POKE-TOOL – ESTADO ATUAL DE UMA FERRAMENTA PARA TESTE ESTRUTURAL DE SOFTWARE BASEADO EM ANÁLISE DE FLUXO DE DADOS

Marcos L. Chaim
DCA/FEEC/UNICAMP
Av. Albert Einstain, 400 CEP: 13083-900, C. P. 6101,
Campinas – SP, Brasil
CHAIM@DCA.FEE.UNICAMP.BR

Mario Jino
DCA/FEEC/UNICAMP
Av. Albert Einstain, 400 CEP: 13083-900, C. P. 6101,
Campinas – SP, Brasil
JINO@DCA.FEE.UNICAMP.BR

José Carlos Maldonado[2] ICMSC-USP
Av. Dr. Carlos Botelho 1466 Cep: 13560-250, C. P. 369
São Carlos - SP, Brasil
JCMALDON@ICMSC.SC.USP.BR

Elisa Y. Nakagawa

RESUMO

Neste artigo, são apresentados os principais conceitos e problemas envolvidos na definição e implementação de uma ferramenta de apoio ao teste de programas baseado em análise de fluxo de dados. Em especial, é apresentada a POKE-TOOL – ferramenta de apoio à aplicação dos critérios Potenciais Usos [8] – em seu estado atual e na sua configuração futura.

PALAVRAS-CHAVES

Teste Estrutural de Programas, Critérios de Teste, Teste de Software.

ABSTRACT

In this paper we discuss the main concepts and problems related to the definition and implementation of a tool to support Data Flow Based Structural Software Testing Criteria. In particular, it is presented POKE-TOOL – a testing tool that supports the application of the Potential Uses Criteria family [8] – in its current and future configuration.

KEY WORDS
Structural Program Test, Testing Criteria, Software Testing Tool.

1. INTRODUÇÃO

Os critérios baseados em fluxo de dados foram a grande contribuição ao teste estrutural de software dos anos 80 e 90. O teste estrutural de software tem por objetivo executar trechos de programa selecionados como relevantes para a atividades de teste, ou seja, trechos que, em tese, são mais suscetíveis à ocorrência de defeitos. Estes critérios consideram como trechos relevantes aqueles que envolvem a definição de uma variável e o seu subseqüente uso. Utilizando este pressuposto, uma série de critérios para seleção de requisitos de teste foram propostos [4,7,11].

Maldonado *et al.* propuseram os critérios Potenciais Usos (PU) [8] utilizando parte do pressuposto acima. Para estes autores, são trechos interessantes para teste as associações entre a definição de uma variável v e qualquer comando S do programa para o qual existe um caminho entre a definição de v e o comando S, sem a ocorrência de redefinição de v. Assim, o que interessa é o teste entre as definições e os possíveis usos das variáveis, dando origem ao conceito de *Potencial Uso*.

Como é comum a todos os critérios de teste estrutural, a aplicação dos critérios baseados em análise de fluxo de dados não é possível sem o auxílio de ferramenta automatizada. Por isso, na maioria das vezes, os proponentes de novos critérios de teste também procuraram desenvolver ferramentas para evidenciar a aplicabilidade de suas contribuições. Assim, tem-se como exemplos desse tipo de esforço as ferramentas ASSET [3], SADT [6] e POKE-TOOL [2]. Em todos estes casos, ficou provado que os critérios são viáveis na prática e que seu custo de aplicação para programas comuns é de ordem linear [9,13].

Neste trabalho, é apresentada a POKE-TOOL, ferramenta desenvolvida para apoio aos Critérios PU que se encontra em processo de evolução para permitir sua utilização em ambientes industriais. Na Seção 2, faz-se um breve relato evolutivo das ferramentas e são descritos os esforços atuais mais importantes; na Seção 3, discutem-se os aspectos de implementação de uma ferramenta de teste estrutural baseada em análise de fluxo de dados; na Seção 4, é apresentada a POKE-TOOL em seu estágio atual; e, na Seção 5, as conclusões.

