Mensuração da complexidade de códigos em C com o método do Índice Interno de Esforço

Alisson Antônio de Oliveira[1], Luiz Alberto Pilatti[2],

[1]Doutorando do Programa de Pós-Graduação em Engenharia da Produção – UTFPR – PR - Brasil

[2]Programa de Pós-Graduação em Engenharia da Produção (PPGEP) - Universidade Tecnológica Federal do Paraná (UTFPR) - Ponta Grossa - PR - Brasil

alissonantonio@yahoo.com.br, lapilatti@utfpr.edu.br

Abstract. *The purpose of this article is to present the results of the Internal Effort Index method, created to measure any explicit intellectual activity, in the measuring of the complexity of codes in the C language. With high or extremely high correlation levels, the IIE methodology, in addition to measuring other intellectual activities, such as books, articles, extension activities, etc., was also able to measure the complexity or coding effort in the tested codes, which indicates that it can be used for more than one application, or who knows, be a paradigm shift in the performance evaluation process in intellectual activities in the public service in Brazil.*

Resumo. *A proposta deste artigo é apresentar os resultados do método Índice Interno de Esforço (IIE), criado para mensurar qualquer atividade intelectual explicitada, na mensuração da complexidade de códigos na linguagem C. Com níveis de correlação altos, ou muito altos, a metodologia do IIE além de mensurar outras atividades intelectuais, como livros, artigos, atividades de extensão etc., também conseguiu mensurar a complexidade ou esforço de codificação nos códigos testados, o que indica que ele pode ser usado para mais de uma aplicação, ou quem sabe, ser uma quebra de paradigmas no processo de avaliação de desempenho em atividades intelectuais no serviço público do Brasil.*

1. Introdução

O processo de mensuração de um trabalho ou atividade sempre é difícil. Existem muitas variáveis, como, por exemplo, o retorno do investimento, a necessidade e urgência, o capricho e dedicação na execução, os recursos utilizados etc. As atividades intelectuais, em especial, são ainda mais difíceis de serem mensuradas, uma vez que aos olhos do observador, todas as dificuldades, testes e revisões estão ocultas, o que infelizmente esconde o real esforço para se alcançar o objeto apresentado.

Em softwares o primeiro método conhecido para se mensurar com indicadores um "trabalho" de codificação foi a contagem de linhas de código (*Lines of Code – LoC*) na linguagem *Assembly* (Corrêa, 2011), pois nesta linguagem cada linha gera uma instrução para o processamento. Com o passar dos anos novas linguagens foram sendo criadas e novos métodos desenvolvidos. Nas linguagens de programação modernas uma

linha de código pode conter várias instruções, e por consequência, este método já não é mais eficaz.

Um método clássico para se mensurar a complexidade de um código de computador foi apresentado por McCabe em 1976 (Zuse, 1999) e foi chamado de Complexidade Ciclomática (ou fluxo do grafo - V(G)). De forma resumida, neste método, o número de desvios são contabilizados com a indicação de que cada desvio aumenta a complexidade do código tanto para se confeccionar o código final, bem como para a realização de atualizações ou manutenções.

Segundo Pressman (2011), em 1977 Halstead propôs alguns indicadores de código para serem usados ao final da codificação, bem como para se fazer previsões iniciais de custo, prazo etc. Entretanto Flater (2018) apresentou um conjunto de artigos que comentam sobre a subjetividade de alguns itens propostos por Halstead, ou mesmo, que as conclusões alcançadas não são viáveis. Além disso, as métricas de Halstead não são facilmente encontradas em softwares gratuitos de mensuração de código, o que difere da metodologia da Complexidade Ciclomática de McCabe (1977) ou do número de linhas de código, que é bem difundida e pode ser vista em softwares livres, como, por exemplo, o CCCC (2021) que pode ser facilmente adicionado a IDE Codeblocks.

Para códigos de programação com Orientação a Objetos (OO), por exemplo, metodologias específicas para mensuração de suas especificidades foram criadas (Corrêa, 2011; Zuse, 1999; Pressman, 2011), mas que devido a esta especificidade ir em sentido contrário a robustez proposta pelo Índice Interno de Esforço (IIE), elas não serão mostradas neste artigo, mesmo porque, existem muitas metodologias quando existe foco na especificidade de cada linguagem de programação.

Este artigo apresenta um estudo do Índice Interno de Esforço (IIE) contido no software Mensures®, aplicado na mensuração de códigos padronizados da plataforma Arduino, bem como de outros métodos clássicos de mensuração, como, as pesquisas de campo, LoC e V(G).

