithm performs better than the rest of techniques on large instances of the problem -providing better solutions for nearly all runs-.
- Our implementation of tabu search performs better than GA and ISD, but show a less stable behavior than HGA and only performs better on small-medium instances of the problem -between 10 and 30 tasks and less than 300 candidate services- with short computation times.

The rest of the paper is organized as follows: Section 2 gives a formal definition of the Composite Web Service Selection problem. In section 3 we describe optimization framework, and comment their advantages for the purpose of this paper, while in section 4 we introduce innovative techniques to deal with the problem. In section 5 we present the empirical study and discuss their results. Section 6 describes context and related work. Finally, in section 7 conclusions are drawn.

2 Composite Web Service Selection Problem definition

There are different languages that could be used to specify the composite web services, from UML activity diagrams, to BPMN or BPEL. In the remainder of the paper, we assume that the language used to express the composition contains a set of basic construction blocks (This set is summarized in table 1).

Table 1. Basic building blocks of the composite web service description language

basic Building Block	Description
Sequence	Denotes a sequential execution of a set of tasks
Switch	Denotes an alternative choice of the execution between sets of tasks
Fork	Denotes a parallel asynchronous execution of different sets of tasks
Loop	Denotes an iterative execution of a sequence of tasks

The composition will be expressed in terms of abstract services (tasks in the remainder of the paper). There exists a set of concrete candidate services for each task, that are functionally equivalent but have different behavior in terms of quality properties. In this sense, the solution space of the composite web service selection problem, is defined by all the possible combinations of candidate services for each task. We assume that each composite web service includes a single entry and exit task.

Given the previous composite web service definition and constraints the concept of execution path can be defined as in [3]: 'A sequence of tasks (abstract services) $\{t_1, t_2, ..., t_n\}$ such that t_1 is the initial task of the composition, t_n is the final task, and no t_i belongs to alternative branches'. Execution paths will be denoted by ep_k.
As remarked in [3], for each execution path ep_k a probability of execution $freq_k$

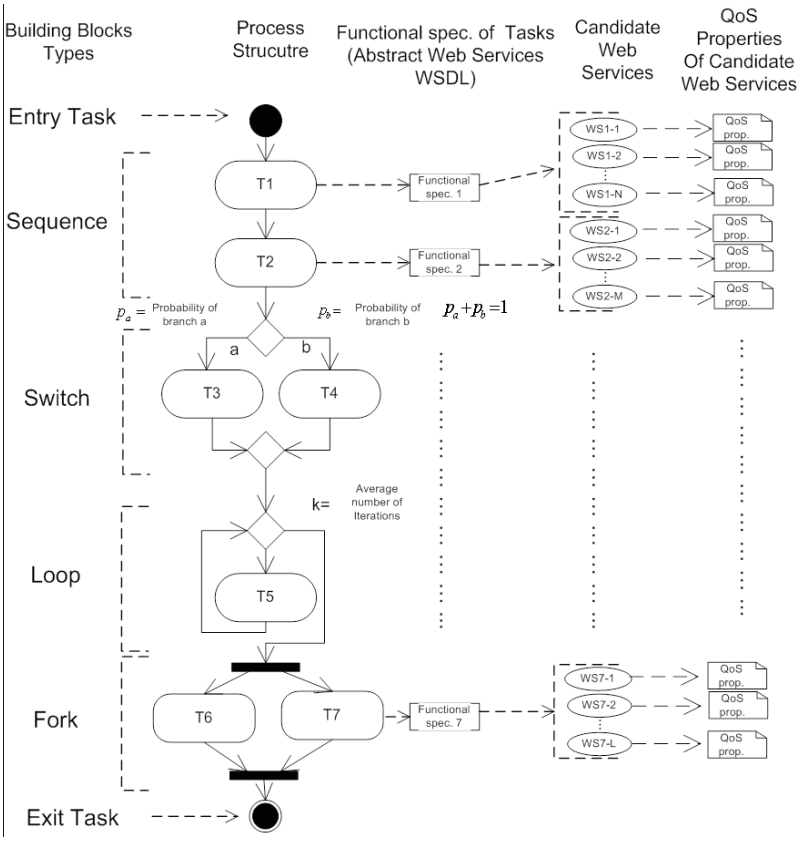

Fig. 1. Building blocks, process specifications and concrete services

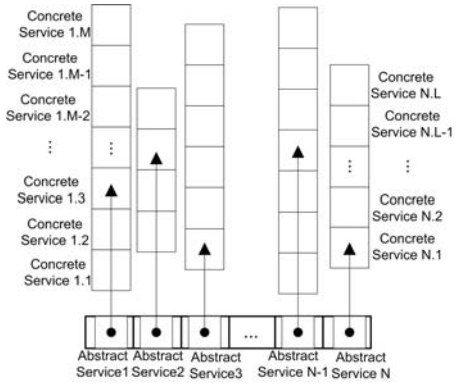

Fig. 2. Solution encoding

can be computed as the product of the probability of execution of the branch conditions included in ep_k. For a composite web service described in any language using the specified building blocks, and according to the stated constraints, all possible execution paths can be built using a simple algorithm.

2.1 QoS Model for Composite Web Services

In this paper, the quality model is based on a set of quality dimensions, which had been used in [3] and [5]. The approach to compute the global QoS of a selection of concrete services for an abstract composite web service is similar to the proposal stated in [5] and [7]. This formulation had been used extensively in the literature with some variations, mainly in the treatment of loops [3] [22]. In our work loops are annotated with an average number of executions k, and the computation of the overall quality values of the loop execution is based on this value. In a similar way, each choice of a switch is annotated with a probability of execution. In table 2 we summarize the aggregation functions applied for each QoS property defined for the composition of web services. The

Table 2. Aggregation functions per QoS Attribute

QoS Property	Aggregation Functions
Cost (c)	$\sum_{i=1}^{n} c_i$
Time (t)	$\sum_{i=1}^{n} t_i$
Reliability (rel)	$\prod_{i=1}^{n} rel_i$
Reputation (rep)	$\frac{\sum_{i=1}^{n} rep_i}{n}$
Security (sec)	$\min_{i=1}^{n} sec_i$
Custom attribute (atr)	$F(attr_i)_{i \in 1...n}$

fitness of a solution is computed by the evaluation of the set of execution paths that conform the description of the composite web service multiplied by their probability of execution $freq_k$. We obtain the fitness of each execution path using each quality properties of the selected concrete services for each task, and applying a Simple Additive Weighting. The specific value of $freq_k$ for each execution path is obtained by monitoring the execution of the composite web service. In our model of composite web service selection it is possible to impose global and local constraints on quality dimensions, and concrete web services dependency constraints as in [3]. For the inclusion of the constraints in the fitness function we have used a relative weight over the feasibility distance as in [5]. We define the feasibility distance $D_f()$ as the number of not meet constraints divided by the total number of constraints. Our final Fitness function for a solution s is:

$$fitness(s) = (1 - w_{feasibility}) * (\sum fitness(ep_k, s) * freq_k) + w_{feasibility} * D_f(s) \quad (1)$$

where:

$$fitness(ep_k, s) = w_{cost} * Cost(ep_k, s) + w_{time} * Time(ep_k, s) + w_{rel} * Reliability(ep_k, s) + w_{rep} * Reputation(ep_k, s) + w_{sec} * Security(ep_k, s) \quad (2)$$

From the previous definition, we can state that the value of $w_{feasibility}$ controls the penality we apply to individuals that violate constraints.

2.2 General solution encoding

In order to apply optimization techniques to our problem we must provide a suitable encoding of solutions. In our solution encoding we have used a structure similar to the vector-based encoding described in [5]. Solutions are encoded as a vector of integer values, with a size equal to the number of tasks. Each value in this vector represents the concrete service selected for each task. In figure 2 we show the structure graphically.

3 Optimization Framework

Most metaheuristic approaches for discrete optimization problems are usually implemented from scratch. In this paper we have used FOM [15] (Framework for Optimization using Metaheuristics), an object-oriented framework that can be used as a general tool for solving optimization problems using metaheuristics. The basic idea behind the framework is to separate the problem from the metaheuristic algorithms used to solve it. This separation allows to fully reuse different metaheuristic optimization components, and partially reuse problem definition in terms of the framework. In this sense, different metaheuristic techniques can benefit from a common solution encoding. Moreover, typically local search techniques based on neighborhood exploration (such as tabu search or simulated annealing), can use a common neighborhood generation structure. From this point of view, the usage of the framework for the purpose of this paper is fully justified, because we need to implement different heuristics and compare their results. In so doing, the usage of the framework reduces significantly the implementation effort and providing other benefits such as declarative configuration and parametrization of termination criteria, monitoring of different parameters during optimization process (including heuristic-specific parameters such as population diversity or number or rejected tabu moves per iteration).

