# UMA FERRAMENTA PARA AUXÍLIO NA REUTILIZAÇÃO DE SOFTWARE

Mário Márcio Ramos Teixeira
Flávio Roberto Dias Velasco

Instituto de Pesquisas Espaciais
Caixa Postal 515
12.201 - São José dos Campos, SP.

Resumo

Neste trabalho discutimos a aplicabilidade da reutilização de software para redução nos seus custos de produção. Apresentamos os vários caminhos para reutilização de software e propomos uma metodologia baseada no trabalho de Prieto-Diaz e Freeman. A metodologia é discutida, assim como a correspondente ferramenta e o seu emprego ilustrado com exemplos. Finalmente, são consideradas as dificuldades remanescentes para a reutilização de software, bem como possíveis soluções.

Abstract

In this work we discuss the applicability of reuse as a way of achieving reduction on software production costs. We present the various existing methods of reuse and a hierarchical multi face methodology is proposed, based on Prieto and Diaz-Freeman's method. Aspects and details of the proposed methodology and corresponding tool are also presented with examples. Finally, remaining dificulties and possible solutions are considered.

## 1. INTRODUÇÃO

A reutilização de software vem se juntar às tentativas de redução do custo do software. Com a reutilização do software, podemos não só aumentar a produtividade e, em consequência, reduzir os custos (ao abandonar o hábito de se inventar a roda a cada novo programa), como, também, aumentar a confiabilidade do software e, ainda, tornar o controle e a previsão de custos uma atividade mais precisa. Enquanto o custo do hardware decresce, sabe-se que o custo do software vem crescendo a cada ano: se continuar crescendo a 12 por cento ao ano, em 1995 serão gastos em todo o mundo, somente com software, 450 bilhões de dólares (Boehm, 1987). Este descompasso é atribuido principalmente a:

a. Especificações de requisitos cada vez mais complexas, como, por exemplo, nos sistemas militares, que devem operar bem, já na primeira vez em que são utilizados em ambientes hostis e imprevistos; algumas vezes isto não é possível, como o programa "Guerra nas Estrelas", na opinião de D.L. Parnas (1987),

b. A necessidade de profissionais especializados e escassos e, portanto, muito bem pagos,

c. As modernas ferramentas e metodologias não têm aumentado significativamente a nossa capacidade de desenvolver software. O aumento de produtividade de software, entre os anos de 1963 a 1983, tem sido de 3 a 8 por cento ao ano, enquanto que o crescimento da capacidade de processamento instalado tem sido da ordem de 40 por cento ao ano, no mesmo período (Horowitz, 1984).

Não obstante as reconhecidas vantagens, hoje reutiliza-se pouco software. Algumas destas dificuldades não são nem ao menos técnicas e sim econômicas, como, por exemplo, a de que, se for feito um produto suficientemente genérico e reutilizável, por mais impressionado (e satisfeito, é claro) que fique o cliente, não haverá uma nova encomenda, pois este não necessitará de um novo produto (Meyer, 1987). Outras dificuldades são mais específicas e relacionadas a cada tendência de reutilização de software. De acordo com Tracz (1987), a reutilização de software não vem acontecendo com maior frequência devido a:

a. não existirem normas definidas e claras, seja para o desenvolvimento de software reutilizável, seja para sistemas baseados em software reutilizável,

b. não existem nem grandes repositórios de software e componentes reutilizáveis, nem ferramentas para acessar e sintetizar sistemas, a partir de software reutilizável.

Uma tentativa inicial de classificação dos vários modos de reutilização de software pode ser: "black-box", que vem a ser a reutilização de um produto de software sem quaisquer modificações, mas em um contexto diferente daquele para o qual foi projetado; e "white-box", ou seja, a reutilização após algumas modificações, necessárias nas condições do novo ambiente. Fragmentando-se um pouco mais, e introduzindo subclasses para a divisão proposta anteriormente, temos, segundo Biggerstaff (1987) a classificação na Figura 1.1.

