AMBIENTE DE APOIO AO TESTE ESTRUTURAL DE PROGRAMAS

Ana Maria de Alencar Price *
Avelino Francisco Zorzo **

RESUMO

Este artigo descreve um ambiente interativo de apoio as três fases do processo de teste: planejamento, execução e avaliação. Apesar de orientado ao teste estrutural, o qual baseia-se no código fonte do programa, PROTESTE é um ambiente genérico podendo ser facilmente configurado para aceitar programas em qualquer linguagem bloco-estruturada.

ABSTRACT

This paper describes an interactive tool which is intended to support all three phases of the testing process: planning, execution and evaluation. Although PROTESTE is oriented to the structural testing method, it is a generic tool such that it can easily be configurated to accept programs written in any block-structured language.

* Doutora em Ciência da Computação pela Universidade de Sussex, Inglaterra.
** Mestrando em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

E-Mail: ANAPRICE@SBU.UFRGS.ANRS.BR
Endereço: Instituto de Informática - UFRGS
Caixa Postal 1501 - TELEX (051) 2680
90001 - Porto Alegre - RS

1. INTRODUÇÃO

Das técnicas de verificação de programas, o teste é ainda hoje a mais utilizada. Apesar de realizar uma verificação incompleta, o teste de programas tem sido intensamente aplicado para validar software desenvolvido para solucionar problemas reais.

Dentre as várias atividades pertinentes ao processo de teste, a seleção de dados é a mais crítica, pois requer que os dados sejam escolhidos com o objetivo de encontrar o maior número de erros possível [MEY79]. No caso de programas de grande porte com lógica complexa, a escolha de dados de teste é cansativa, longa e sujeita a omissões. Faz-se necessário o uso de ferramentas automatizadas ou semi-automatizadas que transformem a seleção de dados de teste numa atividade mais rápida e confiável.

Além da seleção de dados, existem outros procedimentos envolvidos no processo de teste, tais como, a geração de módulos condutores de teste, preparação do plano de integração, instrumentação do programa, etc., que quando automatizados facilitam enormemente a tarefa do programador.

Encontram-se descritos na literatura vários tipos de ferramentas de apoio ao teste que implementam uma ou outra das funções mencionadas acima [DEM87]. No entanto, é reconhecida a necessidade de desenvolver sistemas integrados de ferramentas que auxiliem o analista/programdor nas várias etapas do processo de teste.

Este trabalho apresenta a descrição de um ambiente interativo para apoiar o teste estrutural de software. PROTESTE se propõe a assistir o programador tanto no planejamento de testes, auxiliando nas tarefas de seleção de caminhos e dados, quanto na execução, que envolve a realização dos teste com instrumentação e análise dos resultados.

2. PROTESTE - VISÃO DO USUÁRIO

A concepção de PROTESTE foi norteada por dois objetivos principais: desenvolver um ambiente genérico, independente de uma

linguagem de programação especifica, e apoiar o processo de teste desde o projeto de casos de teste até a análise dos testes realizados.

2.1 CONFIGURAÇÃO DO SISTEMA

Apesar de dirigido ao teste estrutural, o qual baseia-se no código fonte do programa, PROTESTE pode ser considerado como um ambiente genérico, independente de linguagem de programação. Esta afirmação decorre do fato de que as funções de PROTESTE que analisam o código do programa em teste podem ser geradas automaticamente através do ambiente SINLEX, que é um gerador de compiladores baseado em gramáticas de atributos [ROS89].

PROTESTE pode ser configurado para uma dada linguagem através da inserção de atributos e ações semânticas (para analisar fluxo de dados e de controle, gerar tabelas de definições e uso de variáveis, inserir sinalizadores no código fonte, etc.) na gramática da linguagem em questão. Submetida a gramática com as ações semânticas ao sistema SINLEX, este gera os procedimentos que compõem um analisador estrutural, dependente de linguagem (Fig. 1), o qual deve ser compilado e ligado com as demais funções do ambiente.