4 Description of the Proposed Techniques

4.1 Tabu Search

Tabu search is based on a intelligent exploration of the solutions neighborhood [11], and had been extensively used because of their simplicity and practical

effectiveness. In order to apply this technique to our problem, we must define the neighborhood of our solutions. Given a solution S defined using our general solution encoding, a different solution S_n will be a neighbor of S if there exists a simple move that applied to S_n has as result S. We define a simple movement as change in the selected concrete service for a given task defined in the composition. One of the most basic intelligent exploration strategies in tabu search is the usage of recent memory, that is usually implemented using a tabu list in order to avoid reverse moves -that typically generate cycles in the search path and lost of effectiveness-. In our implementation we use a fixed size list with a most recently used policy of insertion. In addition to the core tabu technique, our search algorithm incorporates an aspiration condition. This extension allows that tabu conditions do not prohibit the exploration of really promising solutions. In this sense, if a tabu move improves the best solution found, it will be applied.

4.2 Hybrid Genetic Algorithm

Genetic algorithms maintain a population of individuals that represent more or less efficient solutions to the problem [12]. This population evolves along generations until a termination criterion is reached. During each generation, operators are applied to the individuals generating changes in the individuals and the whole population. Our initial population is generated randomly, and we use the standard two-points crossover operator [9] and a mutation operator that selects the concrete service associated to a task and selects randomly a different service from the set of candidates.

Local improver: Various strategies of hybridization have been suggested in the literature when using genetic algorithms [18]. The usage of hybrid techniques allows to escape from local optima convergence and improve the convergence speed to the global optimum. In this paper we use a local improvement procedure. This procedure is based on an iterative neighborhood search so that a given solution is replaced with best neighbor found. In our initial implementation we explored the whole neighborhood of each individual of the population. However, the computational cost of this exploration for each individual prevents from the execution of enough evolution iterations to provide a sufficient diversification of search in the solution space. In this work we propose the random exploration of a percentage of the neighborhood.

5 Experimental Results

We have tested our composite web service problem model and algorithms in wide set of instances. Those instances are randomly generated based on a problem model by a problem generator. Additionally, we have designed experiments that are executed a repeatedly for each generated problems. The execution of the algorithms had been performed on a 2.7 GHz Dual Core Intel CPU, with 2

Table 3. Statistic Data models used to generate our problem instances

Data	Stat. Dist. (Small-Medium)	Stat. Dist. (Medium-Large)
Tasks	Uniform(min:4, min:20)	Uniform(min:20, max:60)
Execution paths	Uniform(min:4 max: 20)	Uniform(min:20 max: 60)
Tasks per execution path	Uniform(min:4, max:20)	Uniform(min:10, max:60)
Candidate services per task	Uniform(min: 2, max:15)	Uniform(min: 15, max:30)
Value of Cost	Uniform(min:0.1, max:1)	Uniform(min:0.1, max:1)
Value of Time	Uniform(min:0.1, max:1)	Uniform(min:0.1, max:1)
Value of Reliability	Uniform(min:0.9 max:1)	Uniform(min:0.9 max:1)
Value of Reputation	Uniform(min:0.1, max:1)	Uniform(min:0.1, max:1)
Value of Security	Uniform(min:0.5, max:1)	Uniform(min:0.5, max:1)

GB of RAM. In table 3 we summarize the data used to generate our problem instances.

In our experimental results we have applied 4 different optimization techniques, whose parameters are summarized below:

- A basic genetic algorithm (GA).
  - Population size: 50 individuals.
  - Crossover probability: 0.7
  - Mutation probability: 0.1
  - Random selection for mutation.
  - Tournament of three individuals for selection for Crossover.
- The hybrid genetic algorithm described previously in this work (HGA).
  - Population size: 50 individuals.
  - Crossover probability: 0.7 (it is also the value of death rate, because the size of our population is constant)
  - Mutation probability: 0.2
  - Random selection for mutation.
  - Tournament selection for Crossover of three individuals.
  - Percentage of explored neighbors: 5 for small size problems and 2 for medium.
- The tabu search algorithm described previously in this work (TS).
  - Memory structure of 40 moves for small problem instances and 100 moves for medium, with a most recently used policy.
  - An aspiration criteria to select better than current optimal solutions.
- Iterative Steepest Descent [19]using random initial solutions (ISD).

General parameters of the experiment:

- Weight of QoS properties: w_{cost}=0.4, w_{time}=0.2,w_{rel}=0.1, w_{rep}=0.1, w_{sec}=0.2.
- Feasibility Weight: $w_{feasibility}$=0.5.
- Global Constraints:
  - Cost¡=0.8*(Max length of computed execution Paths)
  - Time¡=0.8*(Max length of computed execution Paths)

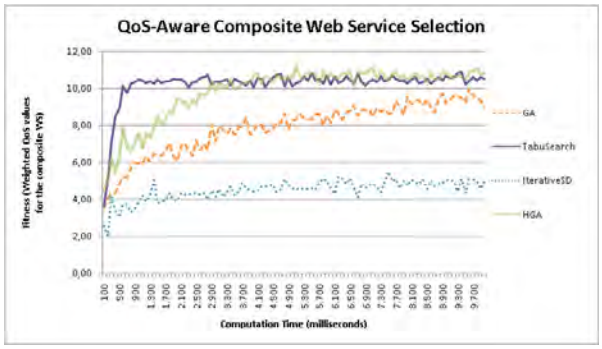

Fig. 3. Performance of metaheuristics for small-medium size composite web services

- Number of repeated executions for each problem instance and execution time: 5
- Number of problems generated for each problem size: 5

The results shown in figure 3 and 4, represent the average fitness obtained by the different heuristics to the problems generated with small and medium size problem models. Both figures show the performance of heuristics obtained with different computation times ranging from 100 milliseconds to 10 seconds. Figure 3 shows that for execution times minor to 2 seconds and small problem sizes, TS performs better than the rest of the heuristics, and that HGA performs better than GA. For longer execution times, HGA performs better than the rest but the difference with TS is relatively small. Figure 4 shows that for medium size problem instances the results are different. The size of the neighborhood of solutions in this kind of problem makes TS perform bad for short execution times -the search is not diversified enough, because TS performs a smaller number of iterations. HGA shows the best and stable performance of all the heuristics evaluated in this case. All solutions found as final results for each technique were feasible.

However, our proposals have a main drawback: the parameters of this heuristics must be well tuned to obtain good results. When using tabu search during our experimentation phase we must fine tune the size of memory, because for many instances of the problems search cycles obtaining poor results, a strategy of long term memory could help to overcome this problem. Moreover, our hybridization strategy is based on exploration of the neighborhood, the percentage of the neighborhood explored for each individual is a key factor to determine. The algorithm provides results similar to a basic GA if this percentage is small. If this percentage is high, the exploration can obtain good improvements but our algorithm can evolve less generations and results are worst than using a basic GA. Moreover, experiments with bigger problem instances show that basic GA provides better results because of the growth of the neighborhood size.

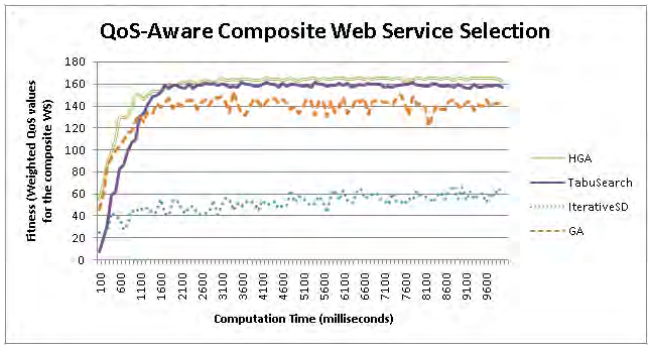

Fig. 4. Performance of metaheuristics for medium-large size composite web services

6 Related Work

Dynamic selection and late binding of web services are amongst the more promising capabilities that SOA brings. In fact it has been identified as a main research area in SOA systems [14], and therefore is being addressed as a main research field by both academy and industry. In this area, QoS-aware composite web services selection appears as a challenging problem. Two kind of optimization strategies had been formulated for this problem in literature [22] [2]:

- Local service QoS-Aware selection. In those approaches, the best candidate service for each task is selected according to the quality properties and stated preferences. This kind of methods has two main drawbacks:
  - Obtained solutions are suboptimal respect to the overall quality of the composite web service.
  - Global constraints according the structure of the composite web service and their quality properties can not be imposed
- Global composite service QoS-aware selection. The whole set of concrete services that are used to implement the composite services are optimized according to their QoS properties. Therefore, QoS global constraints from the whole composition perspective can be formulated.

Global composite service QoS-aware optimal selection has been identified as a NP-hard problem [2], [4]. In order to deal with this complex problem different approaches have been proposed:

- **Usage of Integer** [22] [1]**,Linear** [6] **or Mixed (I/L) Programming techniques** [3] [17]. Although this approaches provide the global optimum of the problem, and their performances is better for small size instances of the problem, genetic algorithms outperforms these techniques for large size instances [5]. Moreover metaheuristics are more flexible, because those techniques can consider non-linear composition rules, constraints and fitness function formulations [2].