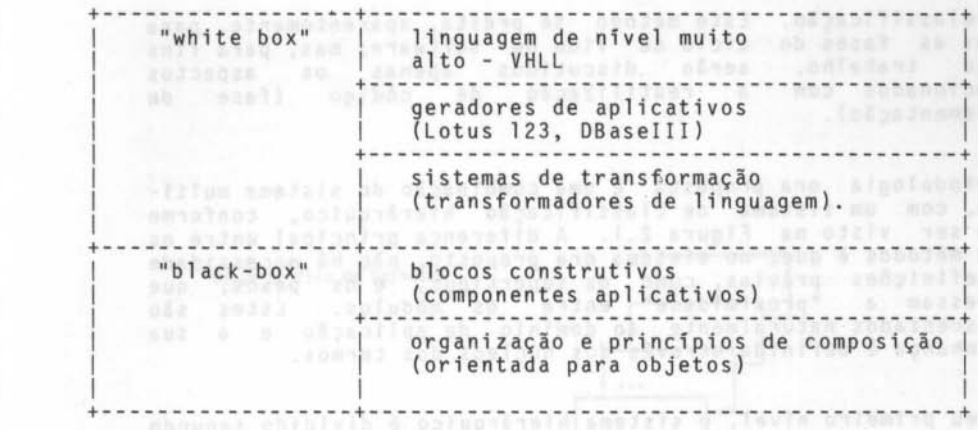


Figura 1.1 Classes de reutilização de software.

## 2. CLASSIFICAÇÃO MULTI-FACE COMBINADA

Existem alguns obstáculos pesados para que a reutilização de componentes aplicativos de software transforme-se em uso corrente. O primeiro é como encontrar um componente para a reutilização; em seguida, há que se entender este componente e, finalmente, a sua adaptação (caso necessite) às novas condições (Biggerstaff, 1987). A solução proposta por Prieto-Diaz (1987) é classificar os componentes de software para reutilização, facilitando a tarefa de busca e recuperação. Quando o número de componentes torna-se grande (este é um dos objetivos da reutilização de software, que haja um grande repositório de componentes), não se pode pensar em inspecionar os componentes um por um até encontrar o que mais se adapte (mais fàcilmente modificado). A proposta é um sistema de classificação multiface, como os usados em bibliotecas científicas, onde cada componente software pode ser identificado por:

a. função que realiza,

b. como realiza a função, e

c. detalhes de implementação.

A vantagem maior de se empregar esta metodologia é a possibilidade de se reutilizar qualquer programa ou sistema hoje existente, mesmo que o sistema não tenha sido desenvolvido pensando-se em reutilização de software. Desta maneira, os repositórios de conhecimentos que são os programas e sistemas existentes, tornam-se disponíveis para a reutilização após a

sua classificação. Este método se presta, aparentemente, para todas as fases do ciclo de vida de software; mas, para fins deste trabalho, serão discutidos apenas os aspectos relacionados com a reutilização de código (fase de implementação).

A metodologia ora proposta é uma combinação do sistema multi-face, com um sistema de classificação hierárquico, conforme pode ser visto na Figura 2.1. A diferença principal entre os dois métodos é que, no sistema ora proposto, não há necessidade de definições prévias, como os supertipos, e os pesos, que expressam a "proximidade" entre os módulos. Estes são acrescentados naturalmente ao domínio de aplicação e a sua vizinhança é definida através dos núcleos dos termos.

No seu primeiro nível, o sistema hierárquico é dividido segundo as diversas fases do ciclo de vida, para acomodar as diversas etapas do desenvolvimento de software. No nível imediatamente abaixo está o domínio (Neighbors, 1984), estruturado segundo a aplicação ("solução conforme a estrutura do problema"). A quantidade de sub-níveis em cada nível, é função:

a. do número de integrantes, pois uma grande quantidade torna difícil a sua manipulação (que consiste basicamente numa inspeção do sub-nível inteiro),

b. de uma eventual especialização (relações de equivalência que provocam a subdivisão), condicionando a criação de um novo sub-nível, que irá comportar mais propriamente determinados integrantes.

Portanto, para poucos integrantes, ou com características pouco contrastantes, o sistema hierárquico reduz-se, expandindo-se com o aumento do número de integrantes, ou de especializações. Note-se que este processo de crescimento é consequência de um maior aprendizado acêrca do domínio em questão, e faz com que as manipulações sejam efetuadas em épocas oportunas. Abaixo da estrutura hierárquica encontramos o sistema de classificação multi-face, representado por uma tripla, formada pelos termos função, objeto e agente (ou meio) e seus núcleos.

