VALIDAÇÃO DE MÉTRICAS PARA USO EM MODELOS GERENCIAIS DE DESENVOLVIMENTO DE SOFTWARE NAS LINGUAGENS PASCAL E C

Saulo de Araujo Pereira
FUCAPI
Av. Danilo Areosa, 381
69075 - Manaus - AM

Maria de Fátima Camêlo
J. Antão B. Moura
UFPB/CCT/DSC
Av. Aprígio Veloso, S/N
58100 - Campina Grande - PB
Fax: (083) 321-0781
E-Mail: copin@brfapesp.bitnet

RESUMO

O uso fundamentado de métricas (padrões de medição) é essencial para a execução de projetos de software de forma mensurável, cujo desenvolvimento possa ser previsto, monitorado e aprimorado. Dentre as métricas para a fase de codificação apresentadas na literatura, este trabalho verifica a validade do número ciclomático e do número de linhas de código como base para definir modelos gerenciais simples para a codificação de software em PASCALe C.

Para isto, foi desenvolvida uma ferramenta para coleta automática das métricas e análise estatística utilizando técnicas de regressão. Amostras feitas em mais de 1600 rotinas correlacionam as métricas de interesse e mostram sua validade para se inferir sobre complexidade (qualidade) de código e estimar tempo de desenvolvimento de programa, um parâmetro básico para a gerência mais profissional do desenvolvimento de software.

ABSTRACT

The use of metrics (measurement standards) is essential for higher efficiency in the development of software projects. Amongst the most frequent metrics in the literature, this paper verifies the validity of McCabe's cyclomatic number and the number of lines of executable code as estimators for parameters related to the coding phase of software in PASCAL and C.

For that, a tool was developed for automatic collection and statistical analysis of the metrics. Samples taken from over 1,600 routines correlate the metrics of interest and show their validity to inform on software complexity (quality) and to estimate program development time. Based on these findings, models using these metrics as parameters for the management of software development, can now be proposed and more realistically used.

1 - INTRODUÇÃO

Na instalação de novos sistemas computadorizados, a relação de investimentos em software e hardware cresceu de 11:9 em 1980 para cerca de 16:4 em 1990 [FORT89]. A razão para este índice tão alto tem sido o desenvolvimento muito rápido da tecnologia de hardware. Nenhuma outra tecnologia conseguiu em apenas 30 anos apresentar tamanha evolução na relação custo-desempenho, da ordem de 10^6 [BROO87; BAER84].

O hardware mais potente e barato, por sua vez, permite uma maior popularização e disseminação de computadores na sociedade e o uso desses em aplicações cada vez mais abrangentes e complexas. Isto faz a demanda por software crescer cerca de 10 vezes por década [MIZU83], enquanto que, nesse mesmo período, estima-se que a produtividade de software cresce a uma taxa de apenas 50% [MOAD90]. Em consequência, a indústria de software mostra-se incapaz de suprir tal demanda, apesar de investimentos vultosos anuais. (A estimativa para 1990 era de aproximadamente 250 bilhões de dólares em todo o mundo [BOEH87].)

Tendo como objetivo "produzir o melhor software possível ao menor custo possível" [CARD87, p. 845], a Engenharia de Software [RAMA84; MILL80] tornou-se alvo de muito interesse, tanto acadêmico como comercial. A sua evolução, contudo, é lenta e árdua, sem as grandes revoluções que tanto caracterizam a história do hardware. E ainda hoje, os maiores problemas desta área são os mesmos de 20 anos atrás: baixa produtividade, atrasos de cronograma, gastos acima dos previstos, falta de controle sobre o desenvolvimento e baixa qualidade.

Diante desta situação problemática no desenvolvimento de software, diversas técnicas e metodologias têm sido elaboradas e aplicadas com o intuito de melhorar a qualidade da produção do software e, também, do software em si, o que é cada vez mais essencial neste competitivo mercado. Exemplos mais relevantes dessas técnicas são os modelos de especificação, de representação de dados e de projetos, a programação estruturada, as técnicas orientadas a objetos, as ferramentas CASE (engenharia de software auxiliada por computador), etc [BROO87; RAMA84].

No entanto, uma área fundamental da engenharia de software, relativamente pouco abordada, é a que compreende os aspectos quantitativos da produção. O desenvolvimento de software, como qualquer outra atividade, não pode ser controlado se não pode ser medido. Desta forma, avanços na previsão e controle acontecerão quando se puder medir e estimar propriedades do software com precisão. As métricas, ou propriedades mensuráveis, "trabalham como mecanismos de controle que, além de avaliarem um processo, servem de guia para o aperfeiçoamento deste processo" [MOHA81, p.117]. Este trabalho trata de métricas aplicadas à engenharia de software.

