Métrica de Complexidade de Software baseada em Critério de Seleção de Caminhos de Teste

Juliana Bonzanini da Silva[1]
e-mail: juliana@inf.ufrgs.br

Ana Maria de Alencar Price[2]
e-mail: anaprice@inf.ufrgs.br

Centro de Pós-graduação em Ciência da Computação - Instituto de Informática
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500. Bloco IV. Porto Alegre, RS.

RESUMO

A qualidade tem sido buscada em todas as áreas de produção e de prestação de serviços. O software, como bem comercial, deve ser passível de um controle de qualidade objetivo e eficiente. Através do teste de um sistema, procura-se aumentar a sua confiabilidade, característica fundamental em programas de boa qualidade. Entretanto, é difícil determinar quando parar de testar determinado programa, tornando esta importante fase do ciclo de desenvolvimento do software um período com custos e tempo não previsíveis. A fim de realizar esta atividade de forma organizada e metódica, critérios de seleção de casos de teste baseados em fluxo de dados foram propostos. Estes critérios orientam a tarefa de teste, de forma a torná-la mais confiável. Para prever os recursos que deverão ser dispendidos nesta fase, métricas de complexidade podem ser utilizadas, as quais fornecem números indicativos da complexidade de um determinado programa. O objetivo do presente trabalho consiste na apresentação de uma métrica de complexidade relacionada diretamente a um critério de seleção de caminhos para o teste, baseado no fluxo de dados do programa.

ABSTRACT

Quality is being searched in all production and suppliers subjects. Software, as a comercial good, must be passible of a efficient and objective quality control. By testing a system, one can increase its reliability, which is a very important feature in good quality programs. However, it's difficult to determine when to stop testing a program, making of this important software development cycle's phase period with unpreviewed costs and time. So that realize this task in a organized and methodical way, test case selection criteria based on data flow were proposed. These criteria guide the testing task, with the purpose of becoming it more confiable. To preview the resources to be spent in this phase, complexity metrics can be used, which give complexity indicative numbers which are relationed directly with test ways selection criteria, based on program's data flow.

Palavras Chave: Qualidade, métricas de complexidade e validação de software.

---

[1]Bacharel em Ciência da Computação, ênfase Software Básico, pela UFRGS. Mestranda em Ciência da Computação, na área de Sistemas de Informação, subárea Engenharia de Software, Instituto de Informática, UFRGS. Áreas de pesquisa e interesse: qualidade, teste e métricas de complexidade de software. Bolsista CNPq.

[2]Doutora em Computação pela Universidade de Sussex (UK), 1985. Mestre em Informática pela PUC-RJ, 1976. Professora dos cursos de Bacharelado e Pós-graduação em Ciência da Computação - UFRGS. Áreas de pesquisa e interesse: linguagens de programação e validação de software.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

1. Introdução

O aumento da qualidade no desenvolvimento de produtos e na prestação de serviços [MOS93], relacionado de maneira intrínseca ao aumento do volume da produtividade, está sendo uma das grandes prioridades nas empresas em geral. Empresas cujo produto é a informática (software ou hardware) têm observado o aumento do custo de software e o decréscimo do custo de hardware, devido aos grandes avanços da microeletrônica. Entretanto, enquanto que a produção de hardware tem se aperfeiçoado na relação de 100% a cada três anos, aperfeiçoamentos na produtividade de software têm aumentado na taxa anual de 4% [PUT91].

Quanto mais poderosos ficam os computadores, usuários demandam software mais poderoso e sofisticado, e, conseqüentemente, mais complexo. E, quanto mais complexo é o sistema de software, maiores devem ser os cuidados envolvidos durante a sua produção, com a escolha de metodologias que tornem o processo de desenvolvimento o mais eficiente e controlado possível [GIL92] [MOS93]. Ou seja, deve-se tornar o produto software passível de avaliação para o seu controle de qualidade.

O teste é um instrumento de avaliação da qualidade de programas. Através da sua utilização, tenta-se produzir programas com maior confiabilidade, característica fundamental quando está se abordando qualidade de software. Entretanto, deve-se determinar o momento de finalização dos testes, de forma que o programa tenha sido satisfatoriamente testado utilizando-se o mínimo de recursos possível. Em [WOO80], [LAS83], [RAP85] e [MAL92], são propostos e estudados critérios de seleção de caminhos para teste com base no fluxo de dados, com o objetivo de suprir as deficiências dos testes baseados no fluxo de controle e de fornecer métodos para melhor organização desta fase do ciclo de desenvolvimento do software.

Com a finalidade de fornecer medidas objetivas de qualidade de software, várias métricas de complexidade de software têm sido propostas nos últimos anos [MCC76], [HAL77], [CHA79], [HAR81] e [CON86]. Objetivamente, métricas de complexidade são métodos quantitativos que auxiliam o desenvolvedor no aperfeiçoamento da qualidade e produtividade de sistemas de software. As métricas fundamentam-se na observação de que não se pode gerenciar facilmente o que não pode ser medido [MOL93].