## 2.1 A TRIPLA DE CLASSIFICAÇÃO

Os três termos da tripla de classificação, são: função, objeto e agente (ou meio) onde, por função (ou ação) entendemos a atividade primária de um determinado módulo, sendo que um módulo pode ter mais de uma atividade e, em consequência, mais de uma função. A função, encontrada na descrição do módulo, é

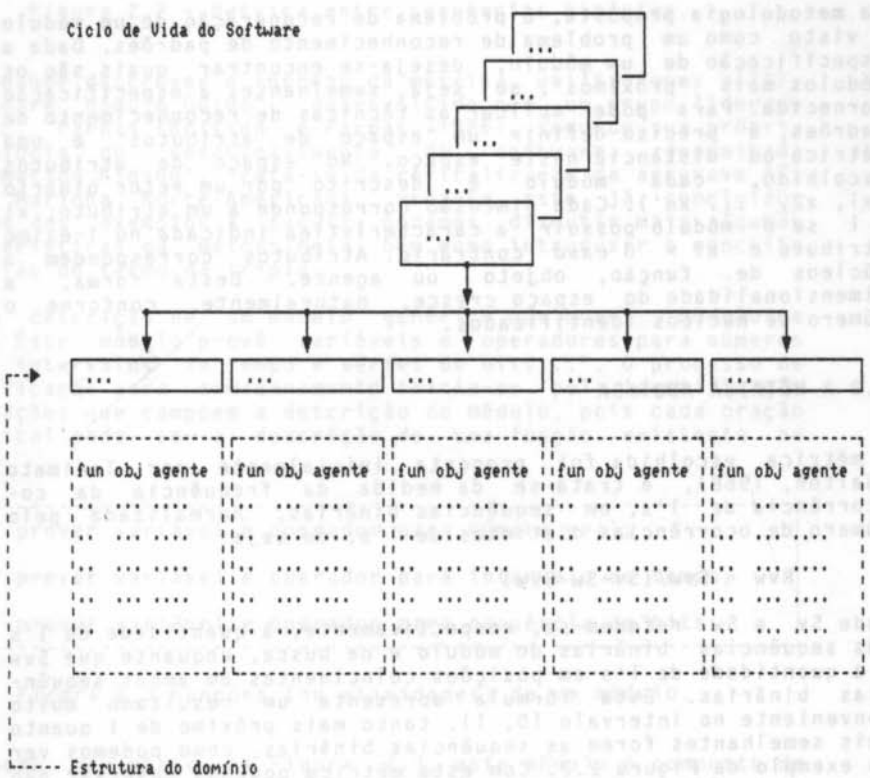


Figura 2.1 Classificação Multi face combinado

normalmente representada por uma oração gramatical e tem um verbo como núcleo. O segundo termo da tripla, objeto, refere-se aos objetos manipulados pelo módulo, tais como caracteres, linhas ou variáveis; o núcleo do objeto é um substantivo singular. Já o terceiro termo da tripla, o agente, refere-se a entidades que servem como locais onde a ação (ou função) é executada, e são representadas por estruturas de dados, como tabelas, arquivos, árvores ou filas, cujo núcleo também é um substantivo singular.

Na metodologia proposta, o problema de recuperação de um módulo é visto como um problema de reconhecimento de padrões. Dada a especificação de um módulo, deseja-se encontrar quais são os módulos mais "próximos", ou seja, semelhantes à especificação fornecida. Para poder aplicar as técnicas de reconhecimento de padrões, é preciso definir um "espaço de atributos" e uma métrica ou distância neste espaço. No espaço de atributos escolhido, cada módulo é descrito por um vetor binário (x1, x2, ... xn'). Cada dimensão corresponde a um atributo: xi = 1 se o módulo possuir a característica indicada no i-ésimo atributo e xi = 0 caso contrário. Atributos correspondem a núcleos de função, objeto ou agente. Desta forma, a dimensionalidade do espaço cresce, naturalmente, conforme o número de núcleos identificados.