2. FERRAMENTAS DE TESTE ESTRUTURAL BASEADO EM FLUXO DE DADOS: EVOLUÇÃO

Ferramentas e critérios de teste estrutural de software são duas coisas inseparáveis. O primeiro critério de teste estrutural de software baseado em fluxo de Dados, proposto por Herman [4], já dispunha de uma ferramenta de apoio à sua aplicação; no caso, para programas em FORTRAN.

Rapps e Weyuker, ao definir sua família de critérios baseados em análise de fluxo de dados, utilizaram uma linguagem muito simples, que não continha mecanismos de modularização (*e.g.*, subrotinas, procedimentos, funções) [11]. A intenção era verificar os conceitos teóricos propostos. Frankl e Weyuker [3], posteriormente, implementaram a ferramenta ASSET para um subconjunto da linguagem Pascal; esta ferramenta não trata atribuição de valores via derreferenciação (através da utilização de ponteiros), não considera campos de registros como variáveis separadas (uma atribuição a um campo é entendida como uma atribuição a todo o registro), o mesmo acontecendo para os elementos de vetores. Apesar das simplificações, alguns experimentos foram realizados e permitiram concluir sobre a viabilidade dos critérios baseados em análise de fluxo de dados [13].

A ferramenta SADT [6] suporta a aplicação do critério *contexto ordenado* [7]. Trata-se apenas de uma ferramenta para demonstração da aplicação dos critérios propostos por Laski e Korel; a linguagem manipulada pela ferramenta é um subconjunto de Pascal com restrições semelhantes às da ferramenta ASSET.

A ferramenta POKE-TOOL foi desenvolvida para auxílio à aplicação dos critérios PU [2]. Seu intuito principal era verificar a viabilidade prática desses critérios; entretanto, desde a primeira versão, seu projeto já a imaginou como uma ferramenta multilinguagem, ou seja, possível de ser configurada para várias linguagens; existem hoje versões para COBOL, FORTRAN, CLIPPER e C. Por isso, desde o seu início, houve a preocupação mais acentuada com aspectos de implementação em linguagens reais; notadamente, com relação a problemas como o tratamento de atribuições por derreferenciação, passagem de parâmetros etc.

TACTIC é a ferramenta desenvolvida por Ostrand e Weyuker [10] para o tratamento de ponteiros. Nesta ferramenta, os critérios da família proposta por Rapps e Weyuker são redefinidos considerando *uso* e *definição* através de ponteiros. Além disso, para suportar esta nova conceituação, faz-se necessário monitorar as posições de memória dinamicamente, no momento da execução dos casos de teste.

Mais recentemente o laboratório BellCore desenvolveu a ferramenta ATAC [5]. Esta ferramenta visa à implementação dos critérios da família proposta por Rapps e Weyuker para a linguagem C. Em ATAC, é utilizada uma abordagem conservadora para o problema de atribuição de valores e o uso de variáveis através de ponteiros. Alguns pontos, entretanto, ficaram sem tratamento adequado, como a passagem de parâmetros por referência.

3. ASPECTOS DE IMPLEMENTAÇÃO

O desenvolvedor de ferramentas para teste de programas baseado em análise de fluxo de dados depara-se com os seguintes problemas: *determinação dos requisitos de teste*, *instrumentação* do programa em teste e *avaliação* da cobertura dos casos de teste. Isto se deve à própria dinâmica de toda ferramenta, que envolve a determinação dos requisitos de teste, a instrumentação do programa em teste (que consiste na introdução de comandos de escrita para registro do caminho percorrido), a submissão de casos de teste e a posterior avaliação da cobertura, através da análise dos caminhos percorridos. Caso a cobertura obtida não seja a desejada, são submetidos novos casos de teste e retoma-se o processo. Entende-se por

cobertura a razão entre o número de requisitos de teste executados e número total de requisitos de teste.

3.1 Determinação dos requisitos de teste

O problema mais difícil é o da determinação dos requisitos de teste porque, em geral, há uma distância entre as proposições dos critérios de teste baseados em fluxo de dados e a sua aplicação em um código propriamente dito. A definição teórica do critério utiliza conceitos fundamentais que variam de linguagem para linguagem. Estes conceitos fundamentais são o modelo de *fluxo de controle* e o modelo de *fluxo de dados*.