PROTESTE é composto de dois segmentos: o primeiro, responsável pela análise do programa fonte e geração de tabelas, é dependente da linguagem de programacão utilizada pelo usuário; e o segundo, responsável pela análise das tabelas, independe da linguagem de programação (Fig. 2).

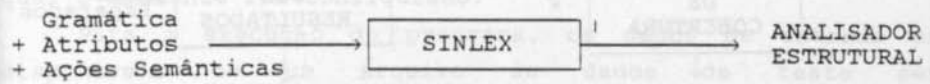


Figura 1 - Configuração de PROTESTE para uma dada Linguagem

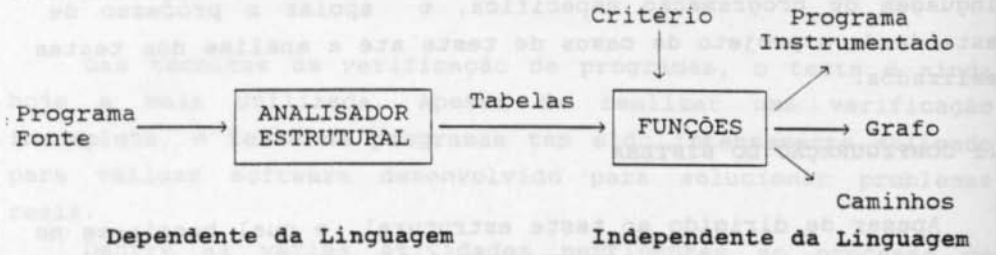


Figura 2 - Segmentos dependente/independentes da Linguagem

2.2 FLUXO DE INFORMAÇÕES

PROTESTE é um sistema interativo que visa apoiar as três fases de teste:

-*planejamento*:auxiliando a seleção de caminhos e dados de teste, e instrumentação do programa fonte;

-*execução*: com monitoramento através de sinalizadores de caminhos; e

-*avaliação* da cobertura dos testes e comparação de resultados computados com resultados esperados.

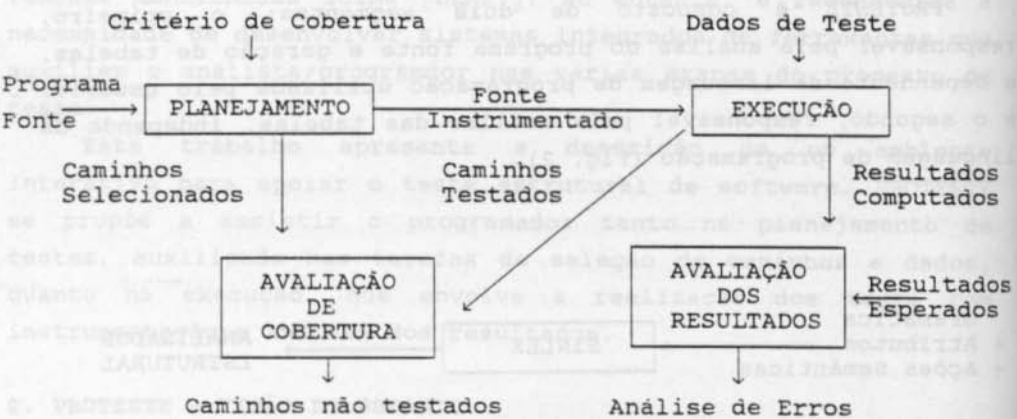


Figura 3 - Funcionamento de PROTESTE na visão do usuário

2.2.1 PLANEJAMENTO

O planejamento de um teste estrutural requer os seguintes passos: escolha de um critério de cobertura; seleção de caminhos que satisfaçam o critério; determinação de casos de teste (dados + resultados esperados) que exercitem os caminhos selecionados; instrumentação do programa fonte com sinalizadores de caminhos; e preparação de "drivers" e "stubs" [MEY79] para a fase de execução dos testes.