Em [SIL94] é apresentado um experimento com métricas de complexidade, com programas prontos, simulando tarefas de teste e depuração de rotinas, e especificações informais de funções, para serem implementadas. O público participante consistiu de aproximadamente 100 alunos de cursos de graduação da Universidade Federal do Rio Grande do Sul (UFRGS). Os programas utilizados neste experimento foram submetidos à análise de complexidade por dez métricas propostas na literatura. As complexidades resultantes foram comparadas entre si e com os tempos utilizados pelos alunos para a realização das atividades.

A partir deste experimento, surgiu a motivação para a realização do presente trabalho. Além da grande importância do controle de qualidade de software, que por si só já motiva a realização

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

de trabalhos na área, verificou-se que alguns fatores que causaram um aumento de tempo para o entendimento do programa, por parte dos alunos, não foram medidos pelas métricas estudadas.

No enfoque de qualidade de software, abordado aqui, são enfatizadas características relacionadas ao teste e à depuração de programas. A partir disto, propõe-se uma métrica que relaciona de forma direta complexidade com a previsão de esforço que será dispendido na fase de teste.

2. Teste de Software

Aproximadamente 50% dos tempos e custos gastos no processo de desenvolvimento de software são dispendidos nas fases de teste e manutenção [MYE79].

"Testar um programa é uma tarefa extremamente criativa e intelectualmente desafiadora" [MYE79]. Há técnicas de validação nas quais o elemento humano é fundamental, tais como inspeção de código, *walkthrough* e o famoso "teste de mesa". Outras podem ser aplicadas manualmente, mas foram idealizadas para utilização através de ferramentas, tais como o teste de casos que cubram todas as instruções do programa, todas as condições, decisões ou estruturas lógicas, bem como o particionamento de dados de teste em classes de equivalência, análise de valores limite, grafos causa-efeito e outros [MYE79]. Há critérios de seleção de teste derivados da análise de fluxo de dados do programa [RAP85], que visam analisar as associações entre definições e usos de variáveis.

Seja qual for a técnica utilizada, o objetivo é encontrar o maior número de erros possível com o menor conjunto de casos de teste. As seguintes características podem ser citadas como fundamentais para a previsão de gastos e de tempo que são dispendidos no teste de módulo: número de caminhos possíveis de execução, complexidade das estruturas de controle (número de decisões e comparações), complexidade das estruturas de dados utilizadas, relações entre definições e usos de variáveis, semânticas distintas de estruturas de controle e tamanho do programa.

Métricas de complexidade podem ser utilizadas para prever, antes do início da fase de testes, quais os módulos do sistema que exigirão maior esforço, por requerer grande quantidade de casos de teste [MCC76]. A partir desta constatação, estes módulos podem ser reestruturados, tendo sua complexidade diminuída. Mesmo não havendo uma reestruturação de módulo, seja qual for o motivo, permanece o principal benefício das métricas, que é a de tornar a fase de testes um período de tempo e custos previsíveis [NEJ88].

3. Experimento com Métricas de Complexidade de Software

O experimento [SIL94] descrito sucintamente a seguir tem a finalidade de fornecer subsídios para a análise comparativa entre dez métricas propostas na literatura. O público participante constituiu-se de alunos da graduação da UFRGS, em um total de 92 alunos.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

O experimento foi composto por quatro atividades distintas distribuídas aos alunos de forma que somente após terminar uma atividade, o aluno receberia a próxima. Na atividade I, os alunos deveriam descobrir um erro de lógica em uma rotina. Na atividade II, deveria ser descoberta a função implementada pelo programa, enquanto que nas atividades III e IV, foi solicitada a escrita de uma rotina, para a função informalmente especificada. No presente trabalho, somente são consideradas as atividades I e II, por serem estas relacionadas diretamente com as atividades de teste e de depuração de programas. No anexo A, encontram-se as versões de programas aplicadas nas atividades I e II.

Os programas utilizados neste experimento foram submetidos à análise de complexidade por dez métricas propostas na literatura: número ciclomático [MCC76], Oviedo adaptada [OVI80] [SIL92], NPath [NEJ88], escopo [HAR81], expressões regulares [MAG81], contagem das linhas de código [HAR82] [CON86], variáveis de controle [MCC78] [MAR85], *software science* [HAL77], métrica de Hansen [HAN78] e extensão do número ciclomático [MYE77]. As complexidades resultantes foram comparadas entre si e com os tempos utilizados pelos alunos para a realização das atividades.

3.1. Análise da Atividade I

Esta atividade tinha o objetivo de simular a tarefa de depuração de um módulo programado por uma pessoa diferente do testador. Foram apresentadas quatro versões distintas implementando algoritmos de ordenação de vetores. Em todas as versões, o cálculo da variável que controla o número de iterações não está correto. Na tabela 1 são apresentadas as médias de tempos, expressos em minutos, para a depuração do erro de cada versão. Na tabela 2, são apresentados os valores de complexidade calculados para as versões.