– **Usage of heuristic techniques**. Given the intractable nature of the problem, the usage of heuristics is a logical choice. In [13] some specific heuristics are developed to solve the service composition problem. Applications of metaheuristics to this problem, are present in the literature, mainly using different Genetic Algorithm based approaches, incorporating variants to the work presented in [5], either on the encoding scheme or the fitness function or QoS model [10] [20] [21], or using population diversity handling techniques [23] [24]. In [8] a multi objective evolutionary approach is used to identify a set of optimal solutions according to different quality properties without generating a global ranking. In [16] fuzzy logic is used to relax the QoS constraints, in order to find alternative solutions when it is not possible to find any solution to the problem. Some authors have proposed the usage of simulated annealing [21]. Our work starts from [5], using a similar QoS model adding some additional quality properties and a modified, simple additive weighting fitness function. Moreover we adopted the executions paths based description of process structure used in [3] among others.

7 Conclusions

In this paper we address the optimal QoS-aware selection in composite web services. We had proposed metaheuristic based algorithms: hybrid genetic algorithm and tabu search, and compared their performance against other two techniques: iterative steepest descent, and a basic Genetic Algorithm. Experimental results show that hybrid genetic algorithm performs better than the basic version for small and medium problem sizes, and that the tabu search based algorithm is not better except for small problem instances and short run times (that can be caused by the absence of long term memory). Future work will perform a more intensive experimentation and tunning of parameters for each technique and analysis of behavior under stronger constraints, in order to confirm the conclusions obtained here, and the usage of other metaheuristics such as ant systems and simulated annealing or different optimization techniques such as Linear/Integer Programming solvers.

8 Acknowledgments

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472), and Andalusian Government project ISABEL (TIC-2533).

References

1. R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in meteor-s. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 23–30, Washington, DC, USA, 2004. IEEE Computer Society.

2. D. Ardagna and B. Pernici. Global and local qos guarantee in web service selection. In *Business Process Management Workshops*, pages 32–46, 2005.
3. D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *Software Engineering, IEEE Transactions on*, 33(6):369–384, 2007.
4. P. A. Bonatti and P. Festa. On optimal service selection. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 530–538, New York, NY, USA, 2005. ACM.
5. G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1069–1075, New York, NY, USA, 2005. ACM.
6. V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti. Efficient provisioning of service level agreements for service oriented applications. In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*, pages 29–35, New York, NY, USA, 2007. ACM.
7. J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, April 2004.
8. D. Claro, P. Albers, and J. Hao. Selecting web services for optimal composition. In *Proc. Int?l Conf. Web Services (ICWS ?05)*, 2005.
9. J. Dreo, A. Petrowski, and E. Taillard. *Metaheuristics for Hard Optimization*. Springer, 2003.
10. C. Gao, M. Cai, and H. Chen. Qos-driven global optimization of services selection supporting services flow re-planning. In *Advances in Web and Network Technologies, and Information Management*, Lecture Notes in Computer Science, pages 516–521. Springer, 2007.
11. F. Glover. Tabu search: part i. *ORSA Journal on Computing*, 1:190–206, 1989.
12. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine learning*. Addison Wesley, 1989.
13. M. C. Jaeger, G. Mühl, and S. Golze. Qos-aware composition of web services: An evaluation of selection algorithms. In *Lecture Notes in Computer Science*, Lecture Notes in Computer Science, pages 646–661. Springer, 2005.
14. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, November 2007.
15. J. A. Parejo, J. Racero, F. Guerrero, T. Kwok, and K. Smith. Fom: A framework for metaheuristic optimization. *Lecture Notes in Computer Science*, 2660:886–895, 2003.
16. M. D. Penta and L. Troiano. Using fuzzy logic to relax constraints in ga-based service composition. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, June 2005.
17. Y. Qu, C. Lin, Y. Wang, and Z. Shan. Qos-aware composite service selection in grids. *Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference*, pages 458–465, Oct. 2006.
18. J. Renders and S. Flasse. Hybrid methods using genetic algorithms for global optimization. *IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics*, 26(2), 1996.
19. J. A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer, 2005.

20. S. Su, C. Zhang, and J. Chen. An improved genetic algorithm for web services selection. In *Distributed Applications and Interoperable Systems*, volume 4531/2007 of *Lecture Notes in Computer Science*, pages 284–295. Springer, 2007.
21. H. Wang, P. Tong, P. Thompson, and Y. Li. Qos-based web services selection. *icebe*, 0:631–637, 2007.
22. L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.
23. C. Zhang, S. Su, and J. Chen. Efficient population diversity handling genetic algorithm for qos-aware web services selection. In *Computational Science ? ICCS 2006*, volume 3994/2006 of *Lecture Notes in Computer Science*, pages 104–111. Springer, 2006.
24. C. Zhang, S. Su, and J. Chen. Diga: Population diversity handling genetic algorithm for qos-aware web services selection. *Comput. Commun.*, 30(5):1082–1090, March 2007.

Finding Defective Modules from Highly Unbalanced Datasets

J C Riquelme[1], R Ruiz[2], D Rodríguez[3], and J Moreno[3]

[1] Department of Computer Science
University of Seville
Avd. Reina Mercedes s/n
41012 Sevilla
riquelme@us.es
[2] Universidad Pablo de Olavide
Ctra Utrera Km 1
41013 Sevilla, Spain
robertoruiz@upo.es
[3] Department of Computer Science
University of Alcalá
28805 Alcalá de Henares, Madrid, Spain
daniel.rodriguezg@uah.es

Abstract. Many software engineering datasets are highly unbalanced, i.e., the number of instances of a one class outnumber the number of instances of the other class. In this work, we analyse two balancing techniques with two common classification algorithms using five open public datasets from the PROMISE repository in order to find defective modules. The results show that although balancing techniques may not improve the percentage of correctly classified instances, they do improve the AUC measure, i.e., they classify better those instances that belong to the minority class from the minority class.
Keywords: Balancing techniques, Classification, Software Engineering datasets, defect prediction.

1 Introduction

The number of software engineering repositories containing project management data, source code data, etc. is increasing mainly due to automated data collection tools that allow managers and developers to collect large amounts of information. However, dealing with such large amount of information convey associated problems. For example, when dealing with defect prediction, most datasets are highly unbalanced, i.e., the number of instances of the majority class (non defective modules) outnumber the number of instances of the other class (defective module). In such cases, data mining algorithms do not generate optimal classification models to predict future defective modules. In the data mining literature, different balancing techniques have been proposed to overcome this problem.

In this work, we compare two balancing techniques and two classification algorithms to the problem of finding defective modules. To do so, we have analysed two common classification algorithms and two balancing techniques with five open public datasets from the PROMISE repository[4].

The rest of the paper is organised as follows. Section 2 cover the background about balancing and classification techniques. Section 3 contains the related work. Section 4 discusses the experimental work carried out and the results. Finally, Section 5 concludes the paper and highlights future research work.

2 Background

2.1 Unbalanced Datasets

With unbalanced datasets, the generated models can be suboptimal as most data mining algorithms assume balanced datasets. In such cases, there are two alternatives, either (i) to apply algorithms that are robust to unbalanced datasets or (ii) balance the data using sampling techniques before applying the data mining algorithm. Sampling or balancing techniques can be classified into two groups:

- *Over-sampling*. This group of algorithms aim to balance the class distribution increasing the minority class.
- *Under-sampling*. This group of algorithms tries to balance the class removing instances from the majority classes.

The simplest techniques in each group are Random Over-Sampling (ROS) and Random UnderSampling (RUS). In ROS, instances of the minority class are randomly duplicated. On the contrary, in RUS, instances of the majority class are randomly removed from the dataset.

In order to improve on the performance of random sampling, there are other techniques that apply more sophisticated approaches when adding or removing instances from a dataset. Within the undersampling group, Kubat and Matwin [1] proposed a technique called One-Sided Selection (OSS). Instead of removing instances from the mayority class randomly, OSS attempts to undersample the majority class by removing instances that are considered either noisy or borderline. The selection of noise or borderline instances is carried out applying the Tomlek links [2]. Another undersampling technique is the Neighbourhood Cleaning Rule (NCL) [3] uses Wilson's Edited Nearest Neighbour Rule (ENN) [4] to remove instances from the majority class when two out of three of the nearest neighbors of an instance contradict the class.

As a non-random oversampling method, Chawla *et al.* [5] proposed a method called Synthetic Minority Oversampling Technique (SMOTE). Instead of randomly duplicating instances, SMOTE generates new synthetic instances by extrapolating existing minority instances with random instances obtained from k

[4] `http://promisedata.org/`

nearest neighbors. The technique first finds the k nearest neighbors of the minority class for each minority example (authors used $k = 5$). Then the new samples are generated in the direction of some or all of the nearest neighbors, depending on the amount of oversampling desired.

2.2 Classification Methods

There are many classification methods but to analyze the effectiveness of balancing techniques, here we use two different and well-known types of classifiers, a decision tree classifier (C4.5) and a probabilistic classifier (Naïve Bayes). These two algorithms have also been selected because they represent different approaches to learning.

- The *naïve Bayes* [6] algorithm uses the Bayes theorem to predict the class for each case, assuming that the predictive attributes are independent given a category. A Bayesian classifier assigns a set of attributes $A_1, \ldots, A_n$ to a class C such that $P(C|A_1, \ldots, A_n)$ is maximum, i.e., the probability of the class description value given the attribute instances, is maximal.
- *C4.5* [7]. A decision tree is constructed in a top-down approach. The leaves of the tree correspond to classes, nodes correspond to features, and branches to their associated values. To classify a new instance, one simply examines the features tested at the nodes of the tree and follows the branches corresponding to their observed values in the instance. Upon reaching a leaf, the process terminates, and the class at the leaf is assigned to the instance. C4.5 uses the gain ratio criterion to select the attribute to be at every node of the tree.