PROTESTE implementa uma hierarquia de critérios de cobertura baseados em fluxo de controle e fluxo de dados. Escolhido um dado critério pelo usuário, a ferramenta identifica um conjunto de sub-caminhos que satisfaz o critério selecionado. Apoio à determinação de dados para executar os sub-caminhos selecionados é obtido através da execução simbólica do programa fonte. PROTESTE instrumenta automaticamente o programa inserindo no código fonte comandos de saída que evidenciam os nodos (do grafo de fluxo) visitados quando da execução do programa. O teste de módulo (unidade) requer a preparação de "drivers" e "stubs" para a validação individual de sub-programas. Esqueletos de "drivers" e "stubs" (com declaracão de variáveis, chamadas de procedimentos e comandos de entrada/saida) são gerados automaticamente, podendo ser completados com mensagens e/ou comandos adicionais de modo interativo.

2.2.2 EXECUÇÃO

O programa formado pelo módulo a ser testado + "driver"/"stubs" é submetido ao compilador da linguagem para a qual o ambiente foi configurado.

Para a execução do programa, os dados de entrada estão disponiveis em um arquivo de dados de teste gerado interativamente pelo usuário. A execução do programa produz um arquivo de caminhos que contém todos os caminhos exercitados a partir do conjunto de dados de entrada, e um arquivo de resultados que contém os dados de saída produzidos pelo programa.

2.2.3 AVALIAÇÃO

Após a execução do programa instrumentado, PROTESTE analisa a abrangência dos dados de teste em relação ao critério de cobertura escolhido pelo usuário. Para esta avaliação, cada sub-caminho identificado pelo critério é verificado quanto à sua presença nos caminhos percorridos durante a execução do programa. Se todos os sub-caminhos identificados pelo critério estiverem presentes nos caminhos percorridos então o critério está satisfeito; caso contrário o usuário é informado de quais sub-caminhos ainda devem ser exercitados.

Para cada conjunto de dados submetido, o usuário deve ter especificado previamente os resultados esperados. Após a execução do programa, é verificado se para todo conjunto de dados de teste, a saída esperada pelo usuário confere com o resultado computado. Uma análise dos testes realizados é reportada ao usuário.

2.3 INTERFACE COM O USUÁRIO

O sistema utiliza uma interface orientada a menus, gerada automaticamente pelo UIMS ("User Interface Management System") GIM [ZOR90]. Os menus organizam-se hierarquicamente em forma de árvore. As opções podem ser selecionadas através de caracteres que as identificam ou através de "hot keys" (onde a função associada é executada independentemente do estado em que se encontra o sistema no momento de sua ativação).

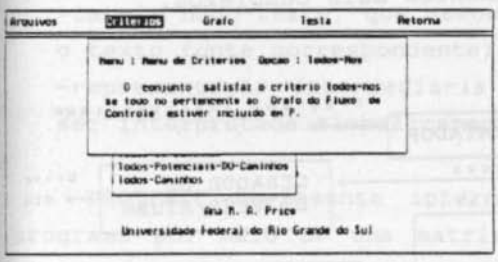
Agregado à interface existe um sistema de auxílio ao usuário ("HELP") que explica a utilização de cada função existente (Figura 4a).

PROTESTE permite visualizar o grafo de fluxo de controle (GFC) do programa em dois níveis: *visualização global*, que é uma redução do grafo original quando este for maior que a tela de exibição (Figura 4b); e *visão detalhada*, que mostra em detalhe parte do grafo do programa. Além destas funções,o usuário poderá visualizar ainda:

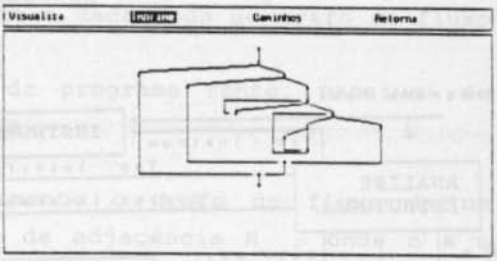
-o subgrafo correspondente a um determinado caminho;