2. O Índice Interno de Esforço

O Índice Interno de Esforço (IIE) é o resultado de uma pesquisa interdisciplinar que envolve a Engenharia de Controle de Sistemas, bem como os princípios da Teoria dos Sistemas para gerar indicadores robustos para a área de administração e afins (Mensures, 2016). De forma específica, o IIE foi desenvolvido para medir de forma robusta, objetiva e qualitativa, ou próximo disso, qualquer atividade intelectual explicitada de servidores públicos federais, para auxiliá-los em um sistema de *feedback* e avaliação de desempenho. O foco desta metrificação do trabalho dos servidores públicos está no aumento da eficácia dos resultados entregues a sociedade pagadora de impostos.

O termo robusto empregado no IIE é por motivo técnico, pois ele segue os princípios do Controle Robusto indicados por Cheng, Lemos, Inverardi e Magee (2009), no qual, o controle robusto tolera conhecimento incompleto sobre o modelo matemático do sistema, ou seja, mesmo sem um modelo matemático preciso sobre a atividade que está sendo mensurada, o IIE converge para valores aceitáveis de uso.

O uso do IIE impõem algumas regras para que um erro sistêmico não aconteça no momento da mensuração. Entre estas regras estão:

- Todas as variáveis usadas na mensuração devem obrigatoriamente ser observáveis de forma direta e sem subjetividades ou ambiguidades. Sendo possível a geração destas variáveis de forma indireta se necessário, mas sempre com um número representativo;
- Todas as variáveis selecionadas devem ser apresentadas em ordem de prioridade, podendo as variáveis de mesma amplitude, importância, complexidade ou impacto, serem contabilizadas juntas;
- Para uma estrutura que calcule adequadamente o rendimento ou esforço, as variáveis de saída ou resultados, bem como as de entrada ou insumos (recursos) devem ser usadas. Caso as variáveis selecionadas sejam apenas os insumos (*inputs*) ou apenas os produtos (*outputs*), o resultado da mensuração estará desequilibrado, ou seja, errado;
- Como a complexidade dos itens está fortemente atrelada à quantidade de variáveis, um modelo tetraédrico proposto impõem que sejam usadas no mínimo quatro (4) variáveis distintas (sem correlação) para que o resultado do IIE esteja dentro de uma área robusta e aceitável.

Caso os itens acima são sejam respeitados, a mensuração final pode estar fora da região de robustez. E como o resultado da mensuração é um indicador, caso ele esteja errado naturalmente haverá uma orientação para uma tomada de decisão errada.

Saindo da teoria geral do IIE apresentado acima e indo para o caso específico da dificuldade, complexidade ou esforço de codificação, as variáveis apresentadas abaixo foram as selecionadas para o teste deste artigo. Estas variáveis são básicas na linguagem C, C++ e similares, como, por exemplo, a linguagem Processing que é um *subset* da linguagem C contida no Arduino. Os quatro grupos encontram-se em ordem decrescente de dificuldade e estão na ordem inversa da qual corriqueiramente são lecionadas, ou seja, são lecionados os itens mais simples primeiro para depois os mais difíceis.

I. Iterações e seus controles: For, While, Do While, Continue, Goto;

II. Desvios condicionais e seus controles: IF-Else, Switch/Case, Break;

III. Operadores relacionais e operadores lógicos: &&, ||, !, <, <=, ==, >=, >, !=;

IV. Organizadores de código e de estruturas: chaves {, colchetes [, parênteses (;

A função *break* na linguagem C pode ser usada ou se encontra em mais de um grupo, entretanto, conforme as regras do IIE apresentadas, para minimizar ambiguidades de interpretação ela deve ser contabilizada dentro de apenas um grupo. Outros cuidados são necessários, como, por exemplo, com o carácter de abrir chaves "{" que impõem, segundo a sintaxe da linguagem, um carácter de fechar chaves "}", então para minimizar uma possível dupla pontuação, apenas a abertura do grupo IV foi contabilizado.

Como insumo dos códigos em C foi considerado que apenas uma pessoa desenvolveu os códigos. Outras possibilidades de insumos existem, como, por exemplo, um indicador da dificuldade de cada linguagem de programação, mas como tal indicador não é trivial, ele será desconsiderado nesta pesquisa.