Embora as métricas possam ser aplicadas às diversas fases do ciclo de desenvolvimento de software, a grande maioria das métricas encontradas na literatura se concentra na fase de codificação. As métricas para outras fases mostraram-se até agora bem mais imprecisas, principalmente por dependerem muito da metodologia de desenvolvimento adotada.

Ainda que numerosas métricas tenham sido propostas por pesquisadores desde meados da década de 70, para poucas foram feitos experimentos demonstrando os benefícios de sua utilização. Desta forma, mais do que identificar novas métricas, é essencial hoje, executar testes de validação das existentes, pois só assim elas poderão ser efetivamente adotadas na caracterização e avaliação do software e na previsão de seus atributos [CURT83]. Este estudo busca acrescentar subsídios para a validação de métricas aplicadas à fase de codificação de software.

Entre as diversas métricas para a fase de codificação apresentadas na literatura, se destacam as métricas da Ciência de Software de Maurice Halstead [HALS77] e o Número Ciclomático (**NC**) de Thomas McCabe [MCCA76], pela atenção e críticas recebidas. É importante notar que estas métricas não se restringem à fase de codificação, apenas. Outra métrica popular (e simples de se obter) é o número de Linhas de Código Executável (**LDCe**). Devido às limitações de espaço aqui e face a simplicidade, elegância e abrangência da métrica de McCabe - a qual serve também para avaliar "qualidade" de software produzido, o presente artigo enfoca apenas o Número Ciclomático, adotando a métrica LDCe a título de comparações. Um estudo mais detalhado e completo, incluindo as métricas da Ciência de Software, é relatado em [PERE91].

Em função da revisão bibliográfica que foi realizada em [PERE91], pode-se afirmar que muito pouco foi publicado sobre a validação do NC para programas em Pascal e quase nada quanto à linguagem C - linguagens preferidas hoje em ambientes acadêmicos e para desenvolvimento de software básico.

Este artigo enfoca a adequação de NC (e de LDCe) para modelar o comportamento de programadores em Pascal no ambiente MS-DOS e em C, sob Unix. O estudo da adequação considerou 1670 rotinas, totalizando mais de 50.000 linhas de código fonte, com funcionalidade cobrindo as áreas de tarefas escolares, aplicativos comerciais, processamento de texto e software básico.

1.1 - Resumo das Contribuições

O presente trabalho deve contribuir para a engenharia de software devido aos seguintes pontos:

- expande os horizontes de aplicação de uma métrica simples, mas robusta (como será visto), a ambientes de programação populares (C sob Unix e Pascal sob DOS);
- o estudo de validação oferece uma base a partir da qual modelos de avaliação de produção e produtividade no desenvolvimento de software podem ser construídos devidamente ajustados para refletir características operacionais de produtos de software de qualquer porte;
- o mero fato de serem oferecidos valores para alguns parâmetros da fase de codificação e ter-se demonstrada sua validade para software feito no Brasil tem despertado o interesse de profissionais em adotarem modelos gerenciais baseados nas métricas estudadas para controlar e melhorar a qualidade de nossos produtos de software;
- o incipiente estudo realizado motivará esforços para uma maior abrangência de pesquisa na área com desenvolvimento de mais ferramentas para coleta automática de dados - o que contribuirá para estudos futuros de validação.

1.2 - Organização do Restante do Artigo

O restante do artigo é organizado em mais 4 seções. A seção 2 descreve as abordagens e definições para as métricas de interesse, fazendo uma breve revisão crítica da bibliografia sobre elas. A seção 3 detalha os experimentos do estudo, os critérios adotados e a ferramenta desenvolvida para a validação estatística das métricas. A seção 4 apresenta os resultados das análises efetuadas com códigos em Pascal e em C, comparando-os com conclusões chegadas em outros estudos e a seção 5 oferece conclusões e recomendações para trabalhos futuros.

2 - NÚMERO CICLOMÁTICO E NÚMERO DE LINHAS DE CÓDIGO EXECUTÁVEL

2.1 - Número Ciclomático

O Número Ciclomático (NC) de McCabe [MCCA76] é uma métrica baseada na teoria dos grafos [BERG73] e depende apenas da estrutura de decisões do programa (fluxo de controle) e não do seu tamanho físico. Por esta razão, ela é aplicável não só a programas fonte, mas também a projetos (*designs*) de software, desde que estejam suficientemente detalhados. Um valor de NC "muito alto" pode significar um programa com módulos potencialmente difíceis de testar e fazer manutenção.