-o programa fonte com o seu respectivo GFC (Figura 4c);

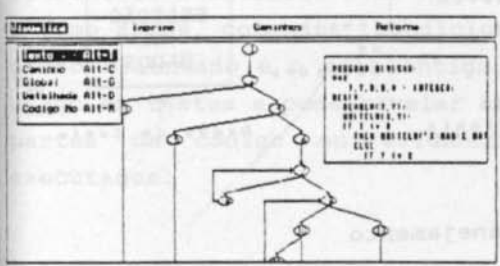
-o texto correspondente a um determinado nodo do GFC (Figura 4d);
-o conjunto de caminhos identificados a partir do critério de cobertura escolhido pelo usuário (Figura 5).

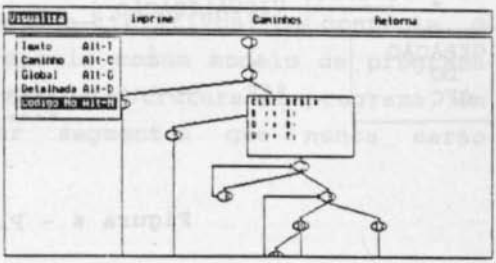


a. Tela de HELP



b. Visão Global





c. Programa Fonte e GFC d. Texto de um Nodo

Figura 4 - Telas da Interface do Ambiente

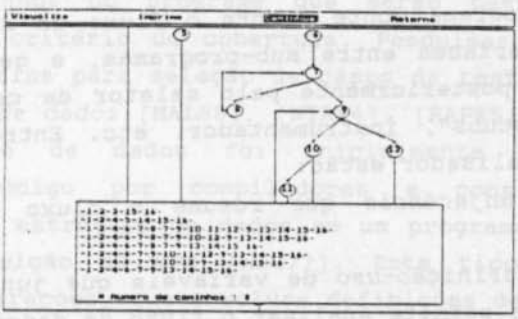


Figura 5 - Conjunto de Caminhos Identificado pelo Critério Todos-Caminhos
(passando uma única vez pelos laços)

3. FUNCIONAMENTO INTERNO DO AMBIENTE

Esta seção descreve sumariamente o funcionamento dos módulos que implementam as funções principais da fase de planejamento de testes que é a que envolve procedimentos mais complexos.

Figura 6 - Planejamento

3.1 ANÁLISE ESTRUTURAL

O analisador estrutural tem como objetivo obter informações, a partir do programa fonte, sobre o fluxo de dados , fluxo de controle e interfaces entre sub-programas, e gerar tabelas que são utilizadas posteriormente pelo seletor de caminhos, gerador de "drivers"/"stubs", instrumentador, etc. Entre as estruturas geradas pelo analisador estão:

-*matriz de adjacência* que resume o fluxo de controle do programa;

-*tabela de definição-uso* de variáveis que junto com a matriz de adjacência permite analisar o fluxo de dados do programa;

-*tabela de interfaces*, a qual contém informações, relativas a cada componente do grafo, sobre parâmetros, variáveis de

entrada e saída e chamadas a subprogramas, e serve de base para a geração de drivers e stubs;

-*tabela de instrumentação*, contém endereços do fonte onde devem ser inseridos comandos que evidenciam os nodos visitados quando da execução do programa;

-*tabela nodo-texto*, que associa a cada nodo do grafo de fluxo o texto fonte correspondente;

-*representação intermediária* do programa fonte, passível de ser interpretada simbolicamente.

PROTESTE representa internamente o grafo do fluxo de um programa por meio de uma matriz de adjacência M_{nxn}, onde n é o número de nodos e $M_{i,j} = 1$ se existe um arco do nodo i para o nodo j. Se tal arco não existir então $M_{i,j} = 0$. Para a construção da matriz de adjacência, e consequentemente do GFC, o analisador estrutural identifica como nodos, segmentos de código sequencial, e como arcos, comandos (condicionais e iterativos) de controle. O grafo orientado é o mais antigo e mais comum modelo de programa usado em testes e pode revelar erros na estrutura do programa, em partes de código ou evidenciar segmentos que nunca serão executados.