O *modelo de fluxo de controle* define como o programa em uma dada linguagem é mapeado para um grafo de fluxo de controle. Em geral, os nós são compostos por comandos executados seqüencialmente e os arcos são transferências de controle devido a comandos de fluxo de controle (*if, while, for* etc). O *modelo de fluxo de dados* define os conceitos de *definição* e *uso* de uma variável. A grosso modo, ocorre a definição de uma variável *v* se esta recebe um valor, seja através de uma atribuição, seja através de um comando de leitura de dados. Há um uso de uma variável se o valor de uma variável *v* é utilizado.

Note-se que definições acima são muito simplistas e não levam em consideração as idiossincrasias de cada linguagem. Assim, ao imaginar-se a aplicação de um dado critério de fluxo de dados deve-se ter em mente qual a linguagem em que será utilizado. Esta decisão influenciará na identificação do modelo de fluxo de controle e no modelo de fluxo de dados.

O problema do modelo de fluxo de controle intra-procedimento aparentemente não é o mais complicado, pois as linguagens procedimentais possuem comandos para controle do fluxo de execução muito semelhantes. Entretanto, apesar de semelhantes, nota-se que todas as linguagens de programação possuem características próprias quanto ao modelo de fluxo de controle.

Portanto, cabe ao projetista da ferramenta definir o mapeamento para a linguagem de programação escolhida. Este mapeamento tem relação direta com o processo de instrumentação do programa fonte, visto que essencialmente é o modelo de fluxo de controle que determinará onde serão colocadas as pontas de prova (comandos de escrita) para registro dos nós percorridos.

O modelo de fluxo de dados é que define o que é uma *definição* e um *uso* de uma variável. É essencial que o projetista estabeleça quais as construções da linguagem alvo serão consideradas como definição e uso. A partir do modelo de fluxo de dados, pode-se determinar as definições e usos das variáveis através da análise sintática do programa. Todavia, há construções comuns à maioria das linguagens e que constituem problemas sem uma solução adequada. Um desses problemas é a questão do tratamento dos sinônimos (*aliasing*); em especial, o tratamento de definições e usos através de ponteiros.

Para este problema algumas soluções foram propostas. Ostrand e Weyuker [10] propõem a reformulação dos critérios, considerando estes tipos de definições. As reformulações propostas implicam na necessidade de monitoração em tempo de execução das posições de memórias utilizadas, para garantir-se a correta avaliação da cobertura dos casos de teste. Outras propostas, como as de Horgan e London [5] e Vilela *et al.* [12], fazem uma análise conservadora da utilização de memória.

A abordagem conservadora, que tanto Horgan e London como Vilela *et al.* propõem, considera que todo ponteiro está sempre apontando para um objeto de memória virtual; este objeto engloba toda a memória do programa. Quando o ponteiro é utilizado para definir uma posição de memória, não importa para onde está apontando, o mesmo objeto está sendo definido, constituindo uma única variável para efeitos de análise de fluxo de dados. Dessa maneira, para a linguagem C, se **p** é um ponteiro apontando para uma certa posição de memória, ***p**, ***(p+i)**, ****p** e **p[i]** referem-se ao mesmo objeto virtual de memória, apesar de especificamente apontarem para posições distintas de memória.

Vilela *et al.* mostram que, para os critérios PU, a suposição acima não implica em alterações nas propriedades de *inclusão* (mesmo na presença de caminhos não executáveis), *complexidade* e *habilidade para detecção de falhas*. Vale ressaltar que estas propriedades somente continuam válidas porque Vilela *et al.* associam a abordagem conservadora e o conceito de Potencial Uso. Outro ponto importante é que não acarreta nenhum dos inconvenientes da abordagem de Ostrand e Weyuker como alteração na formulação dos critérios, alteração do mecanismo de avaliação e a inclusão de requisitos de teste espúrios.