A determinação de NC consiste basicamente em associar um programa a um grafo orientado com nodos únicos de entrada e saída. Cada nodo no grafo corresponde a um bloco de código no programa onde o fluxo é sequencial, e os arcos correspondem a ramificações (*branches*)

existentes no programa [HALL84]. O grafo resultante é conhecido como grafo de fluxo de controle do programa.

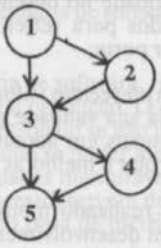
A complexidade do programa, dado o grafo de fluxo de controle correspondente, é o número de caminhos básicos no grafo que, quando combinados, geram qualquer caminho possível entre a entrada e a saída. Este número é o valor de NC, podendo ser expresso como:

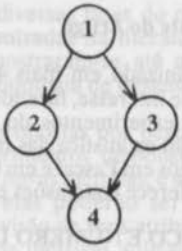
$$NC = a - n + 2p \qquad (2.1)$$

onde: "**a**" é o número de arcos; "**n**" o é número de nodos e "**p**" é número de componentes conectados (rotinas).

Para ilustrar a "utilidade" de NC, suponha um programa que encontre o menor valor entre duas variáveis. Três versões diferentes deste programa, na linguagem C, são apresentadas na Figura1. Os números entre parênteses ao lado do código se referem aos nodos correspondentes no grafo de fluxo de controle de cada versão, mostrados na mesma figura.

Versão	Código	Grafo	Cálculo de NC

1	if (x < =y) (1)
	menor = x; (2)
	if (y >x) (3)
	menor = y; (4)
	.... (5)
			NC = 6- 5+2
			= 3

2	if (x< =y) (1)
	menor = x; (2)
	else
	menor = y; (3)
	... (4)
			NC = 4- 4 +2
			=2

3	menor = min(x,y); (1)		NC = 0- 1+2 = 1

Figura 1: Ilustração do Número Ciclomático de McCabe

Para um programa consistindo de várias rotinas, o cálculo do número ciclomático pode ser feito de duas maneiras. Na primeira, utiliza-se a equação (2.1) para todo o programa, substituindo **p** pelo número de rotinas. Na segunda, calcula-se o número ciclomático para cada rotina (com p = 1), como feito nos exemplos acima, e somam-se os resultados.

Uma forma mais simples de determinar o número ciclomático de um programa é contando o número de decisões (causadas por instruções "if", "while", "repeat", etc.) e operadores booleanos existentes e somar um ao resultado. A razão da inclusão dos operadores booleanos na contagem é que cada um gera uma nova decisão no grafo de estrutura.

2.1.1 - Principais Experimentos com o Número Ciclomático

Os trabalhos mais relevantes acerca da métrica de McCabe verificaram o seu uso como determinadora do limite de complexidade de módulos e como previsora do tempo de programação.

A utilização de NC como determinador do limite máximo de complexidade de uma rotina foi testada em [WALS79], no qual verificou-se que rotinas com complexidade maior que 10 (como sugerido por McCabe) eram muito mais propensas a erros do que aquelas com com plexidade menor. Já [RAMB85] acredita que um valor de complexidade de 14 se correlaciona melhor. Em [SCHN79], um estudo em ALGOL totalizando 31 rotinas, descobriu-se que do total de 64 erros, 40 deles estavam em rotinas cujo número ciclomático era maior que 5.

Com relação ao esforço de desenvolvimento (tempo de programação para as fases de projeto, codificação e testes), os poucos trabalhos encontrados produziram resultados contraditórios. Em [CONT86], um experimento envolveu dois conjuntos de programas em Fortran com 143 e 77 rotinas, desenvolvidos em um concurso na Universidade de Purdue, EUA, para estudantes fluentes na linguagem. O número ciclomático foi relacionado ao tempo total de programação e obteve coeficientes de correlação linear de 0,51 e 0,78. Estes coeficientes são considerados regulares ao serem comparados com as outras métricas estudadas (linhas de código, por exemplo).

Basili et. al. analisaram em [BASI83] um software básico para satélites da NASA, desenvolvido em Fortran, em um total de sete projetos. O coeficiente de correlação linear encontrado entre o número ciclomático e o tempo de programação foi de 0,48, para 215 rotinas retiradas de todos os projetos. Mas, quando as rotinas foram separadas por projeto, este coeficiente foi de até 0,79.

Já em [KOKO88], foi feita uma análise com 19 programas da Faculdade de Ciências Técnicas na Iugoslávia, desenvolvidos em Pascal com 1026 a 4062 linhas de código. A correlação linear encontrada entre o número ciclomático e o tempo de programação foi de 0,938. O tempo de programação incluiu as fases de projeto, codificação, teste e retirada de erros, e foi obtido entrevistando-se os programadores.