Discussões sobre outras variáveis podem surgir, entretanto, como existe uma gama de aplicações dessa linguagem, como processamento de imagem, inteligência

artificial, *kernels* para sistemas operacionais, processamento em paralelo ou em tempo real etc., é importante limitar as variáveis selecionadas, ou grupo de variáveis, aos elementos primitivos para que uma área ou aplicação não gere resultados comparativos desproporcionais que podem gerar falsas conclusões.

Com os grupos de variáveis I, II, III e IV definidos, o próximo passo é contar a quantidade de elementos encontrados (recorrências) em cada elemento contido no grupo. Por último, estes dados contabilizados são processados conforme a metodologia do IIE contido no software Mensures (2016) para se alcançar o indicador ou valoração respectiva de cada código.

Para este trabalho, foram selecionados códigos da plataforma Arduino por dois motivos: 1) por serem códigos pequenos e de simples implementação que estudantes, professores ou profissionais da área podem realizar uma rápida leitura e dar seu *feeling* (palpite) sobre qual é a dificuldade do código e seu esforço de codificação; 2) Os códigos em Arduino são em sua maioria padronizados, tornando possível sua implementação nesta pesquisa, bem como sua replicação em testes e pesquisas futuras desenvolvidas por terceiros.

Pesquisas anteriores com outros códigos usados como *datasets* foram buscadas, mas, por exemplo, as bases de dados de Zhou e Leung (2006), bem como de Thirumalai, Reddy e Kishore (2017) não foram mais encontradas nos *links* apresentados nos artigos. Além disso, outros *datasets* encontrados eram específicos de outras linguagens de programação, como, por exemplo, Java e Python. Então, o Arduino se apresentou como uma boa relação custo versus benefício para a linguagem C, pois ao mesmo tempo que alunos poderiam contribuir na pesquisa, o software ficará em mais de um repositório para uso e comparações futuras.

A escolha pela plataforma Arduino tem suas limitações, pois os códigos são verdadeiramente pequenos quando comparados a códigos de programas para computadores pessoais. Porém, é importante citar que códigos muitos grandes, ou reais, são inviáveis de serem usados para uma pesquisa de opinião (enquete) junto aos desenvolvedores e profissionais da área, afinal, quantos programadores estariam dispostos a estudar códigos grandes que levariam entre duas e quatro horas para serem estudados, entendidos e mensurados por eles? Em resumo, qualquer metodologia de teste terá suas limitações.

O IIE pode ser calculado de duas formas, sendo elas o IIE0 e o IIE1. O primeiro caso, também chamado de teste primitivo, não usa a definição de prioridade entre as variáveis, e com isso as amplitudes naturais de cada variável, ou grupo de variáveis, são usadas plenamente. Este tipo de uso é comum quando não é possível definir prioridade entre as variáveis, como é o caso de circuitos eletrônicos, nos quais, devido a extensa interação entre os componentes, como capacitores, resistores, indutores, transistores etc., não existe *ranking* ou classificação de importância entre os componentes, e por consequência todos têm o mesmo peso, valor ou relevância.

Já no caso do IIE1, ele é usado quando existe algum tipo de *ranking*, posicionamento ou prioridade entre as variáveis selecionadas, como é o caso de alguns elementos no momento da codificação. Outro exemplo de aplicação do IIE1 é na mensuração de livros, onde a variável “editora”, ou pontuação da editora, faz muita diferença, uma vez que as melhores editoras possuem um processo de filtro intrínseco e com isso, estatisticamente falando, é comum os melhores livros serem publicados nas

"melhores" editoras. Da mesma forma, artigos de revistas científicas, ou similares, possuem algum *ranking* ou fator de impacto que deve ser respeitado.

3. Códigos Usados Para Teste

Os códigos usados para teste estão listados na tabela 1. Foram selecionados onze códigos básicos do Arduino retirados diretamente da IDE na versão 1.8.5. Estes códigos foram apresentados para 27 alunos do curso Técnico em Jogos Digitais de uma instituição de ensino técnico profissionalizante, para eles mensurarem mediante uma enquete de pesquisa o seu sentimento sobre a dificuldade, esforço ou complexidade de cada código. É importante salientar que estes alunos não possuem uma disciplina específica sobre Arduino, mas a disciplina de programação com uma carga 200 horas, na qual eles vêm algoritmos de programação e as linguagens C e C++, ou seja, eles se basearam em estruturas básicas e não em funções específicas do Arduino para mensurar os códigos apresentados. O mesmo teste foi feito com alunos do curso Técnico em Eletrônica e Técnico em Informática e os resultados foram semelhantes.