De acordo com a tabela 2, com relação ao fluxo de controle, as diversas métricas consideraram a versão 1 como a mais complexa devido à utilização de comandos de desvio incondicional (GOTO), considerados prejudiciais às boas práticas da programação [DIJ68]. A versão 4 obteve baixos valores, devido à estruturação de seus comandos. A métrica de Hansen demonstrou não ser adequada para a medição deste conjunto de programas, pois atribuiu a eles valores iguais de complexidade.

Na avaliação da complexidade do fluxo de dados, a métrica de McClure considera a versão 2 mais complexa devido à utilização de uma condição composta. A versão 1 é considerada a mais complexa devido, novamente, à utilização de comandos GOTO, que influenciam o fluxo de dados quando está se buscando definições e usos de variáveis. A versão 4 é menos complexa devido ao número reduzido de variáveis e operações.

Considerando o tamanho do programa, há uma concordância em considerar a versão 1 de maior complexidade e a versão 4 de menor. O volume (*software science*) da versão 2 é menor, pois, como na versão 4, há um uso conciso do vocabulário do programa.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Tabela 1. Médias de tempos da atividade I.

Versão	1	2	3	4
Média	20,22	23,32	21,87	17,23

Tabela 2. Valores de complexidade para programas da atividade I.

Complexidade	Métricas	Versão 1	Versão 2	Versão 3	Versão 4
Fluxo de Controle	*No. Ciclom.*	5	4	4	4
	Oviedo	(19, *)	(14, *)	(14, *)	(13, *)
	NPath	6	4	4	4
	Tx. Escopo	0,36	0,46	0,46	0,46
	Expr. Reg.	86	39	38	23
	Myers	4:4	4:5	4:4	4:4
	Hansen	(3, *)	(3, *)	(3, *)	(3, *)
Fluxo de Dados	*Oviedo*	(*, 21)	(*, 15)	(*, 12)	(*, 9)
	McClure	6	8	6	6
	Hansen	(*, 25)	(*, 23)	(*, 21)	(*, 17)
Tamanho do Programa	*LOC*	18	15	13	10
	SW Science				
	Volume	283,5894	125,0512	224,8105	171,3439
	Nível	0,0641	0,0535	0,0726	0,08
	Esforço	4424,1716	2337,4056	3096,5634	2141,7988

Analisando-se os tempos gastos na depuração dos programas, observou-se, entretanto, que a versão 1, considerada mais complexa pelas métricas, foi a segunda versão com menor média de tempo. Isto pode ser explicado pelo tamanho reduzido dos programas, o que fez com que não houvesse dificuldade excessiva na identificação dos rótulos correpondentes aos comandos GOTO.

A versão 2, devido à utilização de três estruturas de controle diferentes (REPEAT-UNTIL, FOR-DO e IF-THEN) e à codificação composta existente no comando REPEAT-UNTIL, teve um maior tempo associado. Foi considerado difícil entender as distintas semânticas associadas às estruturas. Este aspecto não é medido por nenhuma das métricas aqui apresentadas.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

3.2. Análise da Atividade II

A atividade II apresenta cinco versões de programas que implementam o somatório simples dos números inteiros positivos entre 1 e 18. As dificuldades encontradas nesta atividade podem ser comparadas às dificuldades de manutenção de um módulo mal documentado. Na tabela 3, são apresentados os tempos médios (em minutos) que os alunos utilizaram para descobrir a função, enquanto que na tabela 4, são apresentados os valores de complexidade, calculados de acordo com as dez métricas selecionadas.

Tabela 3. Médias de tempos da atividade II.

Versão	1	2	3	4	5
Média	21,84	7,12	16,54	16,03	22,48

Tabela 4. Valores de complexidade para programas da atividade II.

Complexidade	Métricas	Versão 1	Versão 2	Versão 3	Versão 4	Versão 5
Fluxo de Controle	*No. Ciclom.*	4	2	6	10	11
	Oviedo	(15, *)	(6, *)	(18, *)	(40, *)	(28, *)
	NPath	6	2	9	10	11
	Tx. Escopo	0,6842	0,857	0,2424	0,2025	0,5
	Expr. Reg.	25	10	115	67	43
	Myers	4:4	2:2	3:3	10:10	11:11
	Hansen	(3, *)	(1, *)	(2, *)	(9, *)	(2, *)
Fluxo de Dados	*Oviedo*	(* , 39)	(* , 4)	(* , 11)	(* , 65)	(* , 23)
	McClure	6	2	3	10	3
	Hansen	(* , 36)	(* , 5)	(* , 18)	(* , 40)	(* , 33)
Tamanho do Programa	*LOC*	26	6	17	26	20
	SW Science					
	Volume	391,9730	42	208,4606	532,5474	490,6890
	Nível	0,0755	0,3704	0,1183	0,1129	0,1173
	Esforço	5191,6954	113,3909	1762,1352	4716,9832	4183,1969

A versão 2, conforme apresentado na tabela 4, foi considerada por todas as métricas como sendo a versão menos complexa, devido à sua simplicidade de fluxo de controle (uma única estrutura), ao simplificado fluxo de dados e ao tamanho reduzido. Considerando apenas o fluxo de controle, as versões 4 e 5 foram consideradas mais complexas devido ao grande número de possíveis caminhos de execução. A métrica das expressões regulares considerou a versão 3 como

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

mais complexa devido à utilização de vários comandos GOTO, que resultaram em uma expressão regular extensa e complicada.