O número ciclomático é em sua fundamentação uma métrica intensiva de código, ou seja, não mede a quantidade de código e sim a sua complexidade (de fluxo de controle). Por este motivo, a correlação entre o número ciclomático e o número de linhas de código, tida como uma métrica extensiva, deve ser relativamente baixa.

2.2 - Número de Linhas de Código Executável - LDCe

LDCe é uma métrica de volume, que mede o tamanho do software, largamente usada na estimativa de custo e na avaliação de produtividade e complexidade. Por exemplo, em [GRAD87] cita-se que a produtividade média americana é de 100 a 500 linhas de código por mês por programador. Mas apesar de ser a métrica mais levantada e estudada, ela apresenta grandes limitações (que as outras métricas em geral apresentam, mas em menor grau):

- variação quanto ao método de contagem (inclusão ou não de linhas de: declaração, comentário, nulas, continuação, etc.);

- variação quanto à linguagem (um programa em código de máquina, por exemplo, apresenta muito mais linhas que um programa equivalente em Pascal);
- variação quanto ao estilo de programação (um mesmo programa pode ser escrito de forma mais, ou menos condensada);
- não sensibilidade à qualidade e complexidade.

Apesar destas limitações, esta métrica é incluída no estudo para efeito de comparação e devido a sua popularidade junto a programadores e gerentes de desenvolvimento de software que a utilizam com frequência para estimarem custos e tempo de desenvolvimento, normalmente de forma *adhoc*. Talvez os resultados adiante os dêem razão...

3 - DESCRIÇÃO DOS EXPERIMENTOS E DA FERRAMENTA

3.1 - Parâmetros Estatísticos Adotados nos Experimentos

Os experimentos para coleta das métricas NC e LDCe foram realizados com o apoio da ferramenta "Quantum" [PERE91], com exceção do tempo de projeto e codificação. Não são incluidas neste tempo, as fases de requisitos e especificação, projeto global do sistema (inclusive modularização) e testes. Quando a informação sobre o tempo de implementação não era disponível, o que foi frequente, entrevistaram-se os programadores para se estimar este dado. Nesta estimativa teve-se o cuidado de tentar considerar o tempo realmente gasto na implementação (incluindo código descartado e outros contratempos), ao invés de se estimar o tempo que o desenvolvimento "deveria" ter levado. Toda medida de tempo é dada em minutos neste trabalho.

Para que uma métrica possa ser utilizada efetivamente, é necessário que se faça uma série de testes estatísticos para a sua validação. A importância da elaboração de testes estatísticos cuidadosos foi levantada por vários pesquisadores [MORA78; FENI79; SHEN83], que criticam a validação de métricas baseada apenas no coeficiente de correlação linear.

Neste trabalho, as análises estatísticas foram feitas considerando os seguintes parâmetros:

- coeficiente de correlação linear, **r**;
- desvio relativo médio, **DRM**;
- desvio quadrático médio, **DQM**; e,
- índice de acertos de previsões sob um certo nível de erro **P(e)**.

Além disso, sempre que possível, foram feitas regressões não lineares (logarítimica, polinomial, exponencial e exponencial de base e) ao se tentar inferir a relação entre duas variáveis.

Neste estudo foram adotados limites de aceitação para os parâmetros acima segundo critérios sugeridos em [CONT86] para engenharia de software: **DRM < =0,25; DQM < =0,25 e P(e = 0.25) > = 0,75.**

3.2 - QUANTUM - A Ferramenta para Coleta de Métricas

A ferramenta para a coleta de métricas de programas em Pascal e C, denominada de QUANTUM, foi desenvolvida na linguagem C, sob o ambiente DOS, e tem duas versões: uma para a coleta de métricas de códigos em Pascal e outra para códigos em C. Para facilitar o reconhecimento das sintaxes das linguagens foram também usados os utilitários LEX e YACC. Em

cada versão, a parte escrita em C tem aproximadamente 1500 linhas de código, enquanto as partes escritas para os utilitários LEX e YACC têm cerca de 250 linhas cada.

O funcionamento da QUANTUM pode ser dividido esquematicamente em três módulos. O primeiro módulo, o LEX, lê os caracteres do código fonte, identifica os *tokens* e os passa para o segundo módulo, o YACC. Este módulo é responsável, principalmente, pela coleta das métricas. O terceiro módulo reúne todas as rotinas básicas do programa, utilizadas tanto pelo módulo LEX como o YACC.