## 2.2 A MÉTRICA ADOTADA

A métrica escolhida foi proposta inicialmente por Tanimato (Salton, 1968), e trata-se da medida da frequência da co-ocorrência de 1's, em sequências binárias, normalizada pelo número de ocorrências distintas de 1's, ou seja:

$$Rvw = Svw/(Sv+Sw-Svw)$$

Onde Sv e Sw referem-se, respectivamente, à quantidade de 1's nas sequências binárias do módulo e de busca, enquanto que Svw é a quantidade de 1's em posições coincidentes em ambas sequências binárias. Esta fórmula apresenta um resultado muito conveniente no intervalo (0, 1), tanto mais próximo de 1 quanto mais semelhantes forem as sequências binárias, como podemos ver no exemplo da Figura 2.2. Com esta métrica podemos observar não só as coincidências entre as sequências binárias, como também as diferenças entre elas.

| | | | | |
|---|---|---|---|---|
| módulo A | 11100011 | Sv=5; | Svw=3; | Rvw=0,6; |
| módulo B | 11100000 | Sv=3; | Svw=3; | Rvw=1; |
| módulo C | 11111111 | Sv=8; | Svw=3; | Rvw=0,375; |
| busca | 11100000 | Sw=3; | | |

Figura 2.2 - Métrica entre sequências binárias.

Para melhor descrever o emprêgo da métrica, utilizaremos o "A7-E Software Módule Guide", desenvolvido por um grupo liderado por D.L. Parnas (Britton e Parnas, 1986), segundo sua própria metodologia de desenvolvimento de software, denominada "information hiding". Trata-se da revitalização da aeronave A7-E, da Marinha Norte-Americana, projeto este já concluido. Através dos exemplos a seguir, podemos discutir mais algumas características da metodologia, bem como introduzir o conceito de núcleo do termo da tripla.

Seja a descrição de um módulo genérico qualquer: "Módulo de dados. Êste módulo provê variáveis e operadores para números reais, intervalos de tempo e séries de bits...". O processo de classificação para armazenamento inicia-se pela identificação das orações que compoem a descrição do módulo, pois cada oração gramatical pode ser a descrição de uma função existente no módulo.

| |
|---|
| prover variável e operador para número real; |
| prover variável e operador para intervalo de tempo; |
| prover variável e operador para sequência de bits; |

Figura 2.3 Funções (ou atividades) de um módulo.

Como pode ser visto na Figura 2.3, este módulo é composto de apenas uma função, representada pelo verbo "prover", que manipula alguns objetos, como "variável para número real" e "operador para sequência de bits". O agente (ou meio) é identificado como "extended computer", ou expansão, pois êste módulo, juntamente com alguns outros, provêem as capacidades adicionais necessárias ao sistema operacional do "mission computer", ou seja o computador principal. Observa-se que um único módulo pode ter várias entradas em uma tabela de funções, objetos e agentes; precisamos estabelecer as ligações (comunalidade) entre as diversas atividades, sem contudo retirar informações que as distingam entre si. O exemplo da

Figura 2.3 fica da forma como pode ser visto na Figura 2.4, acrescentando os núcleos para cada termo da tripla:

| função | núcleo da função |
|---|---|
| prover variaveis e operadores | prover |
| objeto | núcleo do objeto |
| prover variável para número real, intervalo de tempo e sequência de bits | variável |
| prover operador para número real, intervalo de tempo e sequência de bits | operador |
| agente (meio) | núcleo do agente |
| extended computer | extensão |

Figura 2.4 Função, objeto, agente e seus núcleos.

A cada módulo será atribuido, então, um código binário, resultado de suas funções, objetos e agentes. A cada atributo existente no domínio (núcleos de função, objeto ou agente) corresponde um caracter na sequência binária identificadora de cada módulo, "1" se estiver presente, e "0", se ausente. Com esta métrica, evita-se novamente a necessidade de tomada de decisão antes da criação do domínio. Existem outras métricas para a classificação, mas a melhoria não é expressiva de métrica para métrica.

| | |
|---|---|
| prover ..... | 1 |
| variável ..... | 10 |
| operador ..... | 100 |
| extensão ..... | 1000 |
| módulo ..... | 1111 |

Figura 2.5 - Exemplo de códigos binários.