3.2 SELEÇÃO DE CAMINHOS

O segundo passo do método estrutural de teste consiste na seleção dos caminhos do programa que serão testados, que é baseada em algum critério de cobertura. Pesquisas recentes têm apresentado critérios para seleção de casos de teste baseados em *análise de fluxo de dados* [MAL88], [NTA84], [RAP85] e [WEY84]. A análise de fluxo de dados foi inicialmente utilizada em otimização de código por compiladores e considera que a ocorrência de uma estrutura de dados em um programa pode ser de dois tipos: *definição* ou *uso* [HEC77]. Este tipo de critério requer que as interações, que envolvem definições de variáveis de programa e subseqüentes referências (usos) afetadas por essas definições, sejam testadas.

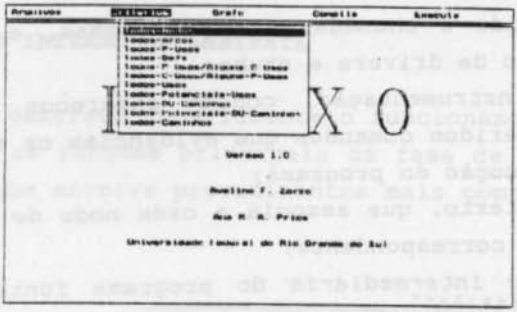


Figura 7 - Critérios de [RAP85] e [MAL88]

A tabela de definições e usos e a matriz de adjacência constituem as entradas para o seletor de caminhos que opera com base no critério de cobertura elegido pelo usuário. O ambiente PROTESTE suporta uma hierarquia de critérios [MAL88](Fig. 7) análoga à proposta em [FRA85] abrangendo critérios de cobertura baseados em fluxo de controle e fluxo de dados. A Figura 9 apresenta a tabela def-uso e o conjunto de caminhos selecionados para satisfazer o critério Todos-Potenciais-DU-Caminhos [MAL88] relativo ao programa da Figura 8.

```
2   BEGIN
      Read(x,y)
      Writeln('x = ',x,' y = ',y);
      If x <= 0
3     Then Writeln('X não é natural')
4     Else If y <= 0
5     Then Writeln('Y não é natural')
6     Else Begin
             r := x;
             q := 0;
             w := y;
7     While w <= r
        Do
8     w := 2 * w;
9     While w <> y
        Do
10    Begin
        q := 2 * q;
        w := w div 2;
        If w <= r
11      Then Begin
               r := r - w;
               q := q + 1;
             End;
12    End;
13    Writeln('Quociente = ',q,' Resto = ',r);
      End;
15    END.
```

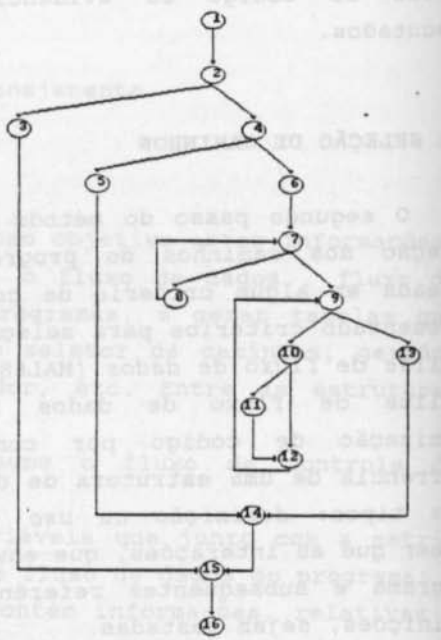


Figura 8 - Programa Fonte e GFC correspondente

Arco	P-Uso
(2,3)	x
(2,4)	x
(4,5)	y
(4,6)	y
(7,8)	w,r
(7,9)	w,r
(8,10)	w,y
(8,13)	w,y
(10,11)	w,r
(10,12)	w,r