3.2.1 - Determinação de NC

A QUANTUM determina o NC de um programa a partir da contagem das instruções condicionais e dos operadores booleanos. Há, porém, uma pequena controvérsia com relação ao operador booleano "not". Enquanto há pesquisadores que incluem o operador no cálculo [SHEN85 e ARTH85] outros não o consideram [CONT86]. McCabe no seu trabalho original [MCCA76] não menciona quais os operadores booleanos que devem influenciar no cálculo. Neste trabalho, preferiu-se considerar o operador "not", pois embora não modifique a estrutura de decisões do programa, ele dificulta a compreensão de um predicado condicional, aumentando assim a complexidade psicológica.

No quadro 1 encontram-se os *tokens* que contribuem para o cálculo do número ciclomático nas linguagens C e Pascal, de acordo com a metodologia aqui adotada.

Linguagem	Instruções	Operadores Booleanos
PASCAL	if, repeat, while, for, case*	and, or, not
C	if, for, while, do,?:, switch*	&&, ||, !
*Ao contrário das outras instruções, case e switch acrescentam ao número ciclomático o número de casos que contêm, não se considerando as cláusulas else (Pascal) e default (C), que não influem no cálculo.		

Quadro 1 - Instruções que Influem no Cálculo do Número Ciclomático

3.2.2 - Determinação de LDCe

Para a QUANTUM, uma linha de código é enquadrada em apenas uma das categorias a seguir (em ordem decrescente de prioridade):

- linha executável;
- linha de declaração;
- linha de diretiva;
- linha de comentário; e
- linha nula.

Se uma linha contiver elementos de duas ou mais categorias, ela se enquadra na categoria de maior prioridade. Por exemplo, uma linha com diretiva e com comentário é considerada uma linha de diretiva. Já uma linha com diretiva e código executável é considerada uma linha executável. Para efeito de cálculo de LDCe, considera-se apenas a primeira categoria.

4 - RESULTADOS

4.1 - Resultados dos Experimentos Realizados com Pascal

Foram analisados 16 programas, totalizando 356 rotinas e mais de 18.000 linhas de código. Os tamanhos dos programas individuais variam de 43 a 3096 linhas de código executável. Além desses programas, foi também considerada uma biblioteca de rotinas básicas, utilizada pelos seis maiores programas analisados. Os programas são, em sua maioria, aplicativos comerciais, tais como: controle de contas a pagar, contas a receber, estoque e aluguel. Outros programas são de engenharia de irrigação e utilitários.

Com o propósito de verificar o comportamento das métricas estudadas e as suposições nas quais se baseiam, buscou-se primeiro determinar os possíveis relacionamentos estatísticos existentes entre as métricas. O melhor ajuste de relacionamento, no caso das 356 rotinas, foi alcançado por regressão linear com **r = 0,897**:

$$\mathbf{LDCe = 9{,}8222 + 2{,}5705*NC} \qquad (4.1)$$

Quando aplicado a programas, ao invés de rotinas, as correlações entre estas métricas tendem a crescer, ultrapassando 0,95, o que está de acordo com a suposição de que a modularização de software diminui a sua complexidade.

A (eventual) validação das métricas como estimadoras do tempo de programação servirá à gerência de desenvolvimento com o seguinte procedimento. Inicialmente, estima-se, a partir da especificação ou projeto do sistema, o valor da métrica que, por sua vez, servirá para a estimativa do tempo de programação.

Neste estudo, foram empregadas regressões lineares e não-lineares, em duas amostras diferentes. A primeira amostra é formada por rotinas e a segunda por programas.

Correlacionando-se o tempo de programação na primeira amostra, consistindo das 356 rotinas, os melhores relacionamentos encontrados foram as regressões lineares seguintes:

$$\mathbf{T(min) = 1{,}041*LDCe \quad e \quad T(min) = 3{,}196*NC} \qquad (4.2)$$

com os seguintes parâmetros:

LDCe: r = 0,863; DRM = 0,419; DQM = 0,756 e P(0,25) = 0,433;

NC : r = 0,853; DRM = 0,578; DQM = 0,782 e P(0,25) = 0,281.

Analisando estes valores, verifica-se que, no geral, a precisão das métricas como estimadoras de tempo de programação de rotinas foi baixa, bem aquém do desejável (segundo critérios em [CONT86]). LDCe saiu-se um pouco melhor que NC, o que porém, não a justifica ainda para uso frequente por gerentes e programadores. Mas, de qualquer forma, até agora nenhuma outra técnica de estimativa publicada é comprovadamente melhor.

Na tentativa de se obter melhores resultados, foram aplicadas regressões para as rotinas divididas por categorias. Os parâmetros estatísticos, no entanto, foram semelhantes aos apresentados. Até mesmo os coeficientes angulares não se alteraram muito, refletindo, portanto, a homogeneidade das rotinas quanto ao ponto de vista do tempo de desenvolvimento.