Em relação ao fluxo de dados das versões, a versão 4 foi considerada a mais complexa, pois há um grande número de variáveis e operadores sendo utilizados. A complexidade de tamanho do programa marcou como mais complexas as versões 1 e 4, por serem estas maiores.

Em relação aos tempos medidos, para o entendimento das versões 2 e 5, foram dispendidos tempos correspondentes às complexidades (menor e maior, respectivamente). A versão 4, apesar de ser uma das versões mais complexas, teve um baixo tempo associado. Ao contrário do que comumente é apresentado na literatura, o entendimento de um programa com uma série de estruturas do tipo IF-THEN-ELSE aninhadas foi mais simples do que o entendimento do programa com uma estrutura CASE. A versão 1 apresentou um tempo alto devido à utilização de dois vetores e vários valores de índices associados. Nenhuma das métricas considerou a indexação como um fator determinante do aumento da complexidade, pois índices variáveis determinam os elementos referenciados somente em tempo de execução, não permitindo que qualquer elemento seja descriminado através de uma análise estática em tempo de compilação.

4. Métrica Proposta

Constatou-se, a partir dos resultados do experimento anteriormente descrito, algumas deficiências nas métricas analisadas em relação à medição dos seguintes aspectos:

- estruturas de controle diferentes implicam em distintos níveis de entendimento;
- esforço necessário para adaptação do raciocínio quando o programa contém diferentes estruturas de controle;
- variáveis indexadas para a implementação de matrizes.

A métrica aqui proposta visa resolver alguns dos problemas detectados. A complexidade medida é a complexidade relacionada às atividades de teste e manutenção de um módulo. Com este objetivo, foram considerados aspectos que influem no entendimento de um programa, relacionados a seus fluxos de dados e de controle. Para a análise dos fluxos de dados e de controle, é utilizada a representação do programa no formato de grafo de fluxo de controle (GFC). A métrica usa como base um critério de seleção de casos de teste (TODOS-DU-CAMINHOS), baseado na análise do fluxo de dados, e estipula pesos distintos para as diferentes estruturas de controle do programa (complexidade bruta).

4.1. Critério de Seleção de Caminhos TODOS-DU-CAMINHOS

Em [RAP85] é descrita uma família de critérios de seleção de testes, derivados de técnicas de análise do fluxo de dados para módulos do programa. Esta hierarquia de critérios foi definida com o propósito de suprir as deficiências encontradas em critérios de seleção de caminhos que

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

examinam apenas o fluxo de controle do programa. Os critérios propostos associam nodos que incluem definição de variáveis com nodos aonde as variáveis são referenciadas (ou usadas).

A métrica de complexidade proposta neste trabalho, usa como base o critério TODOS-DU-CAMINHOS, por ser um critério bastante poderoso porque requer que uma grande quantidade de caminhos sejam selecionados para satisfazê-lo, sem alcançar um número infinito, como ocorre no critério TODOS-CAMINHOS [RAP85]. O critério TODOS-DU-CAMINHOS faz uso dos seguintes conceitos:

- O grafo de fluxo de controle (GFC) é um grafo direcionado finito contendo um conjunto N de nodos e um conjunto E de arcos [OVI80]. Pode ser caracterizado por uma quádrupla (b,S,f,E) (sendo que o elemento f pode ser substituído pelo conjunto F, que representa todos os nodos de saída do grafo em questão) aonde $S = N - \{f,b\}$.
- Um *caminho* é uma seqüência finita de nodos $(n_1,...,n_k)$, $k>=2$, tal que há um arco de n_i para n_{i+1}, para $i = 1,2,...,k-1$;
- Um caminho completo é um caminho cujo nodo inicial é o nodo de início do GFC e cujo nodo final é o nodo final do GFC;
- O conjunto $def(n_i)$ contém as variáveis que são definidas nodo n_i;
- Seja P um conjunto dos caminhos completos de um determinado programa;
- Um caminho $(n_1,...,n_j,n_k)$ é um du-caminho em relação à variável x se n_1 tem uma definição global de x e:

a. n_k tem um uso da variável x e $(n_1,...,n_j,n_k)$ é um caminho livre de definições da variável x; OU

b. (n_j,n_k) tem um uso da variável x e $(n_1,...,n_j)$ é um caminho livre de definições da variável x.