Para determinar os requisitos de teste de qualquer critério baseado em análise de fluxo de dados é necessário o mapeamento do programa fonte para um grafo de fluxo de controle e, a cada nó deste grafo, ter associado o conjunto das variáveis definidas e usadas. As dificuldades para construir-se o grafo de fluxo de controle e os conjuntos de variáveis definidas e usadas em cada nó para linguagens reais foram discutidas acima. A partir deste grafo, pode-se utilizar qualquer algoritmo baseado em estruturas monotônicas de fluxo de dados para determinar os requisitos de teste propriamente ditos, visto que este problema pode ser resolvido através destes algoritmos, conforme demostrado por Chaim *et al.* [1].

3.2 Instrumentação

A instrumentação fornece o programa que será utilizado para a atividade de teste. Este programa contém comandos de escrita que registram os nós efetivamente percorridos pelos casos de teste. A colocação das pontas de prova (comandos de escrita) propriamente dita no programa instrumentado não constitui dificuldade; basta utilizar um analisador sintático para a linguagem do programa e inserir as pontas de prova conforme a definição do modelo de fluxo de controle. Na Figura 1, mostra-se a instrumentação de um comando **for** na versão para a linguagem C da ferramenta POKE-TOOL.

```
main()
/* 1 */      {
             FILE * path = fopen("main/path.tes","a");
             static int printed_nodes = 0;
             ponta_de_prova(1);
/* 1 2 3 */ for(i=0;i< 10;i++)
/* 3 */        {
               ponta_de_prova(2);
               ponta_de_prova(3);
/* 3 */        printf("%d\n",i);
/* 3 */        }
            ponta_de_prova(2);
            ponta_de_prova(4);
            fclose(path);
```

/* 4 */ }

Figura 1 - Monitoração do Comando **for** da linguagem C na ferramenta POKE-TOOL.

3.3 Avaliação

A avaliação dos requisitos de teste é a porção mais genérica de uma ferramenta de teste. Em geral, a implementação do algoritmo de avaliação pode ser reutilizada nas versões para as mais variadas linguagens. Isto ocorre porque a avaliação é feita a partir dos caminhos percorridos pelos casos de teste que nada mais são do que uma seqüência de números identificando os nós percorridos para o caso de teste. Com o caminho percorrido, pode-se verificar a adequação do caso de teste utilizando, por exemplo, autômatos finitos. Cada autômato finito representa um requisito de teste a ser coberto; o algoritmo de avaliação consiste em percorrer os autômatos para uma dada seqüência de nós; aqueles que conseguem atingir o estado final foram satisfeitos; os que ficaram em estados intermediários, ainda precisam ser satisfeitos. Para os critérios que exigem o percorrimento de um dado caminho em especial (*e.g.*, todos du-caminhos, todos potenciais du-caminhos) há duas soluções: a utilização de algoritmos de casamento de padrão ou, novamente, a utilização de autômatos finitos.

4. A POKE-TOOL

A ferramenta POKE-TOOL é orientada a sessão de trabalho. O termo **sessão de trabalho ou de teste** é utilizado para designar as atividades envolvendo um teste. O teste pode ser realizado em etapas em que são armazenados os estados intermediários da aplicação de teste a fim de que possam ser recuperados posteriormente. Desse modo, é possível ao usuário iniciar e encerrar o teste de um programa, bem como retomá-lo a partir de onde este foi interrompido.

Basicamente, o usuário entra com o programa a ser testado, com o conjunto de casos de teste e seleciona todos ou alguns dos critérios disponíveis (Todos-Potenciais-Usos, Todos-Potenciais-Usos/Du, Todos-Potenciais-Du-Caminhos, Todos-Nós e Todos-Arcos). Como saída, a ferramenta fornece ao usuário os conjuntos de nós e de **arcos primitivos** (o conjunto de arcos primitivos é composto dos arcos que, uma vez executados, garantem a execução de todos os demais arcos do grafo de programa), o Grafo Def (grafo de fluxo de controle no qual é associado a cada nó o conjunto de variáveis definidas) obtido do programa em teste, o programa instrumentado para teste, o conjunto de associações necessárias para satisfazer o critério selecionado e o conjunto de associações ainda não exercitadas.