Correlaciona-se agora, o tempo de programação com a segunda amostra de dados, constituída por 16 programas em Pascal. As métricas dos programas (inclusive o tempo) são obtidas somando os valores das métricas de cada rotina pertencente ao programa.

As melhores relações encontradas foram as regressões lineares:

$$T(min) = 0{,}957*LDCe \quad e \quad T(min) = 3{,}994*NC \tag{4.3}$$

com os seguintes parâmetros:

LDCe: r = 0,934; DRM = 0,197; DQM = 0,233 e P(0,25) = 0,750;

NC : r = 0,943; DRM = 0,196; DQM = 0,299 e P(0,25) = 0,810.

O desvio padrão de estimativa foi de 221 minutos para NC e aproximadamente 170 minutos para LDCe. O intervalo de confiança a 80% é, portanto, de +-218 minutos. Um resultado bom, considerando a média da amostra de tempo de programação de 694 minutos. De acordo com os critérios adotados, LDCe é a única com três parâmetros dentro dos limites desejáveis (bastaria um dos limites atendido para aceitação); entretanto, NC tem também um bom desempenho e é mais fácilmente estimada na fase de projeto. A Tabela 1 mostra os tempos reais e os estimados por LDCe e NC para os 16 programas em Pascal.

Para teste de variação dos coeficientes (angulares) de regres são nas equações (4.3), aplicaram-se regressões nos programas e calcularam-se as variações percentuais nos coeficientes. Na média geral, o coeficiente para LDCe variou 24,7% enquanto que o para NC variou apenas 3,2% com o tipo de programa (NC é portanto, uma métrica mais robusta).

Tempo Real	Tempo estimado	
	LDCe	NC
20	31,6	27,6
30	34,4	31,6
80	66,0	78,9
188	168,4	284,0
247	267,9	248,5
286	271,7	272,1
386	463,1	532,4
430	341,6	323,4
666	379,9	536,4
718	754,0	820,4
826	1033,4	989,9
865	1134,8	1080,6
883	560,7	812,5
1202	1208,5	1001,8
1799	1954,8	2192,9
2483	2173,9	1853,7

Tabela 1: Tempo de Programação Real Versus Estimado

4.2 - Resultados dos Experimentos Realizados com C

Foram analisados três conjuntos de programas, totalizando 1314 rotinas (com 33.554 linhas de código executável) e assim distribuídas: 385 rotinas (com 9.288 linhas) selecionadas aleatoriamente dos programas do processador de textos InfoWord; 697 rotinas, compondo 117 programas e totalizando 18.069 linhas, do gerenciador de disco Unix, DiskManager da Infocon; e, 232 rotinas de exercícios de alunos, compondo 89 programas com um total de 6.197 linhas de código.

O relacionamento entre LDCe e NC, para as 1.314 rotinas, é mostrado na Figura 2. Aqui, para r = 0,879, vale a equação:

$$LDCe = 2{,}982*NC \tag{4.4}$$

Quando as correlações são aplicadas a programas inteiros, em vez de rotinas, r cresce significativamente. A Figura 3 mostra o relacionamento entre LDCe e NC para os 117 programas do DiskManager. Neste caso r = 0,964 e tem-se:

$$LDCe = 7.6143 + 4.0530*NC \tag{4.5}$$

Observando-se as Figuras 2 e 3 (e principalmente, comparando-se as equações 4.1 com 4.4 e 4.5) vemos que o comportamento das métricas muda com a linguagem de programação (e

poderíamos acrescentar: programadores, tipos de programas, etc). Faz-se necessário pois, determinar as relações entre as métricas e dados reais para cada equipe e ambiente de programação, antes que possam ser utilizadas com certo grau de confiança para estimativas de tempo de programação, por exemplo.