P satisfaz o critério TODOS-DU-CAMINHOS se para cada nodo n_i e para cada $x \in def(n_i)$, P inclui cada du-caminho em relação a x. Se há múltiplos du-caminhos da definição global até determinado uso, todos devem estar incluídos em P.

Neste trabalho serão considerados os subcaminhos, no lugar dos caminhos completos, sendo nodo inicial aquele em que ocorre a definição da variável e o nodo final o nodo aonde ocorre uma referência da variável.

4.2. Complexidade Bruta dos Nodos

A complexidade bruta de um nodo do GFC é caracterizada pelo tipo de estrutura de controle que ele representa. Ela é calculada para cada nodo, de forma independente da complexidade dos demais, com o objetivo de refletir o nível de dificuldade de entendimento de cada estrutura de controle.

Aos nodos atribue-se as seguintes complexidades brutas, considerando-se um GFC de programas codificados em uma linguagem imperativa (neste trabalho, programas em Pascal são considerados):

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

- nodo representando o teste de uma variável em uma estrutura de controle do tipo CASE: número de alternativas do CASE;
- nodo representando o teste de uma estrutura de controle de seleção IF: 2;
- nodo representando a condição de uma estrutura de controle de repetição do tipo WHILE ou FOR: 3;
- nodo representando o teste de uma estrutura de controle de repetição do tipo REPEAT-UNTIL: 4;
- nodo contendo um bloco básico (comandos seqüenciais): 1;
- Nodo representando um desvio incondicional de controle (GOTO): 8.

O valor da complexidade bruta está relacionado ao fluxo de controle do programa, não sendo influenciado por características relacionadas ao fluxo de dados ou ao tamanho. Desta forma, em um bloco básico de comandos, não há desvio de controle, sendo atribuído ao nodo que contem estes comandos o valor 1.

No caso da estrutura de controle CASE, considera-se o valor de complexidade como sendo o número de alternativas, pois esta estrutura pode ser substituída por uma série de estruturas IF aninhadas. É importante observar que uma série de estrutras IF aninhadas é mais complexa do que a estrutura CASE correspondente (que contém as decisões de maneira mais explícita e, portanto, mais claras para o entendimento).

Considerando-se as estruturas de controle de seleção e de repetição, tem-se a estrutura REPEAT-UNTIL mais complexa do que as demais, pois o seu teste é realizado apenas no final do corpo de comandos. O corpo deve ser executado no mínimo uma vez, e, de acordo com o experimento prático [SIL94], torna-se mais difícil acompanhar a avaliação da condição no final da execução do corpo, do que quando são utilizadas estruturas de controle do tipo WHILE ou FOR, onde o teste da condição é realizado anterior à execução do corpo. A estrutura de controle IF possui complexidade mais baixa porque o teste da condição associada é realizado apenas uma vez, não sendo necessário acompanhar modificações de valores de suas variáveis.

O nodo contendo o desvio incondicional GOTO recebe um valor de complexidade alto com o propósito de representar uma "punição" para o programa que o utiliza. Há vários estudos encontrados na literatura sobre o assunto afirmando como esta estrutura é prejudicial no entendimento de um programa e como ela pode ser substituída. Entre eles, o mais citado é [DIJ68].

4.3. Estruturas de Controle Diferentes

A utilização de estruturas de controle com semânticas diferentes foi uma das características que exigiu maiores tempos dos alunos para a compreensão dos programas [SIL94]. Isto pode ser verificado observando-se a tabela 1, conforme os tempos dispendidos pelos alunos na atividade I e o programa "21" do anexo A. O programa "21" representa a segunda versão dos programas da

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

atividade I, sendo o que teve maior tempo associado para o seu entendimento, pois utiliza três estruturas de controle diferentes, sobre um pequeno escopo de comandos. Esta característica, como exposto anteriormente, não foi considerada por qualquer uma das métricas analisadas.

4.4. Fórmula Geral da Métrica Proposta

Seja (s_1,...,s_k) o conjunto de subcaminhos (n_i,...n_j) selecionados de acordo com o critério TODOS-DU-CAMINHOS, proposto em [RAP85]. A complexidade de um programa é calculada utilizando-se a seguinte fórmula:

$$\sum_{y=1}^{k}\left(\left(\frac{\sum_{z=i}^{j} b(n_{zy})}{j-i+1}\right)\times(\#e+1)\right)$$

Figura 1. Fórmula da métrica.

onde $b(n_{zy})$ indica a complexidade bruta do nodo n_z do subcaminho y e $\#e$ indica o número de estruturas diferentes representadas pelos nodos do subcaminho.

Para cada subcaminho, é realizado o somatório dos valores de complexidade bruta de cada nodo. O valor obtido no somatório é dividido pelo número de nodos do subcaminho, com a finalidade de obter um valor médio de complexidade por nodo. O valor da divisão é, então, multiplicado pelo número de estruturas diferentes do subcaminho, como por exemplo, GOTO, IF, CASE, FOR, WHILE e REPEAT-UNTIL, acrescido de uma unidade, para adequar-se ao caso de o subcaminho possuir apenas nodos com blocos básicos.