A busca é uma comparação entre a sequência binária formada pelos atributos desejados, e a sequência binária identificadora de cada módulo, atribuindo os coeficientes de Tanimato (Rvw). Os módulos que apresentarem valores de Rvw próximos de "1" são aqueles que contém o maior número de coincidências entre funções, objetos e agentes desejados e existentes nos módulos. Um módulo que contenha exatamente os atributos desejadas (funções, objetos e agentes), pode não apresentar valor igual a "1", se, por exemplo, o dado módulo contiver outros atributos adicionais, ou seja, mais funções, objetos ou agentes. Um eventual super-dimensionamento de atributos desejados pode, também, mascarar os resultados, apresentando coeficientes baixos para módulos "bons"; uma segunda rodada, reduzindo a sequência de busca pode conduzir a resultados melhores.

Um valor de classificação entre "1" e "0", obtido na busca, retrata uma relação entre o número de atributos desejados e o número de atributos existentes. Um valor igual a "1" indica a coincidência perfeita, e valor "0" significa que nenhuma dos atributos desejados são encontráveis no domínio em questão. Ressalte-se que o valor de Rvw igual a "0" não é possível, pois a sequência binária de busca é formada a partir de valores existentes no domínio, o que elimina o valor zero.

O valor de Rvw obtido na busca, para cada módulo, dá um idéia da quantidade de esforço que será dispendido para a modificação do módulo encontrado, até transformá-lo no desejado. Porém, um coeficiente de 0.6, não implica em modificar 40% do módulo, para transformá-lo no desejado. A informação exata que se obtém é a de que, dos atributos desejados, o módulo contém um número "x", corrigido pelo número de atríbutos totais do referido módulo. Isto não deve desanimar candidatos a reutilizadores, pois com o emprego de medidas adequadas, advindas principalmente do maior conhecimento do domínio, aliadas às ferramentas inteligentes (Fischer, 1987), pode-se eliminar algumas dificuldades hoje existentes.

## 3. IMPLEMENTAÇÃO DA METODOLOGIA

A metodologia conforme proposta na secção 2 foi implementada, inicialmente, em um protótipo, utilizando um gerador de aplicativos (ou linguagem de quarta geração), a planilha Lotus 123. O objetivo inicial era o detalhamento da interface com o usuário, e a definição dos algoritmos principais (comparação e classificação). O protótipo teve como domínio, o software aviônico da aeronave A7-E, e fase do ciclo de vida, a especificação dos módulos, segundo a metodologia "information hiding" (Parnas, 1986). O protótipo foi construido num prazo curto, mas demonstrou-se lento nas operações de comparação e busca, em virtude dos algoritmos terem sido implementados

utilizando o recurso de "macro", interpretadas em tempo de execução, penalizando portanto, a velocidade de processamento.

O objetivo foi atingido: obteve-se a interface com o usuário, bem como a maioria dos algoritmos principais. Em seguida partiu-se para a implementação em uma linguagem compilável. A escolha recaiu sobre a linguagem C, por sua transportabilidade (que é um modo de reutilização de software). O programa foi escrito com base no protótipo, e atualmente vem sendo testado em uma organização que trabalha com processamento de imagens (de satélites, principalmente), onde pretende-se tomar partido da reutilização de software, para a evolução do sistema de processamento de imagens. A descrição da metodologia, conforme proposta no item anterior, serviu como especificação de requisitos para o desenvolvimento da ferramenta.

## 3.1 A CRIAÇÃO DO DOMÍNIO

Inicialmente deve ser definido o domínio onde se pretende reutilizar software. A estrutura deste domínio irá determinar os vários sub-níveis. Por exemplo, o domínio do software aviônico tem os sub-níveis: utilitários e aplicativos, estes subdividindo-se em navegação, armamento e sensores. Não há necessidade de definição prévia dos sub-níveis, sendo conveniente iniciar a criação do domínio integral, que pode ir sendo subdivido com o tempo, a partir de uma maior compreensão deste.

A inserção das informações no domínio, envolve análise gramatical (sintática) da descrição de cada módulo, identificando orações e, nestas, os predicados e seus complementos. Os verbos fornecem as funções, e dos complementos do predicado saem objetos e agentes. A escolha dos núcleos de cada termo é inicialmente simples, exigindo maior atenção à medida que o domínio cresce, basicamente para atender às exigências de um vocabulário consistente, identificando os sinônimos. Oportunamente, com o aprendizado, podem ser revistas as ligações termo da tripla versus núcleo do termo, ligações estas, que com o passar do tempo tendem a estabilizarem-se.