Nó	Def	C-Uso
2	x,y	x,y
6	r,q,w	x,y
8	w	w
10	q,w	q,w
11	r,q	r,q,w
13	-	q,r

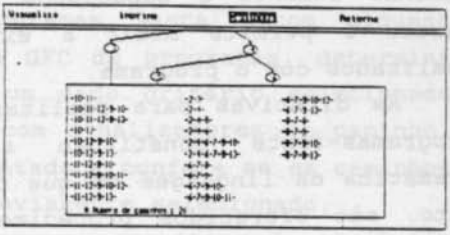


Figura 9 - Tabelas com as definições e usos de variáveis. Caminhos Identificados pelo critério Todos-Potenciais-DU-Caminhos

3.3 PROCESSAMENTO SIMBÓLICO E SELEÇÃO DE CASOS DE TESTE

O código intermediário gerado a partir do programa fonte é interpretado simbolicamente produzindo fórmulas simbólicas como resultado das variáveis e predicados relativos aos caminhos executados. O processamento simbólico implica na execução do programa com valores simbólicos atribuidos às variáveis de entrada, ao invés de dados reais [DEM87]. Se as fórmulas simbólicas resultantes estiverem corretas, o programa está correto para toda a classe de valores de entrada. Os predicados de caminho definem conjuntos do domínio dos dados de entrada cujos elementos causam a execução do caminho correspondente. O seletor de dados verifica e registra os dados de teste sugeridos pelo usuário e que satisfazem os predicados obtidos através da interpretação simbólica.

A determinação dos resultados esperados é um processo manual que deve ser realizado a partir das especificações de requisitos do programa. A partir dos dados selecionados e das especificações do software, o usuário calcula os resultados esperados que, com assistência do ambiente são gravados em um arquivo para posterior conferência com resultados computados.

3.4 INSTRUMENTAÇÃO

A instrumentação do programa fonte com sinalizadores de

caminho aumenta a capacidade de deteção de erros durante os testes e permite medir a extensão da cobertura dos testes realizados com o programa.

As diretivas para realizar a instrumentação automática de programas-fonte constituem ações semânticas associadas à gramática da linguagem em que o ambiente será configurado. Para isto, são oferecidos procedimentos que constroem a tabela de instrumentação, com endereços no programa fonte e tipo de comando a ser inserido (saída ou delimitador de bloco).

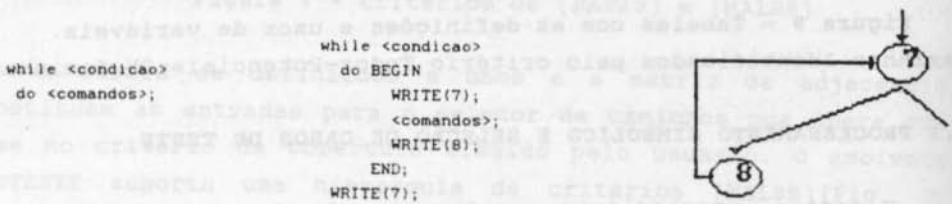


Figura 10 - Exemplo da instrumentação de um comando WHILE-Pascal

3.5 GERADOR DE MÓDULOS CONDUTORES DE TESTE ("DRIVERS" e "STUBS")

Um "driver" é um programa principal que aceita dados de teste e os passa para o módulo a ser testado e mostra os resultados relevantes. Subprogramas "stubs" servem para substituir subprogramas que são chamados pelo módulo que está sendo testado [MEY79].

Esqueletos de "Drivers" e "stubs" são gerados automaticamente a partir dos registros de interface construídos pelo analisador estrutural. Modificações/acréscimos de comandos nestes módulos podem ser efetuados interativamente através de um editor de textos residente no ambiente.