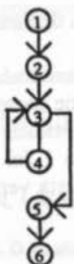
Este cálculo é repetido tantas vezes quantas forem o número de subcaminhos selecionados pelo critério TODOS-DU-CAMINHOS. Os resultados de todos os subcaminhos são somados, resultando na complexidade total do programa. No caso de haver sucaminhos contidos em outros, são considerados, para este cálculo, somente os subcaminhos maiores, a fim de não considerar mais de uma vez determinadas relações entre a definição e uso de determinada variável.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

4.4.1. Exemplo de Cálculo de Complexidade

```
program at22;
var
  c, r : integer;
begin
  r := 0;
  for c := 1 to 18 do
    r := r + c;
  writeln('O resultado eh ', r);
end.
```

Nodo (ni)	Def(i)	Variáveis Usadas
1	——	——
2	{r}	——
3	{c}	{c}
4	{r}	{c,r}
5	——	{r}
6	——	——



SUBCAMINHOS: (2,3,5), (2,3,4), (4,3,5), (3,4,3) e (4,3,4).

(2,3,5) : (5/3)*2 = 10/3
(2,3,4) : (5/3)*2 = 10/3
(4,3,5) : (5/3)*2 = 10/3
(3,4,3) : (7/3)*2 = 14/3
(4,3,4) : (5/3)*2 = 10/3

Soma = complexidade total = 18

Figura 2. Cálculo da métrica a partir de exemplo de programa.

4.5. Valores de Complexidade dos Programas do Experimento

As complexidades calculadas para os programas aplicados no experimento são mostradas nas tabelas 5 e 6:

Tabela 5. Valores de complexidade das versões da atividade I.

Versão	1	2	3	4
Média	38,09	68,54	66,11	24,16

Tabela 6. Valores de complexidade das versões da atividade II.

Versão	1	2	3	4	5
Média	39,45	18	115,24	529,67	588,61

Comparando as tabelas 1 e 5, constata-se que há uma correspondência direta entre os tempos do experimento e os valores de complexidade calculados. Isto ocorreu porque foram considerados aspectos como número de estruturas de controle diferentes, número de comandos

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

GOTO e número de caminhos de execução que, sem dúvidas, influiu nos tempos de depuração das versões.

Com relação às tabelas 2 e 6, verifica-se que não existe uma correspondência total entre os valores encontrados. A métrica classificou as versões na seguinte ordem decrescente de complexidade: 5 > 4 > 3 > 1 > 2, enquanto que o experimento, de acordo com os tempos resultados (em ordem decrescente) clasifica as versões da seguinte forma: 5 > 1 > 3 > 4 > 2.

Esta variação pode ser explicada da seguinte maneira:

- A utilização de matrizes com índices variáveis (versão 1) dificulta a depuração do programa. Esta característica não é medida por esta métrica.
- A versão 4 é composta por uma série de estruturas do tipo IF aninhadas dentro de uma estrutura de repetição WHILE, o que resulta em uma medida alta de complexidade de fluxo de controle. Esta complexidade está relacionada à grande quantidade de subcaminhos existentes. Apesar disso, o experimento demonstrou que a versão 1 foi considerada mais difícil por causa da lógica complexa do algoritmo, mesmo tendo esta versão um fluxo de controle simplificado.

5. Conclusões

A utilização de métodos para o controle da qualidade de software é uma necessidade cada vez mais evidente, tanto em ambientes acadêmicos como comerciais. A qualidade deve ser assegurada de maneira objetiva, a fim de permitir a comparação entre vários programas.

Uma das maneiras de assegurar-se a qualidade de um programa é tornando-o confiável, com resultados previsíveis e adequados à sua função [MYE79]. Utiliza-se comumente o teste para verificar se um determinado programa atende à sua especificação.

Para testar um programa de maneira mais eficiente, deve-se fazê-lo de forma organizada e metódica, para que não se corra o risco de fazer um teste incompleto ou inadequado. Um dos métodos é a utilização de critérios de seleção de caminhos para teste, a partir de um grafo de fluxo de controle. Em [RAP85], é proposta uma família de critérios de seleção de caminhos baseada no fluxo de dados do programa. No presente trabalho, foi utilizado o critério TODOS-DU-CAMINHOS, por ser, após o critério TODOS-CAMINHOS (que é impraticável, pois há um número infinito de caminhos em programas com laços), o critério mais forte na detecção de erros, dentre os critérios propostos em [RAP85].

A partir do critério TODOS-DU-CAMINHOS, foi proposta uma métrica de complexidade que prevê a quantidade de esforço que deverá ser utilizada para o teste de determinado módulo do sistema. Utilizando esta métrica, pode-se comparar os diversos módulos componentes de um sistema, detectando quais serão os críticos na fase de testes.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

A métrica aqui apresentada foi proposta após a realização de um experimento prático, no qual foram analisadas, de maneira comparativa, dez métricas de complexidade encontradas na literatura [SIL94]. A partir desta análise, verificou-se que algumas características que causaram maiores tempo de entendimento dos programas não foram consideradas pelas métricas. Algumas destas características foram incorporadas à métrica apresentada, de forma a construí-la com o objetivo de relacionar complexidade com o teste de programas.