Para a mensuração, os alunos colocaram valores entre zero e dez, sendo os valores próximos de zero para os códigos pequenos e simples, e o valor dez (ou próximo) para os códigos mais difíceis, existindo também a possibilidade deles colocarem o carácter "X" quando não fosse entendido o código, e por consequência, impossibilitando a proposição de um valor de dificuldade.

Para o uso dos dados finais quatro alunos colocaram o valor "X" em quatro códigos ou mais, e por este motivo eles foram retirados integralmente da pesquisa, sendo considerados como pontos fora da curva (*outliers*). Para o cálculo final restaram vinte e três alunos e com suas pontuações foram calculadas as médias de cada código e também o desvio padrão, onde este é importante para entender a convergência ou divergências nas opiniões registradas.

Os códigos apresentados na tabela 1 não estão em ordem de dificuldade, esforço, prioridade ou termo similar. Para ter acesso a eles basta instalar a IDE do Arduino, que é gratuita, ir na pasta de exemplos e selecionar o código conforme o nome indicado na tabela 1.

Tabela 1. Tabela com as avaliações dos alunos

nº	*Local (pasta) - Nome*	*Média dos alunos*	*Desvio padrão*
1	Arduino Basics – Analog Read Serial	2,6	1,8
2	Arduino Basics – Bare Minimum	0,8	1,1
3	Arduino Basics - Blink	3,1	2,1
4	Arduino Basics – Digital Read Serial	4,0	2,2
5	Arduino Basics - Fade	3,9	2,5
6	Arduino Basics – Read Analog Voltage	4,0	2,4
7	Arduino Control - Array	5,3	1,5
8	Arduino Control – If Statement Conditional	4,5	2,3
9	Arduino Control – switch Case	5,0	2,3
10	Arduino Control – while Statement Condictional	6,8	2,1
11	Arduino Display – Bar Graph	6,0	2,1

Após a apresentação dos métodos clássicos de mensuração de códigos, dos dados coletados com os alunos, ou seja, do *feeling* dos educandos, e também da explicação sobre o IIE, segue-se para os resultados estatísticos quanto estes métodos são comparados.

4. Resultados Estatísticos

A tabela 2 apresenta todos os dados de mensuração dos onze códigos sob teste. A coluna com os dados da metodologia LoC (*Lines of Code*) foram encontrados após a formatação padrão AStyle contida na IDE Codeblocks. Já para as métricas da Complexidade Ciclomática (V(G)) foi usado o software CCCC (2021), e conforme apresentado nos tópicos anteriores a nota média dos alunos foi encontrada com uma pesquisa de campo. Já o IIE foi calculado conforme a metodologia contida no software Mensures (2016) usando as variáveis apresentadas no campo 2 deste artigo.

Ao se observar puramente os números da tabela 2, um erro de interpretação pode ocorrer, especialmente quando o foco está sobre o método de V(G) que possui vários zeros. Para diminuir possíveis erros humanos de comparação é necessário recorrer a uma comparação científica. O uso do teste de correlação linear, ou teste de Pearson, é um indicativo do quão próximos os dados estão ou se eles possuem uma raiz comum. Portanto, o teste de Pearson será usado para comprovar se a metodologia do IIE, que é genérica e robusta, pode também ser usado na mensuração de códigos em C e seus *subsets*.

Tabela 2. Dados comparativos entre os métodos de mensuração

nº	*Alunos*	*LoC*	*V(G)*	*IIE0*	*IIE1*
1	2,6	11	0	3,8	2,4
2	0,8	2	0	3,0	2,0
3	3,1	12	0	4,0	2,5
4	4,0	13	0	4,0	2,5
5	3,9	16	2	6,7	4,2
6	4,0	11	0	3,8	2,4
7	5,3	23	3	9,4	6,2
8	4,5	20	1	7,2	4,4
9	5,0	26	5	6,7	4,3
10	6,8	33	3	10,0	6,4
11	6,0	25	3	10,1	6,7

A tabela 3 apresenta uma matriz com o resultado do teste de correlação linear entre cada um dos cinco métodos indicados para mensuração dos onze códigos sob teste. Como o teste é de "todos contra todos", a parte inferior da tabela seria uma resposta espelhada da diagonal superior, e por este motivo os dados foram suprimidos, apresentação similar a apresentada no trabalho de Badri (2012) sobre correlação entre testabilidade de software e programação Orientada a Objetos (OO).