Os conjuntos de nós, de arcos primitivos e de associações a serem executadas pelos casos de teste são os *elementos requeridos* pelos respectivos critérios de teste. O programa instrumentado é o resultado da fase de instrumentação da ferramenta que permite que o programa em teste registre os caminhos percorridos pelos casos de teste. Os nós, arcos ou associações executados constituem o resultado da avaliação da adequação do conjunto de casos de teste com relação ao critério escolhido.

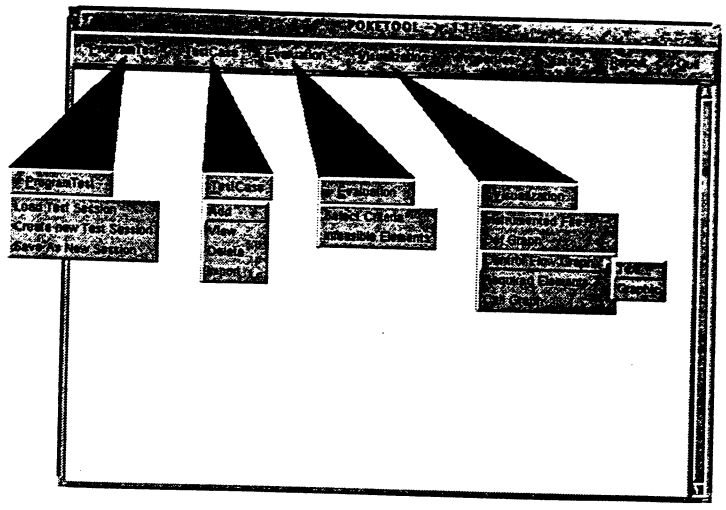


Figura 2. Opções disponíveis na ferramenta POKE-TOOL.

A Figura 2 apresenta as opções disponíveis na POKE-TOOL e a Figura 3 mostra a criação de uma sessão de teste para um programa (função *main*) utilizando todos os critérios apoiados pela ferramenta. Na Figura 4, é apresentada a saída da POKE-TOOL depois da avaliação dos casos de teste submetidos

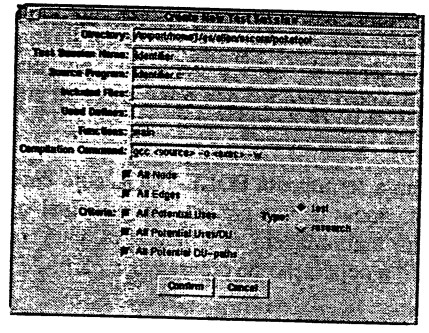
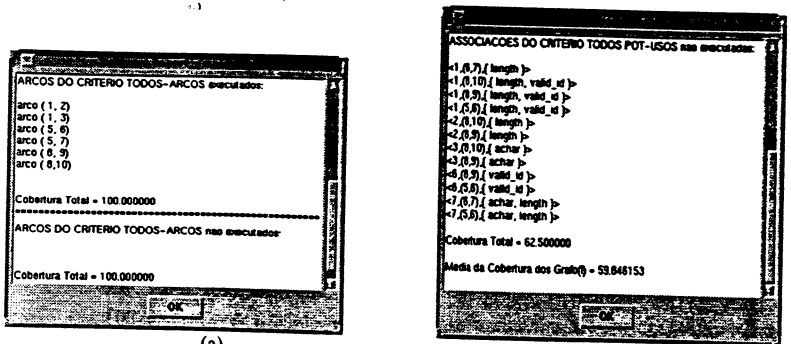


Figura 3. Tela para criar uma sessão de teste na POKE-TOOL.

A POKE-TOOL encontra-se disponível para os ambientes DOS e UNIX. A versão para DOS possui interface simples, baseada em menus. A versão para UNIX possui módulos funcionais cuja utilização se dá através de interface gráfica ou linha de comando (*shell scripts*).



(a) (b)

Figura 4. Relatórios gerados pela *POKE-TOOL* em relação ao programa *identifier*: (a) Arcos executados, (b) Associações não executadas.