## 3.2 BUSCA

A busca é uma inspeção do domínio, selecionando-se os termos, que, no entender do usuário, atendem a uma dada especificação. O programa, do modo como está hoje implementado, permite tanto a busca com privilégio para a recuperabilidade, ou seja, examinam-se todas as funções e, em seguida (caso se deseje prosseguir), todos os objetos e todos os agentes, quanto a

busca com privilégio para a precisão, onde a seleção de determinadas funções restringe o número de objetos a serem selecionados, e estes, por sua vez, restringem o número de agentes (Stanfill, 1986). Após a escolha, os módulos são listados em ordem decrescente de satisfação dos requisitos. Podem ocorrer diversas situações, desde vários módulos com coeficientes iguais (ou bastante próximos) a 1, que pode indicar um mau arranjo do domínio em questão, até vários módulos com coeficientes próximos de 0 que, por sua vez, indicam uma especificação de módulo decididamente vaga. O modo de agir, no primeiro caso, evidentemente, é rearranjar o domínio. No segundo caso, deve-se refazer a busca limitando o número de itens selecionados, pois um módulo com várias funções, objetos e agentes, é uma indicação segura de uma escassa modularização. Após uma busca frutífera, não se deve esquecer de instalar no domínio, o novo módulo, oriundo da reutilização de software.

## 3.3 UM EXEMPLO PRÁTICO

Para demonstrar o emprego da metodologia, retiramos do domínio de processamento de imagens, citado anteriormente neste capítulo, este exemplo que tem a seguinte especificação: deseja-se, para uso numa aplicação, uma função que transfere arquivos (classificados) armazenados em fita para disco e vice-versa. Vamos procurar inicialmente imagens (arquivos) classificadas, dentre os objetos, e escolher as funções e os agentes, a partir dos módulos que manipulam imagens classificadas (Figuras 3.1 a 3.3).

| Objeto | Núcleo |
|---|---|
| banda de imagem do disco | imagem |

(a)

| Módulos | Coeficiente de Tanimato |
|---|---|
| le_atributos | 0.33 |
| indice_tipo | 0.33 |
| fecha_imagem | 0.33 |
| escreve_linha | 0.33 |
| escreve_atributos | 0.33 |
| carrega_imagem_classif | 0.33 |
| carrega_imagem | 0.33 |
| abre_imagem | 0.33 |

(b)

Figura 3.1. (a) objeto (e núcleo) selecionado; (b) módulos selecionados.

Pode ser feita então, uma busca restrita àqueles módulos que contém os objetos desejados, em procura de funções e agentes que atendam à especificação. Por busca restrita, entendemos que somente aqueles módulos selecionados no passo anterior, serão apresentados para posterior escolha.

| Função | Núcleo |
|---|---|
| escreve os atributos de um arquivo de imagem | escrever |
| le os atributos de um arquivo de imagem em disco | ler |

(a)

| Módulos | Coeficiente de Tanimato |
|---|---|
| le_atributos | 0.50 |
| escreve_linha | 0.50 |
| escreve_atributos | 0.50 |
| carrega_imagem_classif | 0.20 |
| carrega_imagem | 0.20 |
| abre_imagem | 0.20 |
| fecha_imagem | 0.20 |

(b)

Figura 3.2. (a) função e núcleo (busca restrita);
(b) módulos selecionados.

Também foram apresentados outros módulos (num total de 45), com coeficientes de Tanimato pequenos, que não foram listados por apresentarem apenas um objeto ou uma função, aparentemente pouco relacionados com a presente busca. Afinal, podem ser escolhidos os agentes que atendem à especificação, novamente, dentre aqueles módulos já selecionados.

| Agente | Núcleo |
|---|---|
| arquivo de imagem em disco ou tela | arquivo |
| fita magnetica | fita |

(a)

| Módulos | Coeficiente de Tanimato |
|---|---|
| le_atributos | 0.60 |
| escreve_atributos | 0.60 |
| fecha_imagem | 0.33 |
| escreve_linha | 0.33 |
| carrega_imagem_classif | 0.14 |
| carrega_imagem | 0.14 |
| abre_imagem | 0.14 |

(b)

Figura 3.3 (a) agente e núcleo (busca restrita);
(b) módulos selecionados a partir de (a).