Comparando os valores de complexidade das versões de programas do experimento, obtidos pela utilização da métrica apresentada, pôde ser observado que há uma correspondência direta entre os tempos utilizados na atividade 1 e os valores de complexidade encontrados. Na atividade 2, a correspondência não foi total, e foram apresentados motivos que provocaram esta discordância.

6. Referências Bibliográficas

[CHA79]
CHAPIN, N. A Measure of Software Complexity. **Proc. of National Computer Conference.** 1979.

[CON86]
CONTE, S. D. et alii. **SW Eng. Metrics and Models.** Benjamin/Cummings Publishing Co. Califórnia. 1986.

[DIJ68]
DIJKSTRA, E. Go To Statement Considered Harmful. **Comm. of the ACM, vol.11(3).** Mar. 1968.

[GIL92]
GILLIES, A. **Software Quality - Theory and Management.** Chapman & Hall Computing, UK, 1992.

[HAL77]
HALSTEAD, M. H. **Elements of Software Science.** Elsevier Computer Science Library. New York. 1977.

[HAN78]
HANSEN, W. Measurement of Program Complexity by the Pair. **SIGPLAN Notices, 13(3).** Mar. 1978.

[HAR81]
HARRISON, W. et alii. A Complexity Measure based on Nesting Level. **SIGPLAN Not.,16(3). Mar. 1981.**

[LAS93]
LASKI, J. W. & KOREL, B. A Data Flow Oriented Program Testing Strategy. **IEEE Trans. on SW Eng., vol. SE-9.** May, 1983.

[MAG81]
MAGEL, K. Regular Expressions in a Program Complexity Metric. **SIGPLAN Notices, 16(7).** Jul. 1981.

[MAL92]
MALDONADO, J. C. et alii. Critérios Potenciais Usos: Análise da Aplicação de um Benchmark. **VI Simpósio de Engenharia de Software.** Nov. 1992.

[MCC76]
McCABE, T. A Complexity Measure. **IEEE Trans. on SW Eng., SE-2(4).** New York. Dec. 1976.

[MCC78]
McCLURE, C. A Model for Program Complexity Analysis. **Proc. of 3rd. Int. Conf. on SW Eng..** 1978.

[MOL93]
MÖLLER, K. H. & PAULISH, D. J. **Software Metrics.** Chapman & Hall Computing. Londres, UK. 1993.

[MOS93]
MOSLEY, D. J. **The Handbook of MIS Application Software Testing.** Prentice-Hall. New Jersey. 1993.

[MYE77]
MYERS, G. J. An Extension to the Cyclomatic Measure of Program Complexity. **SIGPLAN Not., 12(10).** Oct. 1977.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

[MYE79]

MYERS, G. **The Art of Software Testing.** Prentice-Hall. New York. 1979.

[NEJ88]

NEJMEH, B. A. NPATH: A Measure of Execution Path Complexity and its Applications. **Comm. of the ACM, 31(2).** Fev. 1988.

[OVI80]

OVIEDO, E. Control Flow, Data Flow and Program Complexity. **COMPSAC80**, 1980. pp. 146-152.

[PUT91]

PUTMAN, L. Trends in Measurement, Estimation and Control. **IEEE Software**. Mar. 1991. pp. 105-107.

[RAP85]

RAPPS, S. & WEYUKER, E. Selecting Software Test Data Using Data Flow Information. **IEEE Trans. on SW Eng., SE-11(4).** Abr. 1985.

[SIL92]

SILVA, J. B. **Análise Automática de Complexidade Integrada a um Ambiente de Apoio ao Teste de Programas.** Projeto de Diplomação. Instituto de Informática, UFRGS. Nov. 1993.

[SIL94]

SILVA, J. B. **Análise Comparativa de Métricas de Complexidade de Software.** Trabalho Individual. Instituto de Informática. UFRGS. Fev. 1994.

[WOO80]

WOODWARD, M. R. et alii Experience with Path Analysis and Testing of Programs. **IEEE Trans. on SW Eng.**, vol. SE-6. May, 1980.