4. CONSIDERAÇÕES FINAIS

Vários segmentos do ambiente descrito foram implementados constituindo um protótipo passível de demonstração. Como a idéia original do projeto era a construção de uma ferramenta de apoio ao teste de programas Pascal, tem-se hoje, implementado um

protótipo [ZOR89] que aceita programas Pascal (com algumas restrições). A ferramenta mostra o GFC de programas, determina caminhos de teste que satisfazem um dado critério selecionado pelo usuário, instrumenta o fonte com sinalizadores de caminho, compila e executa o código instrumentado e confere se os caminhos executados satisfazem o critério previamente selecionado.

Num segundo estágio, definiu-se uma gramática de atributos para o Pascal [TAM89] com ações semânticas para gerar o GFC, fazer análise de fluxo de dados e instrumentar o programa fonte, com o objetivo de gerar automaticamente um analisador estrutural para esta linguagem. Ao analisador gerado acoplou-se as demais funções independentes de linguagem e obteve-se um protótipo que executa as mesmas funções da ferramenta originalmente implementada.

Tem-se implementado, também, um protótipo de processador simbólico que interpreta código intermediário (gerado automaticamente a partir de Pascal) e produz predicados de caminho. Estes predicados são processados por um simplificador de expressões a fim de obter-se expressões simbólicas de mais fácil entendimento ao usuário.

Encontra-se em desenvolvimento o gerador de "drivers"/"stubs" para Pascal. À gramática Pascal foram acrescentados atributos e ações semânticas para gerar registros de interfaces para procedures e funções. O gerador de "drivers" produz automaticamente esqueletos de "drivers"/"stubs" em Pascal que podem ser estendidos com mensagens/comandos através de um editor residente no ambiente.

AGRADECIMENTOS

Vários estudantes colaboraram no projeto e desenvolvimento dos protótipos implementados: Fabio Garcia, Carla Purper, Kaoru Tamura, Vera Lucia Lima, Otilia Werner e Alexandre Migot. Apoio financeiro foi recebido de SID Informática e CNPQ.

BIBLIOGRAFIA

[DEM87] DeMillo R. et alii. Software Testing and Evaluating. Georgia Institute of Tecnology. 1987.

[FRA85] Frankl, F.G.& Weyuker, E.J. A Data Flow Testing Tool. Proc. Softfair II. San Francisco. Dez. 1985.

[HEC77] Hecht, M.S. Flow Analysis of Computer Programs. North Holland, 1977.

[MAL88] Maldonado, J.C. et alii. Seleção de Casos de Teste Baseada em Fluxo de Dados através dos Critérios Potenciais Usos. Anais II SBES, Canela, RS, 1988.

[MEY79] Myers, G. L. The Art of Software Testing. New York, John Wiley & Sons, 1979.

[NTA84] Ntafos, S. C. On Required Element Testing. IEEE Transaction on Software Engineering, New York, 10(6):795-803, Nov. 1984.

[RAP85] Rapps, S. & Weyuker, E.J. Selecting Software Test Data Using Data Flow Information. IEEE Trans. on Software Eng. Vol SE-11 Apr. 1985.

[ROS89] Rosa, Fernando R. SINLEX - Gerador de Processadores de Linguagem. Trabalho de Diplomação. Porto Alegre, Dez. 1989

[TAM89] Tamura, Kaoru. Uma Ferramenta Generica para o Teste Estrutural de Programas. Trabalho de Diplomação. Porto Alegre Dez. 1989.

[WEY84] Weyuker, E. J. The Complexity of Data Flow Criteria for Test Data Selection. Information Processing Letters, Amsterdam, 19(2), Aug. 1984.

[ZOR89] Zorzo, Avelino F. FLUXO - Uma Ferramenta para Testes de Programa. CTIC - Anais IX Congresso da SBC. Jul 1989.

[ZOR90] Zorzo, A.F., Rosa, F.R & Baggio A. GIM - Interfaces Padrão para a Microeletrônica. V SIM - Simpósio Interno de Microeletrônica. Tramandai. 1990.