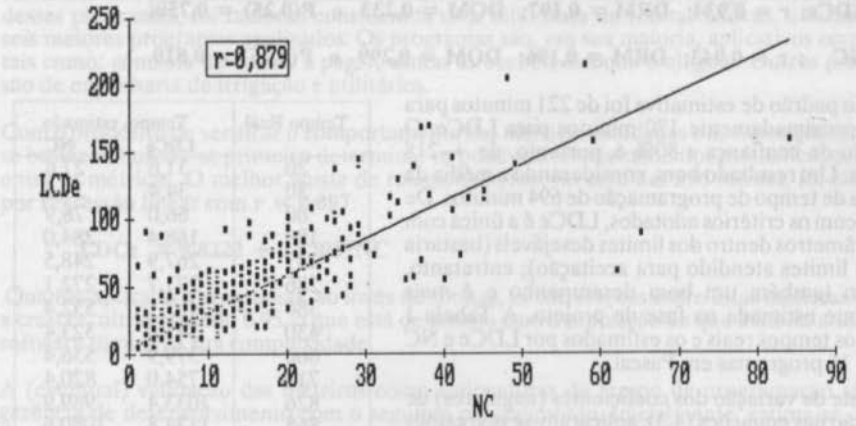


Figura 2: LDCe x NC para 1314 rotinas em C

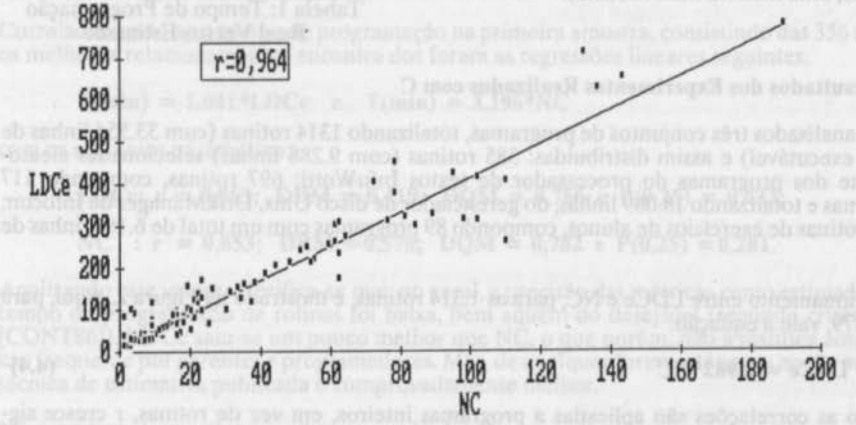


Figura 3: LDCe x NC para 117 programas em C (DiskManager)

Aqui, não apresentamos resultados para a consistência das métricas quanto a tempo de programação devido à falta de informações reais. O estudo encontra-se atualmente em fase de levantamento destes dados. A julgar pelo acompanhamento das relações entre LDCe e NC de Pascal para C, inclusive no tocante a rotinas e programas, é plausível supor correlações semelhantes no caso de se usar estas métricas para estimar tempo de programação em C.

5 - CONCLUSÕES

Resultados preliminares foram apresentados para a validação das métricas Número Ciclomático (NC) e Linhas de Código Executável (LDCe) como métricas válidas para programas (mas, não rotinas) em Pascal e C. Devido à falta de dados reais para os tempos de desenvolvimento dos programas em C, não se avaliou a adequação das métricas como estimadoras de tempo de programação nesta linguagem. A continuação do trabalho está sendo feita neste sentido. Contudo, os elevados índices de correlação entre NC e LDCe observados para Pascal foram mantidos no caso de C, e, como LDCe é uma métrica orientada à "volume" de código, é razoável supor que também NC venha a mostrar adequação satisfatória para estimar tempo em C. Os resultados obtidos até aqui já indicam que investir esforços na definição de modelos gerenciais para Engenharia de Software baseadas nestas métricas pode ser proveitoso.

Os experimentos mostraram uma ligeira superioridade de LDCe sobre NC (maior valor para "r", nas correlações com as amostras) para estimar tempo. Por outro lado, NC mostrou menor variabilidade com o tipo de programa e é uma métrica mais informativa que LDCe, podendo servir para controle de qualidade de software. A quantificação de qualidade é importante para produtores de software comercial e poderia ser feita com um procedimento baseado em NC, semelhante ao que segue. Observe que este procedimento levaria eventualmente a um modelo gerencial de desenvolvimento, válido apenas para a equipe, ambiente, tipos de programas, etc submetidos às medições de que trata os passos do procedimento.

i) A equipe de desenvolvimento coleta dados reais sobre distribuição de falhas (bugs) por linhas de código, bugs removidos antes de teste-alfa, de teste-beta, bugs residuais entregues ao mercado, etc dos programas a medida que são produzidos, testados e evoluem.

ii) Calcular NC no final de cada etapa do desenvolvimento (ex: nível de protótipo, liberação para testes, etc).

iii) Fazer regressões entre NC e os dados de interesse em i).

iv) Definir valores limites para NC que correspondam a níveis aceitáveis de qualidade para a equipe e seus programas (Ex: Não liberar software até que seu "NC" fique aquém do limite aceitável).

Os passos sugeridos exigirão meses para serem realizados. Todavia, os resultados poderão ser significativos com a contribuição que trarão para as equipes em termos de produção, melhoria de qualidade e menos incertezas em suas atividades.

Agradecimentos

Este trabalho está sendo desenvolvido com suporte do CNPq.