ANEXO A - VERSÕES DOS PROGRAMAS DO EXPERIMENTO

ATIVIDADE I:

```
{ Ordenacao de um vetor  - Versão 1. }
type
  vet = array[1..n] of integer;
var
  vetor: vet;
  cont : integer;
procedure ordena(var vetor: vet);
label 10, 20, 30;
var
  c1, c2, aux: integer;
begin
  c1 := 0;
  10:c1 := c1 + 1;
  if c1 > n-2
    then goto 30;
  c2 := 0;
  20:c2 := c2 + 1;
  if c2 > n-1
    then goto 10;
  if vetor[c2+1] < vetor[c2]
    then begin
         aux := vetor[c2];
         vetor[c2] := vetor[c2+1];
         vetor[c2+1] := aux;
       end;
  goto 20;
  30:exit;
end;
-------------------------- // --------------------------
{ Ordenacao de um vetor  - Versão 2. }
type
  vet = array[1..n] of integer;
var
  vetor: vet;
  cont : integer;
procedure ordena(var vetor: vet);
var
  c1, c2, aux: integer;
  trocou : boolean;
begin
  c1 := 1;
  repeat
    trocou := false;
    for c2 := 1 to n-1 do
      if vetor[c2+1] < vetor[c2]
        then begin
             aux := vetor[c2];
             vetor[c2] := vetor[c2+1];
             vetor[c2+1] := aux;
             trocou := true;
           end;
    c1 := c1 + 1;
  until (c1 > n-2) or (not trocou);
end;
-------------------------- // --------------------------
{ Ordenacao de um vetor  - Versão 3. }
type
  vet = array[1..n] of integer;
var
  vetor: vet;
  cont : integer;
procedure ordena(var vetor: vet);
label 10;
var
  c1, c2, aux: integer;
begin
  c1 := 0;
  10:c1 := c1 + 1;
  for c2 := 1 to n-1 do
    if vetor[c2+1] < vetor[c2]
```

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

```
     then begin
          aux := vetor[c2];
          vetor[c2] := vetor[c2+1];
          vetor[c2+1] := aux;
        end;
  if c1 < n-2
    then goto 10;
end;
------------------------- // -------------------------
{ Ordenacao de um vetor  - Versão 4. }
type
  vet = array[1..n] of integer;
var
  vetor: vet;
  cont : integer;
procedure ordena(var vetor: vet);
var
  c1, c2, aux: integer;
begin
  for c1 := 1 to n-2 do
    for c2 := 1 to n-1 do
      if (vetor[c2+1] < vetor[c2])
        then begin
              aux := vetor[c2];
              vetor[c2] := vetor[c2+1];
              vetor[c2+1] := aux;
            end;
end;
------------------------- // -------------------------
------------------------- // -------------------------
```

ATIVIDADE II:

```
program at21;
var
  c, cv1, cv2, r: integer;
  a1: array[1..9] of integer;
  a2: array[1..3] of integer;
begin
  c := 1;
  cv1 := 1;
  cv2 := 1;
  r := 0;
  while c <= 18 do
    begin
      a1[cv1] := c+c+1;
      c := c + 2;
      cv1 := cv1 + 1;
    end;
  cv1 := 1;
  while cv1 <= 9 do
    begin
      a2[cv2] := a1[cv1]+a1[cv1+1]+
                 a1[cv1+2];
      cv1 := cv1 + 3;
      cv2 := cv2 + 1;
    end;
  cv2 := 1;
  while cv2 <= 3 do
    begin
      r := r + a2[cv2];
      cv2 := cv2 + 1;
    end;
  writeln('O resultado eh ',r);
end.
------------------------- // -------------------------
program at22;
var
  c, r: integer;
begin
  r := 0;
  for c:=1 to 18 do
    r := r + c;
  writeln('O resultado eh ',r);
end.
------------------------- // -------------------------
program at23;
label 10,20,30,40,50;
var
  r, c: integer;
begin
  r := 0;
  c := 1;
  40:r := r + c;
  50:if c <= 9
    then goto 10
    else goto 20;
  30:writeln('O resultado eh ',r);
  halt;
  10:c := c + 1;
  goto 40;
  20: c := c + 1;
  r := r + c;
  if c = 18
    then goto 30
    else goto 50;
end.
------------------------- // -------------------------
program at24;
var
  c1, r: integer;
begin
  c1 := 1;
  r := 0;
  while c1 <= 9 do
    begin
    if c1 = 1
    then r := r + c1 + 18
    else if c1 = 2
    then r := r + c1 + 17
    else if c1 = 3
    then r := r + c1 + 16
    else if c1 = 4
    then r := r + c1 + 15
    else if c1 = 5
    then r := r + c1 + 14
    else if c1 = 6
    then r := r+c1+13
    else if c1 = 7
    then r:=r+c1+12
    else if c1 = 8
    then r:=r+c1+11
    else r:=r+c1+10;
    c1 := c1 +1;
  end;
  writeln('O resultado eh ',r);
end.
------------------------- // -------------------------
program at25;
var
  c1, r: integer;
begin
  c1 := 1;
  r := 0;
  while c1 <= 9 do
    begin
      case c1 of
        1: r := r + c1 + 18;
        2: r := r + c1 + 17;
        3: r := r + c1 + 16;
        4: r := r + c1 + 15;
        5: r := r + c1 + 14;
        6: r := r + c1 + 13;
        7: r := r + c1 + 12;
        8: r := r + c1 + 11;
        9: r := r + c1 + 10;
      end;
      c1 := c1 + 1;
    end;
  writeln('O resultado eh ',r);
end.
```

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor