Gómez Meneses, Vicente. *Consideraciones para la construcción de un árbol de clasificación de métodos de diseño.* **Tecnología en Marcha**. Vol. 12 no. 2. 1993. pp. 77-84.

CONSIDERACIONES PARA LA CONSTRUCCION DE UN ARBOL DE CLASIFICACION DE METODOS DE DISEÑO

Vicente Gómez Meneses*

Los métodos de diseño son considerados como una colección de entes individuales que tienen asociados un conjunto de características propias que permiten diferenciarlos del resto. La representación ha sido desarrollada de manera que permite determinar los métodos de diseño mediante reglas de manipulación de los conjuntos de características que los identifican a través de las clases definidas. La colección de métodos y sus características distintivas en un momento dado constituyen un estado del árbol taxonómico. Este sufre transformaciones entre estados para producir nuevos estados, producto de su crecimiento dinámico bajo la creación o aparición de nuevos métodos.

INTRODUCCION

La investigación descrita en este artículo fue realizada en el Centro de Investigaciones en Computación del Instituto Tecnológico de Costa Rica, por la inquietud de determinar una forma adecuada que permita clasificar (en general) métodos de diseño (y como caso particular métodos de diseño de sistemas de *software*) bajo un enfoque formal, estableciendo un modelo taxonómico que ofrezca las herramientas que generen la clasificación y los detalles para su manipulación. En este artículo, el enfoque se ha centrado en la descripción del árbol de clasificación y en el manejo de las características que identifican a cada método de diseño.

* Licenciado en Enseñanza de la Matemática. Máster en Computación. Profesor del Departamento de Matemática Instituto Tecnológico de Costa Rica.

ANALISIS DE LA SITUACION PREVIA

Los últimos años en la industria del *software* han estado marcados por un acelerado desarrollo que pretende resolver nuevos tipos de problemas. La literatura presenta una serie de métodos de diseño de programas que ayudan y orientan a los programadores y diseñadores de sistemas en su trabajo, y parte de la causa de este incremento se encuentra en la complejidad de los problemas que se han venido planteando.

La revolución de la tecnología de la computación y la complejidad de los ambientes en los que está siendo aplicada van de la mano. La computadora es una herramienta que se ha beneficiado enormemente del avance tecnológico y, al mismo tiempo, ha beneficiado y contribuido a los avances en la ciencia y la tecnología.

Desde los primeros años de la computación, el desarrollo del *hardware* ha sido muy rápido, mientras que el *software* ha tenido un avance más lento. No obstante, en la actualidad la producción de *software* se ha incrementado a un ritmo tal que permite decir que su avance y costo económico, se ha nivelado al ritmo de producción del *hardware*. En este sentido dice Pressman (1988)

Los últimos años en la industria del software han estado marcados por un acelerado desarrollo que pretende resolver nuevos tipos de problemas.

Durante las tres primeras décadas de la informática, el principal desafío era desarrollar el hardware de las computadoras de forma que se redujera el coste de procesamiento y almacenamiento de datos. A lo largo de la década de los 80, los avances en microelectrónica han dado como resultado una mayor potencia de cálculo a la vez que una reducción del coste. Hoy el problema es diferente. El principal desafío es reducir el coste y mejorar la calidad de las soluciones basadas en computadoras, soluciones que se implementan con el software.

Los cambios en el *hardware* han requerido que nuevos sistemas de *software* sean construidos para que el primero pueda funcionar o el segundo ser ejecutado en el nuevo *hardware*. Ha ocurrido una fuerte evolución en los conceptos de las bases de datos, los procesadores de lenguajes se han perfeccionado desde el bajo nivel de máquina a lenguajes de alto nivel orientados a procedimientos y más recientemente a objetos. Todo esto llevó, entre otros, al establecimiento de un conjunto de técnicas que se ha denominado *ingeniería de software*, y dentro de él, al área de diseño de *software*.

A partir del surgimiento de esta área muchos objetivos han sido planteados para lograr una mejor administración del *software* y su mantenimiento. En consecuencia, muchos métodos han sido sugeridos para mejorar los procesos de programación, pretendiendo satisfacer objetivos básicos como el pensar en el desarrollo de programas confiables, el reducir los costos de producción y el tener un mantenimiento más flexible del *software*.

Dado que este campo es amplio y complejo ha sido hasta el momento problemático presentar algunos de sus conceptos y principios en una forma clara y bien organizada dentro de restricciones fijas. Puede considerarse que aunque existan métricas para determinar calidad y fiabilidad del *software*, persiste una ausencia de criterios que permitan determinar cuál método utilizar bajo una circunstancia particular de problema. Se hace necesario un mejor entendimiento del diseño de *software* y sus procesos de desarrollo a fin de definir en forma apropiada las categorías de trabajos, e identificar las áreas fundamentales dentro de la ingeniería de *software*.

DECLARACION: ESTRUCTURA BASICA DEL MODELO

La estructura básica sobre la que se construye el modelo es llamada **declaración**. Esta se define como un registro que guarda información correspondiente a un método de diseño. Escribimos una declaración como una expresión funcional de la forma d(m)=r, donde m es llamado **argumento** y representa al método de diseño que se estudiará, d es llamado **descriptor** y establece una característica asociada al método m, r es llamado **referente** y establece el valor específico que caracteriza al método m bajo el descriptor d.

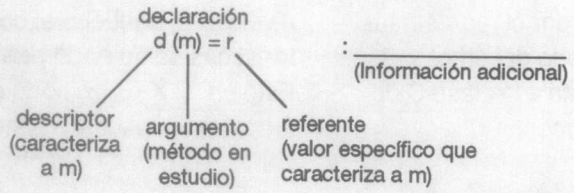
Si para una declaración ocurre que no existe valor específico del referente que caracteriza a un método m bajo el descriptor d se denota d(m)=ø.

Para efectos de claridad es adecuado (cuando se requiera) adicionar a una declaración cualquier anotación pertinente, como una breve descripción, notas de aclaración o cualquier otra información.

En la Figura 1 se esquematizan estos conceptos.

Ejemplos de declaraciones son las siguientes:

FIGURA 1. Registro de información sobre un método m.



descomposición_del_problema (DE) = flujo de datos

nivel_de_aplicabilidad (HOS) = problemas de alta precisión : objetivo primario es el control del algoritmo

Estos ejemplos muestran como argumentos a los métodos de diseño de *software* DE (Diseño estructurado) y HOS (*Higher Order Software*). Los descriptores son «descomposición_del_problema» y «nivel_de_aplicabilidad». Los valores referentes son respectivamente «flujo de datos» y «problemas de alta precisión».

En la segunda declaración, la expresión a la derecha de los dos puntos es un comentario de aclaración y ampliación sobre el referente.

CLASE Y CARACTERISTICAS

Llamamos clase de método de diseño a un conjunto de tales métodos.

Los métodos clasificados en cada clase deben satisfacer un conjunto de características que los identifican como miembros de la clase.

Se denotan las clases con letras mayúsculas, los métodos y las características con letras minúsculas.

Sea X una clase de método de diseño. Si $\{x_1,x_2,...,x_n\}$ es el conjunto formado por las n características que identifican a los métodos ubicados en la clase X, entonces denotamos a $\{x_1,x_2,...,x_n\}$ por car(X). Esto es, $car(X) = \{x_1,x_2,...,x_n\}$.

Cada una de las características que identifican a una clase de método de diseño se establecerá como una declaración. Por tanto, si para una clase X se tiene que $car(X)=\{x_1,x_2,...,x_n\}$ entonces x^i tendrá la forma d(m)=r para todo $x_i \in car(X)$ con i=1,2,...,n. Decimos entonces que $car(X) = \{d_i(m)=r_j / m \in X, i=1,2,...,n, j=1,2,...,n\}$.

Si m es un método que satisface las características x_i, para todo $x_i \in car(X)$ con i=1,2,...,n donde X es una clase de método de diseño, se dice que **m satisface car(X)** y, sin pérdida de generalidad, se denota con car(m) al conjunto de características del método m. Por tanto, m satisface car(X) <=> car(X) C car(m).

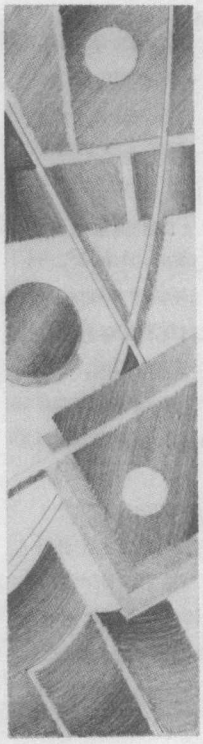
Por otra parte se establece que dos clases A y B de métodos de diseño son iguales si contienen el mismo conjunto de características para los métodos bajo su rango, esto es, A=B <=> car(A)=car(B).

ARBOL TAXONOMICO

Llamamos árbol taxonómico al que empleamos para representar la estructura jerárquica de la taxonomía de métodos de diseño. Un árbol taxonómico se define entonces como un par (N,C) donde N es el conjunto de nodos y C el conjunto de arcos entre ellos, tales que, en N los nodos internos representan las clases de métodos de diseño y las hojas representan los métodos específicos. Los arcos entre los nodos del árbol representan relaciones entre ellos, de manera que, si existe un arco dirigido de A a B, entonces, define una relación de «herencia» de A a B. Esto es, B «hereda» aquellas características de A que no presentan conflicto con sus propias características.

Cada árbol presenta un estado de la clasificación en un momento dado. Por su estructura será un árbol n-ario.

Dado que cada nodo interno en el árbol taxonómico representa una clase de método de diseño, en adelante, nos referimos al conjunto de características de un nodo como el conjunto de características que identifican a los métodos que se



ubican dentro de la clase por él representada. Es decir, si X es un nodo del árbol taxonómico que representa a la clase C de métodos de diseño, entonces, car(X)=car(C).

El nodo raíz (que en adelante se identificará como RAIZ) es tal que car(RAIZ) = { }.

Denotamos los nodos internos (que representan las clases) con letras mayúsculas (y subíndices si lo requieren). Los nodos terminales (hojas) que representan los métodos se denotan con letras minúsculas (y superíndices si lo requieren).

La Figura 2 muestra un ejemplo de un árbol taxonómico donde las hojas a, a^{21}, a^{221}, a^{231}, b^1, b^2, b^3, b^4 representan métodos de diseño, mientras que los nodos A_1, A_2, A_{22}, A_{23}, B representan una clase de métodos de diseño diferenciados por sus características.

DEFINICION FORMAL DE CLASE

Se llama clase de método de diseño al conjunto formado por los métodos que se clasifican en cada nodo interno del árbol taxonómico. Esto es, cada clase de método de diseño se corresponde con cada nodo interno en el árbol. En sentido estricto, se dice que una clase es el conjunto de nodos hojas que son alcanzables desde la clase.

En primera instancia, decimos que el conjunto de características de un nodo interno (una clase en la taxonomía) en el nivel i (i>0) contiene al conjunto de características de su nodo padre en el nivel i-1. Esto es, si X_i es un nodo del nivel i (i>0) del árbol taxonómico y X_{i-1} su padre, entonces $car(X_{i-1}) \subset car(X_i)$. No obstante, pueden presentarse situaciones de conflicto cuando algunas características de $car(X_{i-1})$ no son coincidentes con algunas características de $car(X_i)$.

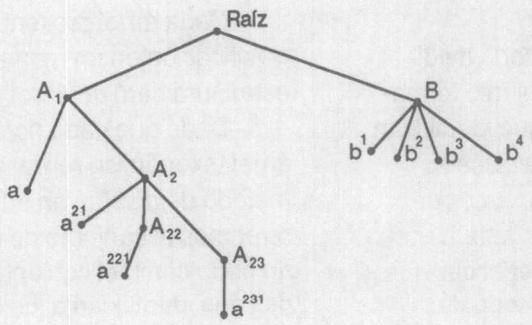


FIGURA 2. Arbol taxonómico.

HERENCIA Y CONFLICTO

El proceso de herencia se da a través de las rutas en el árbol taxonómico mediante las relaciones de los arcos entre nodos.