Foram listados 47 módulos. Apresentamos somente aqueles que tinham coeficientes de Tanimato altos, ou que apareciam nos exemplos anteriores. Os coeficientes de Tanimato obtidos representam: 0.60, o módulo tem função (uma das duas funções), objeto e agente (um dos dois agentes) desejados; 0.33, o módulo tem função e agente, ou função e objeto, ou agente e objeto; e finalmente, 0.14, o módulo tem função, ou objeto, ou agente. Os coeficientes de Tanimato, são pequenos em função do número de atributos desejados (cinco).

## 4. CONCLUSÕES

Para aumentar a reutilização de software, é preciso atender às duas exigências mencionadas por Tracz (1987) na introdução. Com esta metodologia (e ferramenta), pretende-se atender a uma parte da segunda exigência, que é a criação de repositórios de itens de software reutilizáveis (classificados, e portanto, acessíveis). Faltam ainda ferramentas capazes de auxiliar o usuário a escolher, na lista dos módulos próximos da

especificação desejada, aquele mais facilmente modificável. Estudos realizados por Woodfield et al (1987) concluem que:

a. desenvolvedores de software, não treinados em reutilização, não são capazes de determinar o item que mais se aproxima de uma dada especificação (utilizando tipos abstratos de dados); tampouco a utilização de uma métrica sugerida pelos experimentadores, baseada em ciência de software e número de linhas de código, apresentou resultados;

b. desenvolvedores de software, não treinados em reutilização, são influenciados por atributos considerados não importantes, como o número de linhas de código e percentual de acréscimo de operações, e não são sensíveis a atributos importantes como o percentual de operadores a serem modificados, e estimativas de esforço de reutilização, empregando-se a métrica sugerida pelos experimentadores.

c. desenvolvedores de software, treinados em reutilização, reutilizam um dado módulo se perceberem que o percentual de esforço para reutilização for inferior a 70 por cento do esforço para criarem o módulo, a partir do zero.

Os caminhos de pesquisa conduzem ao desenvolvimento de ferramentas capazes de eliminar as deficiências acima, possivelmente utilizando inteligência artificial. Tais ferramentas devem auxiliar o usuário a escolher o módulo mais adequado, e a modificá-lo sem perda da sua qualidade intrínseca (na prática, o atendimento das duas exigências mencionadas na introdução).

## 5. BIBLIOGRAFIA

Biggerstaff T. e Richter C., Reusability Framework, Assesment and Directions, IEEE Software, Mar 87, pag. 41-49.

Boehm B W., Improving Software Productivity, IEEE Computer, Set 87, pag. 43-57.

Britton H K, e Parnas D L., A-7E Software Module Guide, NRL Memorandum Report 4702, Dez 86.

Fischer G., Cognitive view of reuse and redesign, IEEE Software, Special Issue, Jul 87, pag. 60-72.

Horowitz E. e Munson B., An expansive view of reusable software, IEEE Transactions on Software Engineering, vol SE-10, n. 5, Set 84, pag. 477-487.

Meyer B., Reusability: the case for object-oriented design, IEEE Software, Mar 87, pag. 50-64.

Neighbors J M., The Draco approach to construting software from reusable components, IEEE Transactions on Software Engineering, vol SE-10, n. 5, Set 84, pag. 564-574.

Parnas D L., SDI - A violation of professional responsability, Abacus, vol. 4, n. 2, Inverno 87, pag 46-52.

Prieto-Diaz R. e Freeman P., Classifying software for reusability, IEEE Software, Jan 87, pag. 6-16.

Salton G., Automatic information organization and retrieval, McGraw-Hill, 1968, capítulo 7.

Stanfill C. e Kahle B., Parallel free-text search on the Connection Machine System, Communications of the ACM, vol 29, n. 12, Dez 1986, pag. 1229-1239.

Tracz W., Reusability comes of age, IEEE Software, Special Issue, Jul 87, pag. 6-8.

Woodfield S. N. et al., Can programmers reuse software ? IEEE Software, Special Issue, Jul 87, pag. 52-59.

Agradecimentos:

Os autores agradecem aos revisores pela pertinência e qualidade das sugestões que contribuiram para a legibilidade do trabalho.