REFERÊNCIAS

[ARTH85] L. J. Arthur, **Measuring programmer productivity and software quality**, John Wiley & Sons, EUA, 1985.

[BAER84] Jean-Loup Baer, Computer architecture, **Computer**, vol. 17, nº10, EUA, out/1984.

[BASI83] V.R.Basili, R.W.Selby, Jr e T.Phillips, Metric analysis and data validation across Fortran projects, **IEEE Trans. Software Engineering**, vol. SE-9, nº6, 1983.

[BOEH87] B.W.Boehm, Improving software productivity, **Computer**, EUA, set/1987.

[BROO87] F.P. Brooks, No silver bullet - essence and accidents of software engineering, **Computer**, vol.20, nº4, EUA, abr/87.

[BUCK84] F.J.Buckley, Software quality assurance, **IEEE Trans. Software Engineering**, vol. SE-10, nº1, 1984.

[CARD87] D.N.Card, F.E.McGarry e G.T.Page, Evaluating software en gineering technologies, **IEEE Trans. Software Engineering**, vol. SE-13, nº7, jul/87.

[CONT86] S.D.Conte, H.E.Dunsmore e V.Y.Shen, **Software engineering metrics and models**, Benjamim/Cummings Pub.Inc, EUA, 1986.

[CURT83] B.Curtis, Software metrics: guest editor's introduction, **IEEE Trans. Software Engineering**, vol. SE-9, nº6, EUA, nov/1983.

[FENI79] Robert Fenichel, Heads I win, tails you lose, Correspondência à seção Surveyors' Forum, **Computing Surveys**, vol. 11, nº3, EUA, set/1979.

[FORT89] **Revista Fortune**, EUA, edição de novembro de 1989.

[GRAD87] R.B.Grady e D.L.Caswell, **Software metrics: establishing a company-wide program**, Prentice-Hall, Inc, EUA, 1987.

[HALL84] N.R.Hall e S.Preiser, Combined network complexity measures, **IBM Journal of R&D**, vol.28, nº1, jan/1984.

[HALS77] M.H.Halstead, **Elements of Software Science**, North-Holland, EUA, 1977.

[KOKO88] P. Kokol, B. Ivanek e V. Zumer, Software effort metrics: how to join them, **ACM Sigsoft Software Engineering Notes**, vol. 13, nº2, EUA, abr/1988.

[MCCA76] T.J.McCabe, A complexity measure, **IEEE Trans. Software Engineering**, vol. SE-2, nº4, dez/1976.

[MILL80] H.D.Mills, D.O'Neill, R.C.Linger, M.Dyer e R.E.Quinnan, The management of software engineering, **IBM Systems Journal**, Vol. 19, nº4, EUA, 1980.

[MIZU83] Y.Mizuno, Software quality improvement, **Computer**, vol.16, nº3, EUA, mar/1983.

[MOAD90] J.Moad, The software revolution, **Datamation**, EUA, fev/90.

[MOHA81] S.N.Mohanty, Software cost estimation: present and future, **Software - Practice and Experience**, John Wiley & Sons, vol. 11, EUA, 1981.

[MORA78] P.B.Moranda, Is Software Science Hard?, Correspondência à seção Surveyors' Forum, **Computing Surveys**, vol.10, nº4, EUA, dez/1979.

[PERE91] S. A. Pereira, **Estudo de Validação de Métricas Aplicadas às Linguagens C e Pascal**, Dissertação de Mestrado, UFPb/CCT/COPIN, maio/1991.

[RAMA84] C. V. Ramamoorthy, A. Prakash, W. Tsai e Y. Usuda, Software engineering: problems and perspectives, **Computer**, vol. 17, nº10, EUA, outubro de 1984.

[RAMB85] R. Rambo, P. Buckley e E. Branyan, Establishment and validation of software metric factors, **Proceedings of the International Society of Parametric Analysts Seventh Annual Conference**, EUA, 1985.

[SCHN79] N. F. Schneidewind, An experiment in software error data collection and analysis, **IEEE Trans. Software Engineering**, vol. SE-5, nº3, EUA, maio de 1979.

[SHEN83] V. Y. Shen, S. D. Conte e H. E. Dunsmore, Software Science revisited: a critical analysis of the theory and its empirical support, **IEEE Trans. Software Engineering**, vol. SE-9, nº2, março/1983.

[SHEN85] V.Y.Shen, T.Yu, S.M.Thebaut e L.R.Paulsen, Identifying error-prone software - an empirical study, **IEEE Trans. Software Engineering**, vol.SE-11, nº4, EUA, abr/85.

[WALS79] T.J.Walsh, A software-reliability study using a complexity measure, **Proceedings of National Computer Conference**, AFIPS Press, EUA, 1979.