Se dice que dos características están en conflicto si poseen el mismo descriptor y diferente referente. Así las características $d_1(m)=r_1$ y $d_1(m)=r_2$ están en conflicto si $r_1 \neq r_2$.

CARACTERISTICAS EXPLICITAS Y DERIVADAS

El conjunto de características de un nodo en el árbol taxonómico se divide en dos subconjuntos, uno formado por las características explícitas de él y el otro por las características derivadas donde:

- las características explícitas son aquellas asociadas explícitamente con el nodo
- las características derivadas son aquellas que el nodo hereda de sus ancestros y que no presentan conflicto con sus características explícitas

El subconjunto de características derivadas de un nodo en el árbol taxonómico se obtiene como la unión de los subconjuntos de características de cada uno de sus ancestros que se dan en herencia; es decir, se transmiten de padres a hijos siempre que no se encuentren en conflicto.

La computadora es una herramienta que se ha beneficiado enormemente del avance tecnológico y, al mismo tiempo, ha beneficiado y contribuido a los avances en la ciencia y la tecnología.

El conjunto total de características de un nodo se obtiene entonces como la unión de los subconjuntos de características explícitas y derivadas. Si denotamos el subconjunto de características explícitas de un nodo X por car_exp(X) y al subconjunto de características derivadas de X por car_deriv(X) entonces car(X)=car_exp(X) U car_deriv(X)

En el proceso de herencia cada nodo hijo hereda todas las características de su nodo padre, excepto las del padre que se encuentren en conflicto con las suyas.

PROCESO DE OBTENCION DE LAS CARACTERISTICAS EN CONFLICTO Y DEL CONJUNTO TOTAL DE CARACTERISTICAS DE UNA CLASE

a. Buscar las características del nodo padre y del hijo
b. Detectar las características que tienen descriptor común
c. Para aquellas características con descriptor común y distinto referente se respetan las del hijo (aunque sea referente nulo) y se incluyen en el conjunto total de características del hijo (si los referentes son iguales no hay problema, basta incluirlos)
ch. Formar la unión de los dos conjuntos de características (del padre e hijo) excepto las del padre que estén en conflicto con las del hijo, en cuyo caso quedan en la unión las del hijo.

SITUACIONES DE CONFLICTO

Son aquellas en que se requiere tomar decisiones de clasificación de un método de diseño bajo situaciones fuera de lo normal. Es decir, aquellas en que se requiere decidir ante nuevas situaciones que no se habían contemplado en la estructura jerárquica de clases de métodos de diseño en la taxonomía.

Casos de situaciones de conflicto

Al analizar un método m se detecta que no existe ninguna clase C que posea todas las características de m, o al contrario m no posee todas las características que identifican a clase alguna C, pudiendo en estos casos presentarse una de las situaciones siguientes:

- en car(m) faltan algunas características que se encuentran en car(C)
- algunas de las características en car(m) difieren de las características en car(C)
- existen características en car(m) que no se encuentran en car(C)

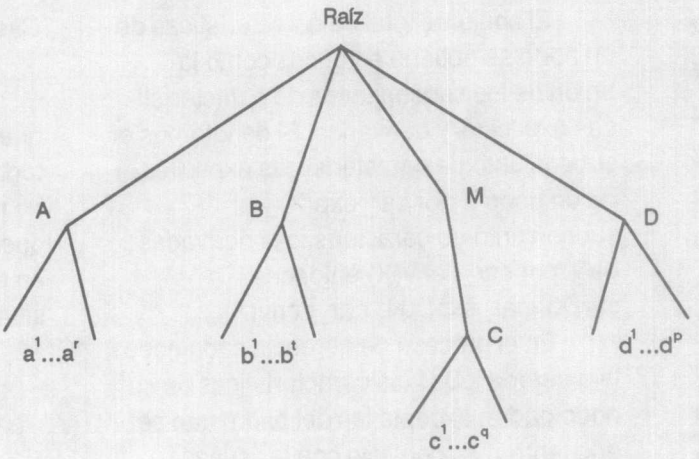
SITUACIONES DE DECISION Y UBICACION DE UN METODO EN EL ARBOL TAXONOMICO

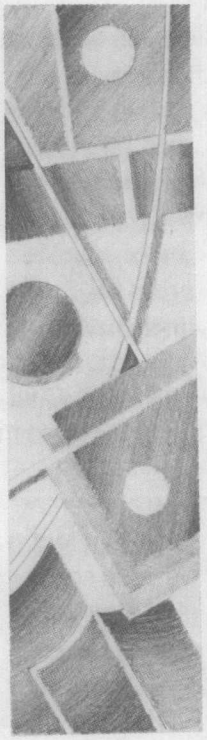
Supongamos que en el árbol taxonómico se tiene una clase de método de diseño representada en el nodo C tal que car(C)={c_1,c_2,...,c_n} (Figura 3). Sea m un método de diseño.

Al establecer car(m) se encuentra que sus características presentan (entre todas las clases) mayor similitud con las de car(C), aunque se presentan algunas de las características de m que difieren de algunas de las determinadas para definir la clase C.

Por ejemplo si car(C) = {$d_1(\alpha)=r_1$, $d_2(\alpha)=r_2$, $d_3(\alpha)=r_3$, $d_5(\alpha)=r_8$} donde α es un método de diseño y car(m) = {$d_1(m)=r_1$, $d_2(m)=r_4$, $d_4(m)=r_7$, $d_5(m)=r_8$} se tienen dos declaraciones iguales para los descriptores d_1 y d_5. Supongamos que estas son dos características tales que si no se cumplen entonces m no puede pertenecer a C. Para el descriptor d_2 el referente en C es r_2 y en m es r_4, lo que muestra que se presenta la característica pero con diferentes valores específicos para la caracterización bajo el descriptor d_2, en

FIGURA 3. Arbol taxonómico.





este caso y por razones particulares de esta segunda declaración **podría** permitirse dispensarla en car(C) de manera que no sea impedimento para clasificar a m en C. En igual forma puede presentarse el hecho de que la tercera declaración en car(C) ($d_3(\alpha)=r_3$) no sea tan determinante para permitir que m pertenezca a C aunque ésta no esté presente en car(m). Para la declaración $d_4(\alpha)=r_7$ en car(m) se determina que no causa problema pues es propia a m y no presenta contradicción con las características de car(C).

Bajo este análisis se permite decir que m es un método de la clase C.

Al detectar las diferencias entre car(m) y car(C) entra en juego, para efectos de clasificar a m, el conocimiento que sobre dicho método se tenga así como el buen juicio en el análisis de tales diferencias para juzgar si se requiere crear una nueva clase de métodos donde clasificar a m, o por el contrario, las diferencias no determinan dicha necesidad permitiendo que m pueda ser clasificado en C.

De aquí que el carácter de la decisión ante las diferencias entre estas características es dependiente del buen análisis del método. Bajo un buen criterio y estudio descansa el decidir si las características diferentes son tan fuertes que requieran construir una nueva clase (con el peligro de un innecesario crecimiento del árbol) o si son tales que pueda hacerse la salvedad y clasificar a m en C a pesar de estas diferencias.

En general se presentan métodos m tales que será necesario clasificarlos en una clase C a pesar de no cumplir con algunas características de esta clase o tener algunas diferencias entre sus características y las de C, en este caso decimos que m satisface parcialmente car(C).

Generalizando decimos que

$$m \in C \iff \left[\begin{array}{l} \text{m satisface car(C)} \\ o \\ \text{m satisface parcialmente car(C)} \end{array}\right.$$

De esta manera se permite la normal flexibilidad que se requiere en cualquier análisis taxonómico. Esto es básico en el manejo de la taxonomía con objeto de no obligar a crear una nueva clase sin que exista una absoluta necesidad.

Teniendo que el orden(car(C))=n (representa la cardinalidad del conjunto car (C)) y m un método de diseño, puede ocurrir alguna de las siguientes situaciones:

a. orden(car(m)) = orden(car(C)) = n
b. orden(car(m)) = orden(car(C))+k = n+k, con k>0

En el caso (a) se tiene que el número de características en car(m) es igual al número de características del nodo C y además n-j elementos de car(m) coinciden

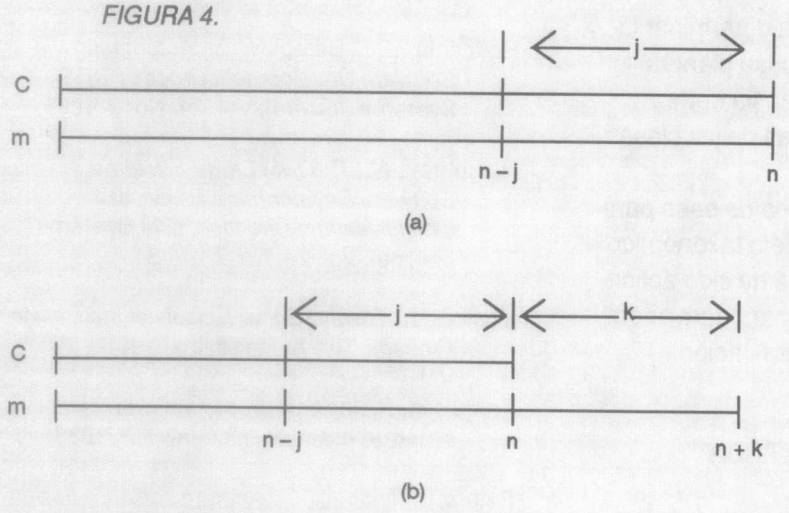
con n-j elementos de car(C) y j elementos presentan diferencias (Figura 4a). En el caso (b) se presenta un mayor número de características para el método m que las que definen al nodo C, y ocurre que n-j características de car(m) coinciden con n-j características de car(C), j características presentan diferencias y hay k características más en car(m) (Figura 4b).

En cualquiera de los dos casos las características que presentan diferencias entre car(m) y car(C), así como los elementos en el caso (b) que superan en car(m) el orden(car(C)), deben ser considerados con prudencia para determinar la necesidad o no de crear una nueva clase de método de diseño, pues puede ocurrir que sus diferencias sean irrelevantes o no afecten en el hecho de considerar a m dentro de C.

Si en cualquiera de los dos casos anteriores las j o las j+k características diferentes entre car(m) y car(C) requieren y obligan a crear una clase X que contendrá los métodos con características como las de car(m), se tendrá un **crecimiento del árbol taxonómico** con un posible estado como se muestra en la Figura 5. Si las diferencias son tales que permiten clasificar a m en la clase C se tendrá un árbol de clasificación como en la Figura 6. Puede ocurrir que no se requiera crear una nueva clase X y tampoco sea factible clasificar a m en C, en cuyo caso m se coloca como hijo de M en la Figura 7.

Del análisis anterior se observa que el árbol taxonómico puede ir creciendo en forma dinámica conforme a los requerimientos de nuevos métodos con características no consideradas hasta la instancia actual del árbol.

FIGURA 4.



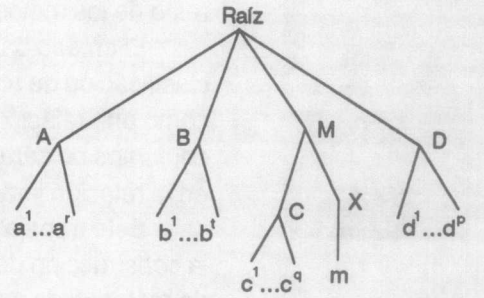


FIGURA 5. Arbol taxonómico.

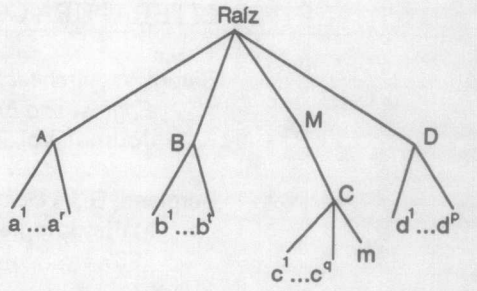


FIGURA 6. Arbol taxonómico.