3 Related Work

There is a large set of literature related to defect prediction using statistical regression techniques (e.g. Basili *et al.* [8] to data mining. For example, Khoshgoftaar *et al.* [9] which used neural networks for quality prediction. Authors used a dataset from a telecommunications system and compare the neural networks results with with a non-parametric model. Also, Khoshgoftaar *et al.* [**?**] have applied regression trees as classification model to the same problem. Fenton *et al.* [10] propose Bayesian networks as a probabilistic technique to estimate defects among other parameters.

In relation to the problem of unbalanced datasets, sampling techniques are the most widely used. For example, Seiffert *et al.* [11] studied the reduction of noise in the data using a large number of sampling techniques such as *Random undersampling* (RUS), *random oversampling* (ROS), *one-sided selection* (OSS), *cluster-based oversampling* (CBOS), Wilson's editing (WE), SMOTE (SM) and *borderline*-SMOTE (BSM). Authors concluded that (RUS, WE and BSM) are more robust than (ROS and SM) and that the unbalanced level affects the generation of optimum models. For this work, authors used only a public dataset

and generated a number of artificial datasets but extended in another work [**?**] with further methods and datasets from the UCI repository.

In the software engineering domain, Yasutaka *et al.* [12] analyzed four sampling techniques (ROS, SMOTE, RUS, ONESS) to balance a dataset before applying three classification algorithms (lineal discriminant analysis – LDA, Logistic Regression – LR, Neural Networks – NN and Classification Trees – CT) in defect prediction. In this study, sampling improved the classification accuracy of LDA and LR but not for NN and CT. Authors, however, used only one dataset that was not publicly available.

Applying the approach of using robust algorithms to handle unbalanced datasets, Li and Reformat [13] describe a algorithm based on fuzzy logic for this kind of problem.

As the PROMISE repository is publicly available, there are several works applying the same datasets to the ones used here. For example, Menzies *et al.* [14] have used have applied J48 (an implementation of C4.5) and Naïve Bayes to several datasets of the repository to predict defects concluding that are good estimators and suggesting as part of future work a resource bound exploration.

4 Experimental Work

4.1 Datasets

In this paper, we use the CM1, KC1, KC2, and PC1 datasets available in the PROMISE repository [15] to generate models for defect classification. All these datasets were created from projects carried out at NASA and collected under their metrics[5]. Table 1 shows the number of instances (modules) for each dataset with the number of defective, non-defective and their percentage showing that all are highly unbalanced, varying from 7% to 20%. The last attribute is the programming language used to develop those modules.

Table 1. Dataset used in this work.

Dataset	*# of instances*	*Non-defective modules*	*Defective*	*% defective*	*Language*
CM1	498	449	49	9.83	C
JM1	10,885	8,779	2,106	19.35	C
KC1	2,109	1,783	326	15.45	C++
KC2	522	415	107	20.49	C++
PC1	1,109	1,032	77	6.94	C

All datasets contain the same 22 attributes composed of 5 different lines of code measure, 3 McCabe metrics [16], 4 base Halstead measures [17], 8 derived Halstead measures [17], a branch-count, and the last attribute is a binary class

[5] `http://mdp.ivv.nasa.gov/`

with two possible values (false or true, whether the module has or not reported any defects).

McCabe metrics were introduced in 1976 and are based on the count of the number of paths contained in program based on its graph. To find the complexity, the program, module, method of class in an object oriented programming is represented as a graph, and its complexity can be calculated as: $v(g) = e - n + 2$, where e is the number of edges of the graph and n is the number of nodes in the graph. The cyclomatic complexity measures quantity, but McCabe also defined *essential complexity*, `ev(g)`, to measure the quality of the code (avoiding what is known as *spaghetti* code). Structured programming only requires sequences, selection and iteration structures, and the *essential complexity* is calculated in the same manner than the cyclomatic complexity but from a simplified graph where such structures have been removed. The *design complexity* metric, `iv(g)`, is similar, but taking into account calls to other modules.

Another set of metrics is known as Halstead's Software Science also were developed at the end of the 70s. These metrics are based on simple counts of tokens (using compilers's jargon), grouping those into: (i) *operators* such as keywords from programming languages such as `IF THEN`, `READ`, `FOR`; arithmetic operators `+`, `-`, `*`, etc; relational operators ($>$, $\leq$, etc.) and logical operators (`AND`, `EQUAL`, etc.); and (ii) *operands* that include variables, literals and constants. Halstead distinguishes between the number of unique operators and operands and total number of them. Table 2 summarises the metrics collected for all of the datasets. Table 2 summarizes the metrics collected from the datasets.

4.2 Execution and Results of the Experiments

The experiments were conducted using WEKA [18]. As sampling methods in this work, we have use *resample* implementation of WEKA which replicates instances randomly of and our implementation of SMOTE[6]. The classification methods were already implemented in WEKA. The results reported in this section were obtained 10 cross-validation, i.e., data is divided into 10 bins using the 90% of the instances for training and then, 10% for testing. The procedure is repeated 10 times so all data is used for both training and testing and the reported results are the average of those 10 runs.

In the case of balanced datasets, it is reasonable to use *accuracy* to measure performance. However, with unbalanced datasets, models may not consistently produce a useful balance between false positive rates and false negative rates. The ROC (Receiver Operating Characteristic) graphs are used to analyse the relationship between *true positive rate* and *true negative rate*. However, the *Area Under the ROC Curve* (AUC) is generally used to provide a single value of the performance. *AUC* is a useful metric for classifier performance as it is independent of the decision criterion selected and prior probabilities.

These measures have been used to compare the accuracy of the classifiers with and without balancing techniques. The results of the experiment can be

[6] The implementation of SMOTE is available at `http://www.ieru.org/wiki`

Table 2. Attribute Definition Summary

	Metric	*Definition*
McCabe	loc	McCabe's Lines of code
	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	iv(g)	Design complexity
Halstead base	uniq_Op	Unique operators, n_1
	uniq_Opnd	Unique operands, n_2
	total_Op	Total operators, N_1
	total_Opnd	Total operands N_2
Halstead Derived	n	Vocabulary, $n = n_1 + n_2$
	L	Program length, $N = N_1 + N_2$
		Estimated Length: $N' = n_1 \cdot log_2 n_1 + n_2 \cdot log_2 n_2$
	V	Volume, $V = N \cdot log_2(n)$
	d	Difficulty $D = 1/L$
	i	Intelligence
	e	Effort $e = V/L$
		estimated $e = n_1 N_2 N log_2 n / 2 n_2$ (elementary mental discriminations)
	b	Error Estimate
	t	Time estimator $T = E/18$ seconds
	lOCode	Line count of statement
	lOComment	Count of lines of comments
	lOBlank	Count of blank lines
	lOCodeAndComment	Count of lines of code and comments
Branch	branchCount	No. branches of the flow graph
Class	false, true	Whether the module has reported defects

observed in tables 3 and 4. The percentage of correctly classified instances did not increase, but the AUC improves when using sampling techniques, especially with SMOTE.

Table 3. Percentage of Correctly Classified Instances

	CM1	*JM1*	*KC1*	*KC2*	*PC1*
J48	87.95	79.50	84.54	81.42	93.33
Naïve Bayes	85.34	80.42	82.36	83.52	89.18
Resample & J48	91.57	86.84	89.81	90.80	94.86
Resample & Naïve Bayes	83.94	79.40	81.32	85.82	85.03
SMOTE & J48	84.28	77.27	82.75	80.13	88.34
SMOTE & Naïve Bayes	78.43	70.93	76.18	75.68	83.62

Table 4. AUC Values

	CM1	*JM1*	*KC1*	*KC2*	*PC1*
J48	0.56	0.65	0.69	0.70	0.67
Naïve Bayes	0.66	0.68	0.79	0.83	0.65
Resample & J48	0.80	0.80	0.82	0.84	0.80
Resample & Naïve Bayes	0.72	0.69	0.78	0.87	0.67
SMOTE & J48	0.77	0.75	0.79	0.77	0.77
SMOTE & Naïve Bayes	0.72	0.68	0.79	0.83	0.61

5 Conclusions and Future Work

In this paper, we analysed two balancing techniques and two classification algorithms in order to find defective modules in five publicly available software engineering datasets. The results show that although balancing technique do not improve the percentage of correctly classified instances, they do however improve the AUC measure, i.e., they classify better those instances from the minority class which are the ones of interest (in this case, to find defective modules). Also, SMOTE seems to improve the AUC measure over the resampling method.

Future work will consist of implementing further balancing algorithms and how to combine balancing techniques and feature selection of attributes when datasets are highly unbalanced.

Acknowledgements

This research was supported by Spanish Research Agency (TIN2007–68084–C02–00) and project CCG07-UAH-TIC-1588 (jointly supported by the Univer-

sity of Alcalá and the autonomic community of Madrid). Also to the Universities of Seville, Pablo de Olavide and Alcalá.

References

1. Kubat, M., Matwin, S.: Addressing the curse of imbalanced training sets: one-sided selection. In: Proc. 14th International Conference on Machine Learning, Morgan Kaufmann (1997) 179–186
2. Tomek, I.: Two modifications of cnn. IEEE Transactions on Systems, Man and Cybernetics **6**(11) (November 1976) 769–772
3. Laurikkala, J.: Improving identification of difficult small classes by balancing class distribution. Technical Report Technical Report A-2001-2, University of Tampere (2001)
4. Wilson, D.L.: Asymptotic properties of nearest neighbor rules using edited data. Systems, Man and Cybernetics, IEEE Transactions on **2**(3) (July 1972) 408–421
5. Chawla, N.V., Kevin W. Bowyer, Lawrence O. Hall, W.P.K.: Smote: Synthetic minority over-sampling technique
6. Mitchell, T.: Machine Learning. McGraw Hill (1997)
7. Quinlan, J.R.: C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, California (1993)
8. Basili, V., Briand, L., Melo, W.: A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering **22**(10) (1996) 751–761
9. Khoshgoftaar, T., Allen, E., Hudepohl, J., Aud, S.: Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks **8**(4) (1997) 902–909
10. Fenton, N., M.Neil, Krause, P.: Software measurement: uncertainty and causal modeling. IEEE Software **19** (2002)
11. Seiffert, C., Van Hulse, T.M.K.J., Folleco, A.: An empirical study of the classification performance of learners on imbalanced and noisy software quality data. IEEE International Conference on Information Reuse and Integration (IRI 2007). (13–15 Aug. 2007) 651–658
12. Kamei, Y., Monden, A., Matsumoto, S., ichi Matsumoto, T.K.K.: The effects of over and under sampling on fault-prone module detection. In: Empirical Software Engineering and Measurement (ESEM 2007). (September 2007) 196–204
13. Li, Z., Reformat, M.: A practical method for the software fault-prediction. IEEE International Conference Information Reuse and Integration (IRI 2007) (13-15 Aug. 200) 659–666
14. Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering **33**(1) (2007) 2–13
15. Boetticher, G., Menzies, T., Ostrand, T.: Promise repository of empirical software engineering data (`http://promisedata.org/`). Technical report, West Virginia University (2008)
16. McCabe, T.J.: A complexity measure. IEEE Transactions on Software Engineering **2**(4) (December 1976) 308–320
17. Halstead, M.: Elements of Software Science. Elsevier (1977)
18. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. 2 edn. Morgan Kaufmann, San Francisco (2005)

Integrating the Development of Data Mining and Data Warehouses via Model-driven Engineering

Jose Zubcoff[1], Jesús Pardillo[2], Jose-Norberto Mazón[2], and Juan Trujillo[2]

[1] Department of Sea Sciences and Applied Biology,
University of Alicante, Spain
Jose.Zubcoff@ua.es
[2] Department of Software and Computing Systems,
University of Alicante, Spain
{jesuspv,jnmazon,jtrujillo}@dlsi.ua.es

Abstract. Data mining is one of the most important analysis techniques to automatically extract knowledge from large amount of data. Nowadays, data mining is based on low-level specifications of the employed techniques typically bounded to a specific analysis platform. Therefore, data mining lacks a modelling architecture that allows analysts to consider it as a truly software-engineering process. Bearing in mind this situation, we propose a model-driven approach which is based on (i) a conceptual modelling framework for data mining, and (ii) a set of model transformations to automatically generate both the data under analysis (that is deployed via data-warehousing technology) and the analysis models for data mining (tailored to a specific platform). Thus, analysts can concentrate on understanding the analysis problem via conceptual data-mining models instead of wasting efforts on low-level programming tasks related to the underlying-platform technical details. These time consuming tasks are now entrusted to the model-transformations scaffolding. The feasibility of our approach is shown by means of a hypothetical data-mining scenario where a time series analysis is required.

Keywords: data mining, data warehouse, model-driven engineering, model transformation, multidimensional modelling, conceptual modelling.

1 Introduction

Data-mining techniques allow analysts to discover knowledge (e.g. patterns and trends) in very large and heterogeneous data sets. Data mining is a highly complex task which requires a great effort in preprocessing data under analysis, e.g. data exploration, cleansing, and integration [1]. Therefore, some authors suggest the suitability of data-warehousing technologies [2] for improving the conventional knowledge discovery process by means of providing an integrated and cleansed collection of data over which data-mining techniques can be straight applied [3,4]. However, current data-mining literature has been focused on the presenting new techniques and improving the underlying algorithms [5], whilst the most known software platforms do not apply the data warehousing principles in data-mining design. To overcome this situation, several mechanisms have been proposed [6,7,8,9] to model data mining techniques in conjunction with data-warehousing technology from the early stages of design (i.e. conceptual). These data-mining models do not only support analysts in using and understanding the required data-mining techniques in real-life scenarios, but also allow designers to document the data-mining techniques in detail. Hence, these data-mining models are truly blueprints that can be used to manually obtain the required data-mining metadata as a basis of the implementation in a certain data-mining platform. However, this highly-complex task is only accessible to expert analysts and requires too much effort to be successfully completed [3,10].

In this work, we will go beyond the definition of new models, here we define a model-driven engineering [11] approach for data-mining. Moreover, we propose the use of a well-known visual modelling standard, the "unified modelling language" (UML) [12] for facilitating the design and implementation tasks. In order to spread the usage of data-mining models to a broader scope of analysts and reduce the required effort our approach automatically generate a vendor-specific data-mining implementation from a conceptual data-mining model, taking into consideration the deployment of underneath data warehouse (i.e. data under analysis).

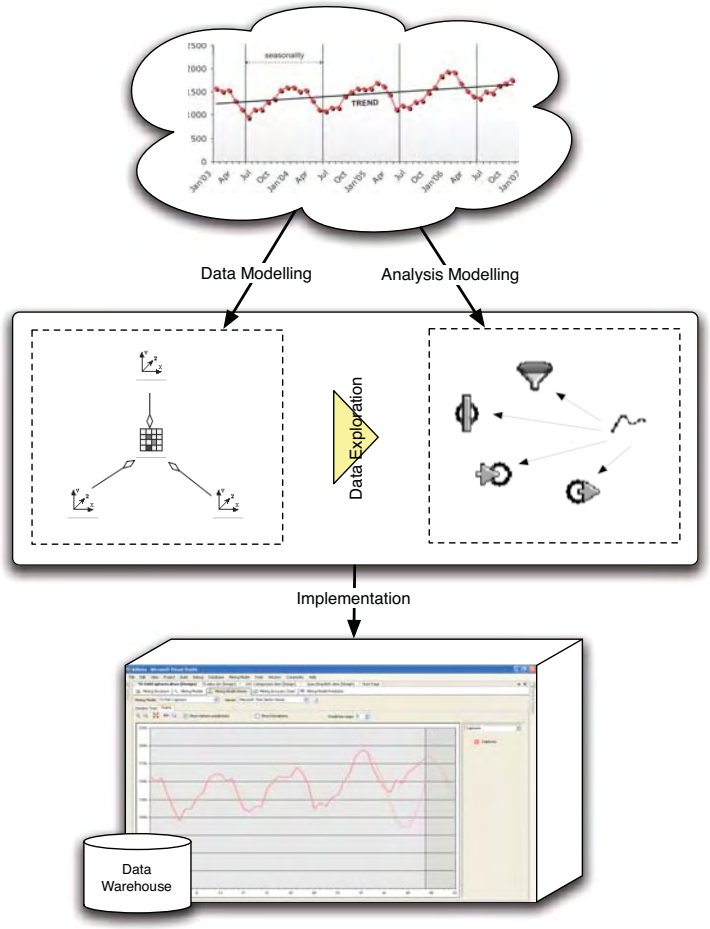

Fig. 1. Modelling example of a time-series analysis in data warehouses

Running example. In order to illustrate the discussion, in Fig. 1, we show an example of data-mining conceptual modelling. In this example, a small organisation is trying to highlight patterns and trends in the evolution of the fish-species population along time. Therefore, a time-series analysis has to be modelled [9]. Together with this data-mining technique, the data under analysis can be specified by exploring the underlying multidimensional data of the data warehouse. This crucial phase is done at a high level of abstraction, that is at conceptual level, focusing only on data mining concepts and avoiding platform specific

details. This process is represented in the middle of Fig. 1. Finally, to obtain the results, our approach provides the required transformations for mapping the data-mining conceptual models to a vendor-specific data-mining implementation[3].

Outline. The rest of the paper is structured as follows: the next Section outlines the related work. Section 3.2 describes our model-engineering approach for data mining. A running example is used through the paper to clarify every theoretical detail Finally, Section 4 exposes conclusions, also sketching the future work.

2 Related Work

Current approaches for data-mining design can be classified on those that are a general description of data mining process, or those that are mathematical oriented and propose solutions at a very low-abstraction level. Therefore, these approaches overlook the definition of understandable artifacts which could be easily used by designers in a software engineering process. The main standard proposed for the data mining process is the "cross industry standard process for data mining" (CRISP-DM) [13]. This standard is a detailed description of each of the six phases of the data mining process. This standard neither proposes a concrete modeling tool nor present a conceptual model for data mining. CRISP-DM is focused on the description of how to perform a data mining task.

An overview of current data-mining modelling languages is provided in Table 1. The "common warehouse metamodel" (CWM) [12] and the "predictive model markup language" (PMML)[4][5] are really standards for the metadata interchange proposed by vendor-independent consortiums (OMG and DMG, respectively) between data-mining applications based on XML, but they cannot be used as analysis artefacts. The "data mining extensions" (DMX)[6] is a SQL-like language for (textually) coding data-mining models in the Microsoft Analysis Services platform, and therefore it is difficult to gain understanding of the data-mining domain. In addition, some data-mining libraries have been also proposed as a modelling mechanism. Two of the most known are the "extended library for Prudsys embedded solutions" (XELOPES)[7] (derived from CWM) and Weka[8]. They provide an entire framework to carry out data mining but, once again, they are situated at very low-abstraction level, since they are code-oriented and they do not contribute to facilitate understanding of the domain problem. On the other hand, there are software architectures related to data mining such as the "pattern-base management system" (PBMS)[9] designed to store and manage patterns obtained from the usage of data-mining techniques, but they cannot be considered a truly modelling proposal as we state herein.

All of these approaches have the same drawback, since they are focused on solving the technical scaffolding instead of providing analysts with intuitive artefacts to specify data mining. To the best of our knowledge, only the proposal described in Zubcoff *et al.* [6,7,8,9] provides a modelling framework (named as CDM in Table 1) to define data-mining techniques at a high-abstraction level by using the "unified modelling language" (UML) [12]. However, these UML-based models are mainly used as documentation. In this paper, we propose to extend this modelling framework as a first step in turning data mining into a

[3] The code is included in the appendix A
[4] URL: `www.dmg.org/pmml-v3-2.html` (March 2008)
[5] In Table 1, we exclude PMML due to the space constraints. PMML is similar to CWM but it is a language (Type field) coded in XML schema (Technology).
[6] URL: `msdn2.microsoft.com/en-us/library/ms132058(VS.90).aspx` (March 2008)
[7] URL: `www.prudsys.com/Produkte/Algorithmen/Xelopes` (March 2008)
[8] URL: `www.cs.waikato.ac.nz/ml/weka` (March 2008)
[9] URL: `www.pbms.org` (March 2008)

Table 1. Comparison of data-mining modelling languages

Language	CDM [6,7,8,9]	CWM	XELOPES	DMX	Weka
Type	metamodel	metamodel	library	query language	library
Technology	UML profiles	MOF Instance	CWM Extension	SQL-like	Java
Subject	interaction	interoperability	interoperability computation	querying	computation
Abstraction	high	middle	middle	low	low
Complexity	low	medium	high	medium	high
User type	analysts	data managers	data managers	data miners	data miners
Expertise	low	medium	high	medium	high

real software engineering process (see Fig. 1). Specifically, we use model-driven engineering concepts to (i) specify data-mining analysis in two technology-independent models (the multidimensional-data model of the underlying data warehouse and the data mining technique model), and (ii) provide transformations to automatically deploy data and analysis specifications into their physical implementations.

3 Model-driven Architecture for Data Mining in Data Warehouses

Our model-driven engineering approach for data mining advocate defining the underneath data warehouse (i.e. data under analysis) together with the data-mining technique. In this section, both tasks are explained, as well as the required transformations to obtain the data-mining implementation. Furthermore, our running example is used through this section to clarify the theoretical details.

3.1 Deployment of Data under Analysis

This section explains how to develop the underlying data warehouse required for data-mining to provide integrated and cleansed data. The data warehouse is based on a multidimensional model which defines the required data structures, namely facts and dimensions and their respective measures, hierarchies and attributes. Multidimensional modelling resembles the traditional database design [4]. First, a conceptual design phase is performed whose output is an implementation-independent and expressive conceptual multidimensional model for the data warehouse. A logical design phase then aims to obtain a technology-dependent model from the previously defined conceptual multidimensional model. This logical model is the basis for the implementation of the data warehouse. In previous work, we have aligned this process with a model-driven approach [14,15,16,17] in order to support designers to develop a conceptual multidimensional model and the automatic derivation of its corresponding implementation.

A conceptual multidimensional model for the running example is provided in the Fig. 2. This conceptual model has been defined by using our UML profile for multidimensional modelling [18]. A *capture* fact have been designed together with a set of four dimensions: *fish*, *ship*, *time*, and *location*. For each dimension, several levels are defined that form a hierarchy of aggregation. Locations have a <*site, marine area, region*> hierarchy, fish have a <*species, genus, family*> hierarchy, and the time dimension has the typical <*day, week, month, quarter, year*> hierarchy. Finally, ships have no hierarchies to be described, thus presenting only one aggregation level with data of the ship.

From this conceptual model, an implementation of the required data structures can be automatically obtained tailored to several specific platforms, e.g. relational [19] or multidimensional [20].

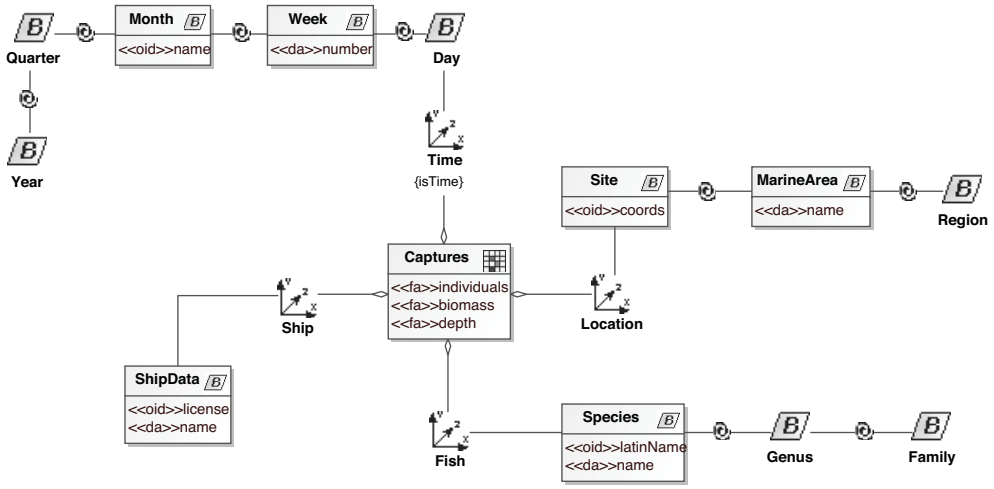

Fig. 2. The conceptual multidimensional model of the repository

3.2 Transformations for Model-driven Data Mining

Whilst the derivation of the data under analysis is traditionally performed through a three-step process, analysis techniques such as data mining present different requirements for their development. In this section, a model-driven engineering approach for the deployment of data-mining models together with the data under analysis is described.

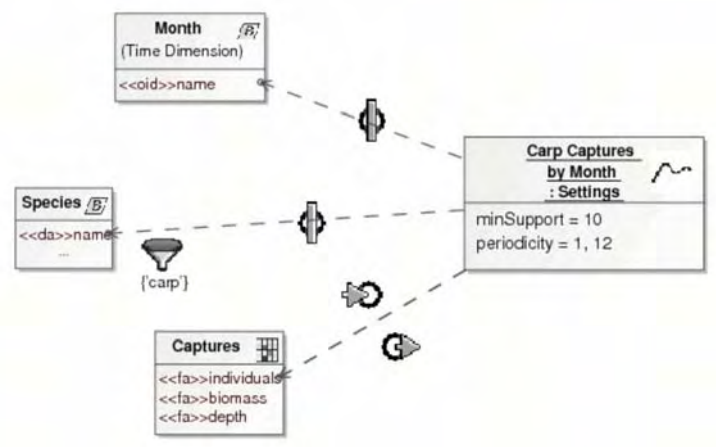

Fig. 3. The conceptual model of the time series analysis

The novelty of our approach is twofold: (i) it is based on defining vendor-neutral models of data-mining techniques together with the model of the underlying data warehouse, and (ii) the deployment of those data-mining techniques is done automatically. Therefore, on one hand, we use a modelling approach [6,7,8,9] for defining platform-independent models for several data-mining techniques. This approach is a high-level vendor-neutral modelling

language to visually and easily specify analysis by means of applying data-mining techniques. Following the running example of Fig. 1, from the underlying multidimensional-data model of fish captures, an analyst can explore this model and specify its data-mining needs by the conceptual modelling of a time-series analysis in this case. Specifically, this conceptual model has been defined by using the UML extension presented in [9]. Therefore, a *carp captures by month* analysis for the *individuals of each captures* time series can be easily specified and carried out. The conceptual model defined for the required analysis is shown in Fig. 3. The analysis model also includes a time axis is specified in increments of *months* and a filter for the *carp specie*. Nevertheless, additional parameters of the data-mining technique can be also specified such as a *minimum data support* or a suggestion of the series *periodicity*.

On the other hand, this language is not directly implemented in any data-mining platform, and thus, it only acts as a blueprint of the executable analysis. Therefore, the model-transformation configuration has to be described in order to consider every kind of target platform from this platform-independent modelling language. In Fig. 4, we provide an overview of the required model-transformation architecture, stressing some of the current data-mining standards and platforms in the market.

The conceptual data-mining modelling framework in data warehouses (Zubcoff *et al.* [6,7,8,9]) is shown at the top of this model-driven architecture. Fig. 4 also shows the transformation paths to derive several implementations through mapping data-mining models to other languages that really have established an executable environment: CWM, XELOPES, DMX, or Weka acting as bridge. Depending to the characteristics of the analysis itself (*e.g.*, the required technique) or the data-mining solution available (*e.g.*, it can only be open-source platforms), we choose one of the transformation paths. Furthermore, Depending on the target-language representation, model-to-model or model-to-text transformations could be needed. Therefore, some of the data-mining solutions that are able to interpret the previous modelling languages are also represented in Fig. 4. On the lower side, some of the data-mining platforms are also represented. Whereas there are standards such as CWM that are vendor-neutral and many CWM-compliant tools can be considered (Oracle Miner, CWM4ALL, etc.), others such as DMX are commonly restricted to the platform for which they are originally were thought (the Microsoft's in this case).

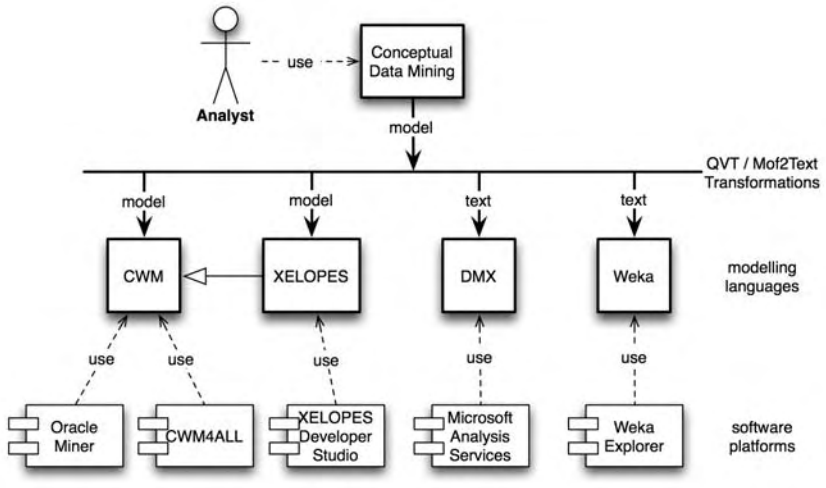

Fig. 4. Model-transformation architecture for data mining

From a technical point of view, we propose the usage of the "model-driven architecture" (MDA) [12] in order to implement these transformations between data-mining models. Within an MDA-based approach the "query/view/transformation" (QVT) language can be used as a standard mechanism for defining formal relations between MOF-compliant models that allows the automatic derivation of a implementation. Nevertheless, there are transformations that are applied from models (*i.e.*, MOF-based) to implement code (*i.e.*, textual modelling languages). In these cases, MDA offers the "MOF models to text transformation" (Mof2Text) language that allows us to specify transformations by means of textual templates in order to automatically derive the corresponding implementation.

3.3 Example Data-mining Transformation for a Specific Platform

Herein, we follow our running example for carrying out a time-series analysis of fish-species population (see Fig. 1), in order to automatically derive the implementation for a specific software platform. Specifically, in this paper, we employ the Microsoft Analysis Services[10] as the target platform. As we previously shown, this platform provides the DMX language to code data-mining models. Therefore, given the time-series analysis model of Fig. 1, we have designed the required mapping from this model into DMX code. Due to the space constraints, we exclude an abstract specification of the involved mapping, also omitting an example transformation of the data under analysis that can be found in [16]. Nevertheless, in Fig. 5, it is shown the implementation of this mapping (left-hand side) over the Eclipse development platform. In order to accomplish this task, we have used the MOFScript[11] plug-in for this platform. MOFScript is a transformation-language implementation of the Mof2Text standard language that enable us to specify model-to-text transformations in the "model-driven architecture" (MDA) [12] proposal. On the right-hand side, the resulting DMX code for the time-series analysis of Fig. 1 is shown.

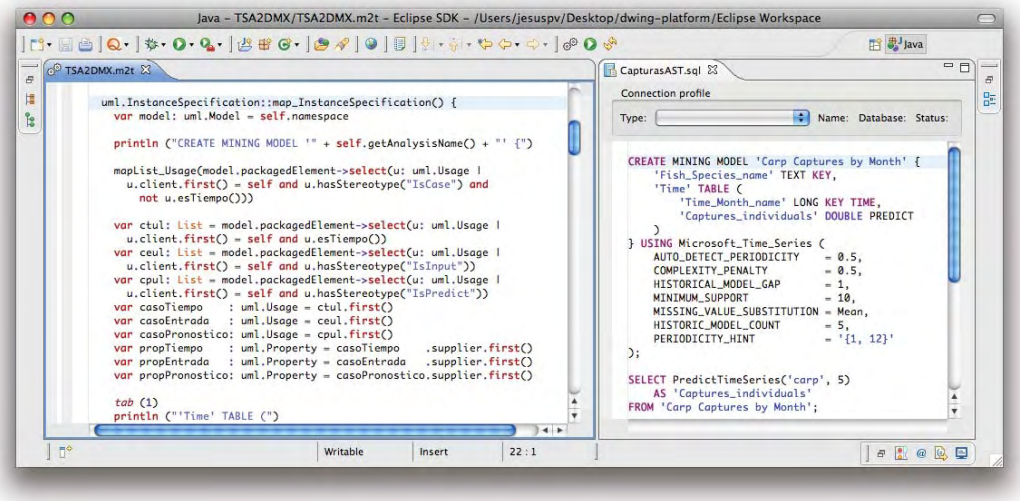

Fig. 5. Example of a Mof2Text transformation and the generated code

[10] URL: `www.microsoft.com/sql/solutions/bi` (March 2008)
[11] URL: `www.eclipse.org/gmt/mofscript` (March 2008)

Given Fig. 5, the mapping overview is as follows: each time-series analysis (represented by some kind of modelling element in the source metamodel)[12] is mapped into a data-mining model in DMX (`MINING MODEL` instruction). Every parameter of the analysis technique is also mapped into their DMX counterpart. In addition, the unspecified parameters in the conceptual model are later explicitly defined in the implementing code. On the other hand, each data under analysis (taken from the multidimensional model of the underlying data warehouse) is mapped into a data-mining attribute in DMX (by defining a table and then creating its corresponding column). Once the mapping is correctly established, the MOF-Script engine can interpret this one in order to translate a certain conceptual model of time-series analysis to the DMX code, and thus, implementing it in the Microsoft Analysis Services platform. Finally, within this solution, analysts can consult the data-mining results (see the bottom-side of Fig. 1) by visualising the obtained patterns and trends and extracting new knowledge from them.

4 Conclusion

Due to mathematical foundations of data-mining techniques, there are no formalised mechanisms to easily specify data-mining activities as a real software engineering process. In this paper, we propose a model-engineering approach for overcoming this limitation. On one hand, we provide a set of models to specify data-mining techniques in an vendor-neutral way that are close to the way of analysts thinking about data-mining (i.e. conceptual models). On the other hand, we provide transformations to automatically derive platform-specific models from the conceptual ones, altogether with the deployment of data under analysis [16,17]. Thus, analysts can only focus on data mining itself at an abstract level instead of distracting by details related to a certain vendor data-mining solution whilst the model transformations can automatically derive vendor-specific implementations in background for current data-mining platforms. Furthermore, our approach for data-mining modelling is also concerned about modelling the data under analysis, i.e. the data warehouse in order to provide analysts with a way of quickly understand for being close to their way of thinking about data. The data-mining techniques are smoothly integrated in this model.

Therefore, the great benefit of our approach is that, once we have established the model-driven architecture for both data under analysis and analysis techniques for data mining, analysts can model their data-mining related tasks easily in a vendor-neutral way whereas the model-transformations scaffolding is entrusted to automatically implement them in a certain platform.

Future Work. Our immediate future work covers other high-level mechanism to specify data-mining related tasks. For instance, we will study how current goal-oriented approaches for requirement analysis [21] can help us to guide the selection of data-mining solutions. In addition, we are investigating on the integration of the proposed data-mining framework together with the analysis technologies traditionally employed in the data-warehouse domain.

5 Acknowledgements

This work has been supported by the ESPIA (TIN2007-67078) project from the Spanish Ministry of Education and Science, and by the QUASIMODO (PAC08-0157-0668) project from the Castilla-La Mancha Ministry of Education and Science (Spain). Jesús Pardillo and Jose-Norberto Mazón are funded by the Spanish Ministry of Education and Science under FPU grants AP2006-00332 and AP2005-1360, respectively.

[12] Please, see [9] for an additional explanation of the modelling primitives involved in a time-series analysis.

References

1. Pyle, D.: Data Preparation for Data Mining. Morgan Kaufmann (1999)
2. Kimball, R., Ross, M.: The Data Warehouse Toolkit. Wiley (2002)
3. Inmon, W.H.: The Data Warehouse and Data Mining. Commun. ACM **49**(4) (1996) 83–88
4. Rizzi, S., Abelló, A., Lechtenbörger, J., Trujillo, J.: Research in data warehouse modeling and design: dead or alive? In: DOLAP. (2006) 3–10
5. Hand, D.J., Mannila, H., Smyth, P.: Principles of Data Mining. MIT Press
6. Zubcoff, J.J., Trujillo, J.: A UML 2.0 profile to design Association Rule mining models in the multidimensional conceptual modeling of data warehouses. Data Knowl. Eng. **63**(1) (2007) 44–62
7. Zubcoff, J.J., Pardillo, J., Trujillo, J.: Integrating Clustering Data Mining into the Multidimensional Modeling of Data Warehouses with UML Profiles. In: DaWaK. (2007) 199–208
8. Zubcoff, J.J., Trujillo, J.: Conceptual Modeling for Classification Mining in Data Warehouses. In: DaWaK. (2006) 566–575
9. Pardillo, J., Zubcoff, J., Trujillo, J.: Un perfil UML para el análisis de series temporales con modelos conceptuales sobre almacenes de datos. In: IDEAS Workshop. (2007) 369–374
10. González-Aranda, P., M.E.M.S.S.J.: Towards a Methodology for Data mining Project Development: The Importance of abstraction. In: In: ICDM Workshops (FDM). (2004) 39–46
11. Bézivin, J.: Model Driven Engineering: An Emerging Technical Space. In: GTTSE. (2006) 36–64
12. Object Management Group: Common Warehouse Metamodel (CWM), Unified Modeling Language (UML), Model Driven Architecture (MDA), Query/View/Transformation Language (QVT), MOF Model to Text Transformation Language (Mof2Text). `http://www.omg.org` (March 2008)
13. CRISP-DM Consortium: CRISP-DM, version 1.0. `http://www.crisp-dm.org/` (may 2008)
14. Pardillo, J., Trujillo, J.: Integrated Model-driven Development of Goal-oriented Data Warehouses and Data Marts. (2008) In Press
15. Pardillo, J., Mazón, J.N., Trujillo, J.: Model-driven OLAP Metadata from the Conceptual Models of Data Warehouses. (2008) In Press
16. Mazón, J.N., Trujillo, J.: An MDA approach for the development of data warehouses. Dec. Support Syst. **In Press** (2007)
17. Mazón, J.N., Trujillo, J., Lechtenbörger, J.: Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms. Data Knowl. Eng. **63**(3) (2007) 725–751
18. Luján-Mora, S., Trujillo, J., Song, I.Y.: A UML profile for multidimensional modeling in data warehouses. Data Knowl. Eng. **59**(3) (2006) 725–769
19. Mazón, J.N., Trujillo, J., Serrano, M., Piattini, M.: Applying MDA to the development of data warehouses. In: DOLAP. (2005) 57–66
20. Mazón, J.N., Pardillo, J., Trujillo, J.: Applying Transformations to Model Driven Data Warehouses. In: DaWaK. (2006) 13–22
21. Mazón, J.N., Pardillo, J., Trujillo, J.: A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. In: ER Workshops. (2007) 255–264

6 Appendix A. Code of the model-transformation example

```
texttransformation AST2DMX (in uml:"http://www.eclipse.org/uml2/2.0.0/UML") {
  uml.Model::main() {
    file (self.name + ".sql")
    println ("-- Generated by MOFScript (" + date() + " " + time() + ")\n")
    mapList_InstanceSpecification(self.packagedElement
      ->select(is: uml.InstanceSpecification | is.hasStereotype("AnalisisST")))
  }
  module::mapList_InstanceSpecification(isl: List) {
    var fis: uml.InstanceSpecification = isl.first()
```

```
fis.map_InstanceSpecification()
    isl.remove(fis)
    isl->forEach(is: uml.InstanceSpecification) {
      newline (1)
      is.map_InstanceSpecification()
    }
  }
  uml.InstanceSpecification::map_InstanceSpecification() {
    var model: uml.Model = self.namespace
    println ("CREATE MINING MODEL '" + self.getAnalysisName() + "' {")
    mapList_Usage(model.packagedElement->select(u: uml.Usage |
      u.client.first() = self and u.hasStereotype("IsCase") and
        not u.esTiempo()))
    var ctul: List = model.packagedElement->select(u: uml.Usage |
      u.client.first() = self and u.esTiempo())
    var ceul: List = model.packagedElement->select(u: uml.Usage |
      u.client.first() = self and u.hasStereotype("IsInput"))
    var cpul: List = model.packagedElement->select(u: uml.Usage |
      u.client.first() = self and u.hasStereotype("IsPredict"))
    var casoTiempo    : uml.Usage = ctul.first()
    var casoEntrada   : uml.Usage = ceul.first()
    var casoPronostico: uml.Usage = cpul.first()
    var propTiempo    : uml.Property = casoTiempo    .supplier.first()
    var propEntrada   : uml.Property = casoEntrada   .supplier.first()
    var propPronostico: uml.Property = casoPronostico.supplier.first()
    tab (1)
    println ("'Time' TABLE (")
tab (1)
    propTiempo.map_Property()
    println (",")
    tab (1)
    propEntrada.map_Property()
    newline (1)
    tab (1)
    println (")")
    var sl: List
    sl.clear()
    self.slot->forEach(s: uml.Slot) {
      if (not s.definingFeature.name.equals("numPeriodos")) {sl.add(s)}
    }
mapList_Slot(sl)
newline (1)
var cl: List    = model.packagedElement->select(c: uml.Constraint)
var c : uml.Constraint = cl.first()
var b : Boolean = true // not c.constrainedElement->select(u:uml.Usage).isEmpty()
print ("SELECT PredictTimeSeries(")
if (b) {       print (c.specification.body.first() + ", ")}
var sl:List = self.slot->select(s:uml.Slot | "numPeriodos".equals(s.definingFeature.name))
var s : uml.Slot = sl.first()
println (s.value.first().value + ")")
tab (1)
```

```
println ("AS '" + propPronostico.class.name + "_" + propPronostico.name + "'")
println ("FROM '" + self.getAnalysisName() + "';")
  }
  module::mapList_Usage(ul: List) {
    var fu: uml.Usage = ul.first()
    fu.map_Usage()
    ul.remove(fu)
    ul->forEach(u: uml.Usage) {
      println (",")
      u.map_Usage()
    }
    println (",")
  }
  uml.Usage::map_Usage() {
    var p: uml.Property = self.supplier.first()
    p.map_Property()
  }
  uml.Property::map_Property() {
    var c: uml.Class = self.class
    var s: String = c.name + "_" + self.name
    tab (1)
    if (c.hasStereotype("Fact")) {
      print ("'" + s + "' DOUBLE PREDICT")
    } else if (c.getDimension().getValue("Dimension", "isTime")) { // assumes base
      print ("'" + c.getDimension().name + "_" + s + "' LONG KEY TIME")
    } else {
      print ("'" + c.getDimension().name + "_" + s + "' TEXT KEY")
    }
  }
  module::mapList_Slot(sl: List) {
    var fs: uml.Slot = sl.first()
    println ("} USING Microsoft_Time_Series (")
    fs.map_Slot()
    sl.remove(fs)
    sl->forEach(s: uml.Slot) {println (","); s.map_Slot()}
    newline (1)
    println (");")
  }
  uml.Slot::map_Slot() {
    var name: uml.LiteralString = self.definingFeature.name
    tab (1)
    if ("autoPerÃodo".equals(name)) {
    } else if ("complejidad".equals(name)) {
    } else if ("ventana".equals(name)) {
    } else if ("minSoporte".equals(name)) {
      'MINIMUM_SUPPORT = ' self.value.first().value
    } else if ("valoresAusentes".equals(name)) {
    //} else if ("numPeriodos".equals(name)) {
    } else if ("periodo".equals(name)) {
      'PERIODICITY_HINT = '
      mapList_LiteralUnlimitedNatural(self.value)
```

```
    }
  }
  module::mapList_LiteralUnlimitedNatural(lunl: List) {
    var flun: uml.LiteralUnlimitedNatural = lunl.first()
    "\'{"
    flun.value
    lunl.remove(flun)
    lunl->forEach(lun: uml.LiteralUnlimitedNatural) {
      ', ' lun.value
    }
    "}'"
  }
  uml.InstanceSpecification::getAnalysisName(): String {
    if (self.name <> null) {
      result = self.name
    } else if (self.ownedComment.first().body <> null) {
      result = self.ownedComment.first().body
    } else {
      result = "AST_" + date() + "_" + time()
    }
  }
  uml.Usage::esTiempo(): Boolean {
    if (self.hasStereotype("Caso")) {
      var p: uml.Property = self.supplier.first()
      result = p.class.getDimension().getValue("Dimension", "isTime") // assumes a base
    } else {
      result = false
    }
  }
  uml.Class::getDimension(): uml.Class { // assumes a base
    self.ownedAttribute->forEach(p: uml.Property) {
      if (p.name <> null) {
        if (p.name.equals("D")) {
          result = p.opposite.class.getDimension() // assumes opposite
          break
        }
      } else {
        if (p.opposite <> null) {
          var c: uml.Class = p.opposite.class
          result = p.opposite.class // assumes stereotyped by Dimension
          break
        }
      }
    }
  }
}
```