5. CONCLUSÕES

Neste trabalho foi apresentada a evolução das ferramentas para aplicação de critérios baseados em análise de fluxo de dados. Observou-se que poucas delas foram pensadas para atuar em linguagens de programação *reais*, visto que seu principal objetivo era verificar formulações teóricas.

Um grande empecilho à difusão das ferramentas de teste baseado em fluxo de dados em ambiente industrial é a falta de ferramentas adequadas para linguagens como C e C++. Todavia, a disponibilidade de ferramentas adequadas para estas linguagens passava antes pela solução do problema do tratamento de usos e definições por derreferenciação. Ora, só recentemente chegou-se a um resultado teórico que equaciona este problema como demonstra Vilela *et al* [12].

Foi apresentada também a ferramenta POKE-TOOL cujo enfoque principal foi a sua utilização em várias linguagens, todas de uso prático. Sua principal versão, a que implementa a linguagem C, pretende incluir a abordagem de Vilela *et al.* para tratamento de ponteiros e também mecanismos de auxílio à depuração de programas a partir de informações resultantes da atividade de teste.

Acredita-se, portanto, que os critérios baseados em fluxo de dados atingiram maturidade teórica suficiente para serem utilizados em ambientes industriais, pois os principais problemas para a implementação de ferramentas adequadas encontram-se resolvidos. As primeiras ferramentas comerciais começam a surgir a exemplo da ferramenta X-ATAC, a versão para ambiente X-View da ferramenta ATAC.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. L. Chaim; J. C. Maldonado e M. Jino. Modelando a determinação de potenciais du-caminhos através da análise de fluxo de dados. Terceiro Simpósio Brasileiro de Engenharia de Software, Recife, PE, Nov. 1989.

[2] M. L. Chaim. POKE-TOOL -- uma ferramenta para suporte ao teste estrutural de programas baseado em análise de fluxo de dados. Dissertação de Mestrado, DCA/FEE/UNICAMP, Campinas, SP, Abr. 1991.

[3] F. G. Frankl e E. J. Weyuker. A data flow testing tool. IEEE Softfair II, Dec. 1985.

[4] P. M. Herman. A data flow analysis approach to program testing. Australian Computer Journal, 8(3), Nov. 1976.

[5] J. R. Horgan e S. London. Data flow coverage and the C language. Fourth Symp. Soft. Testing, Analysis, and Verification, pag. 87-97, Oct. 1997.

[6] B. Korel e J. Laski, A tool for data flow oriented program testing. IEEE Softfair II, Dec. 1985.

[7] J. W. Laski e B. Korel. A data flow oriented program testing strategy. IEEE Trans. Soft. Eng., SE-9(3), Maio 1983.

[8] J. C. Maldonado; M. L Chaim e M. Jino. Seleção de casos de testes baseada em fluxo de dados através dos critérios potenciais usos. Segundo Simpósio Brasileiro de Engenharia de Software, Gramado, RS, Nov. 1992.

[9] J. C. Maldonado; S. R. Vergílio; M. L Chaim e M. Jino. Critérios potenciais usos: análise da aplicação de um benchmark. Sexto Simpósio Brasileiro de Engenharia de Software, Gramado, RS, Nov. 1992.

[10] T. J. Ostrand e E. J. Weyuker. Data flow-based test adequacy analysis for languages with pointers. coverage and the C language. Fourth Symp. Soft. Testing, Analysis, and Verification, pag. 74-86, Oct. 1997.

[11] S. Rapps e E. J. Weyuker. Selecting software test data using data flow information. IEEE Trans. Soft. Eng., SE-11(4), Abr. 1985.

[12] P. R. S. Vilela; J. C. Maldonado; M. L. Chaim e M. Jino. Data flow based testing of programs with pointers: a strategy based on potential uses. 10th International Software Quality Week 1997 - QW97 San Francisco, California, 27-30 de Maio, 1997.

[13] E. J. Weyuker. The cost of data flow testing: an empirical study. IEEE Trans. Soft. Eng., SE-16(2), Fev. 1990.