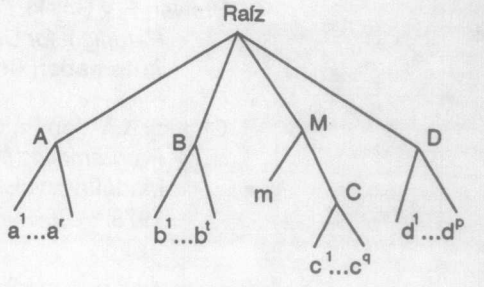
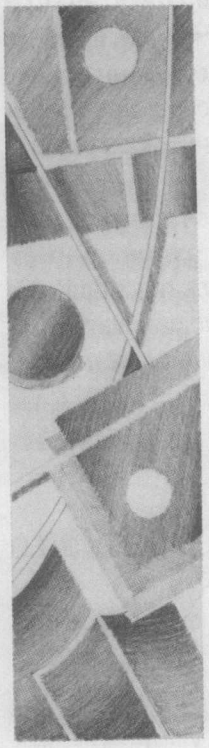


FIGURA 7. Arbol taxonómico.



CONCLUSIONES

Se ha descrito aquí la estructura de un árbol taxonómico para la clasificación de métodos de diseño. En él, una clase está formada por un conjunto de métodos con una serie de características que los identifican para ser miembros de dicha clase.

Ha sido desarrollado como una representación jerárquica de clases donde los métodos son los entes individuales que tienen asociados una colección de características propias que los diferencian del resto de los métodos.

El criterio básico para determinar la clasificación de un método, se plantea como una relación de herencia entre conjuntos de características de las clases en la relación jerárquica.

Este trabajo ha servido de base para la construcción de un modelo taxonómico de métodos de diseño, que ha sido aplicado en el área de *software* para derivar en éste una propuesta de clasificación.

LITERATURA CONSULTADA

Beregi, W., *Architecture Prototyping in the Engineering Environment,* **IBM Systems Journal**, Vol. 23, No. 1, 1984.

Bergland, G., *A Guided Tour of Program Design* **Methodologies Computer**, Octubre, 1981.

Birrel, N. y Ould M., **A Practical Handbook for Software Development**, Cambridge University Press, 1985.

Brewer, F. y Gajski, D. *An Expert-System Paradigm for Design*, **23rd. Design Automation Conference**, 1986.

Chandy K. y Yeh R., *Current Trends in Programming Methodology*, **Software Modeling**, vol. 3, Prentice-Hall, Inc., 1978.

Freeman, P., *The Nature of Design*,**Tutorial on Software Design Techniques** (IEEE Press), 1980. No. 3 y 4, 1986.

Gries, D., **The Science of Programming**, Springer-Verlag, 1981.

Hamilton,M., Zeldin,S., *Higher Order Software-A methodology for defining software*, **IEEE. Transactions on Software Engineering**, Vol. SE-2, No. 1, marzo, 1976.

Henry, S. y Kafura, D., *The Evaluation of Software Systems' Structure Using Quantitative Software Metrics*, **Software - Practice and Experience**, Vol. 14. No. 6. Junio, 1984.

Hoffnagle, G.F., Beregi, W.E., *Automating the Software Development Process*, **IBM Systems Journal**, Vol. 24. No. 2. 1985.

Humphrey, W., *The IBM Large-Systems Software Development Process: Objectives and Direction*, **IBM Systems Journal**, Vol. 24. No. 2, 1985.

Mostow, J., *Toward Better Models of the Design Process*, **The AI Magazine**, 1985.

Pressman, R., **Ingeniería del software. Un enfoque práctico**, McGraw Hill, 1988.

Peters, L., Tripp, L., *Comparing software design methodologies*, **Datamation**, Noviembre, 1977.

Spillers, W.R., *Design Theory*, **IEEE Transactions on Systems, Man, and Cybernetics**, Marzo, 1977.

Vick, C.R., Ramamoorthy, C.V., **Handbook of Software Engineering**, Van Nostrand Reinhold Company, 1984.

Zelkowitz, M.V., Shaw, A.C., Gannon, J.D., **Principles of Software Engineering and Design**, Prentice-Hall, Inc., 1979.