Para que se possa afirmar que existe correlação linear entre os itens é necessário levar em consideração o tamanho da amostra e o respectivo valor resultante do teste de Pearson (r). Segundo Triola (2013) para um grupo de 11 códigos, existe uma correlação forte sempre que o valor alcançado com o teste de Pearson for superior a 0,602. Já para uma correlação muito forte, ou seja, 99% de certeza, o valor deve ser igual ou superior a 0,735.

Tabela 3. Análise de correlação entre os métodos de mensuração

	LoC	*V(G)*	*Alunos*	*IIE0*	*IIE1*
LoC	1	0,827	0,95	0,901	0,886
V(G)		1	0,707	0,773	0,770
Alunos			1	0,875	0,862
IIE0				1	0,999
IIE1					1

Baseado nos dados da tabela 3, os únicos itens que não possuem uma correlação muito forte foram os "alunos" quando comparados a metodologia V(G), pois foi de apenas 0,707, fato compreendido uma vez que a Complexidade Ciclomática (V(G)) tem

algumas limitações quando comparado a todas as possibilidades contidas nos códigos, como, por exemplo, o uso de ponteiros, vetores (*arrays*) etc. Já a maior correlação encontrada entre todos os testes foi dos alunos versus LoC, que foi de 0,95, sendo desconsiderado o teste de IIE0 versus IIE1.

Entretanto, a metodologia do IIE teve uma correlação muito forte com todos os outros métodos, o que gera uma forte hipótese de que ele pode ser usado como um indicador genérico e robusto para mensurar em larga escala, os códigos em seu estado atual ou final.

O teste do IIE para os casos IIE0 e IIE1 é importante apenas para se testar o impacto na definição das prioridades entre as variáveis, mas que neste estudo de caso, como os códigos são pequenos e são apenas 4 grupos de variáveis, o IIE0 e IIE1 mostram uma correlação muito forte, ou seja, qualquer um destes métodos pode ser usado para mensurar os códigos. Entretanto, em outras atividades intelectuais mensuradas com o IIE, isso pode não ser verdadeiro e apenas um dos métodos deve ser usado, sendo recomendado, quando possível, o IIE1 devido à prioridade das variáveis.

A amplitude de aplicações do IIE em códigos em C parece ser grande, podendo ser aplicado em um simples exercício em sala de aula, em um trabalho bimestral, em projetos para as disciplinas, ou ainda, como um indicador administrativo resultante de um pedido de registro de software no Instituto Nacional de Propriedade Intelectual (INPI, 2021).

5. Conclusões e Considerações Finais

Neste estudo de caso, o IIE cumpriu sua função de mensurar de forma automatizada uma atividade intelectual próxima da avaliação humana, como no caso da comparação com a opinião dos alunos (*feeling* humano), bem como, em uma mensuração equivalente a metodologias tradicionais da área de programação, como a Complexidade Ciclomática e o número de linhas de código.

Conforme relatado anteriormente, o IIE foi desenvolvido para auxiliar nas avaliações automatizadas e em larga escada das atividades intelectuais explicitadas realizadas dentro do serviço público Brasileiro, mas que neste artigo se apresentou como um robusto indicador da complexidade ou do esforço de codificação em pequenos códigos de nível acadêmico.

Teste futuros com grandes códigos, como *Kernels*, devem ser realizados para fazer a mesma comparação estatística, e com isso, poder ser aferido que o IIE funciona em qualquer escala, sendo ela pequena, média ou grande. Além disso, novos testes com códigos em outras linguagens de programação, mas com um nível de aplicação acadêmico, são importantes para se buscar as limitações da aplicação do IIE.

Paralelamente a esta pesquisa com códigos de programação em C, o IIE está sendo testado em outras atividades intelectuais, como livros, artigos, atividades de extensão, patentes etc., e caso não seja encontrado um caso crítico que limite o seu uso, ele poderá ser uma quebra de paradigmas no processo tradicional de avaliação de desempenho em atividades intelectuais no serviço público, no ensino e em áreas marginais ou interdisciplinares.

Agradecimentos

Agradecemos aos alunos dos cursos técnicos do Instituto Federal do Paraná (IFPR), em especial aos alunos do curso de Jogos Digitais pela sua contribuição no processo de responder a enquete da pesquisa. Também ao colega Carlos Maffini pelas contribuições no artigo.

Referencias

Corrêa U. B. (2011), “Aplicação de métricas de software na predição de características físicas de software embarcado”. Dissertação. UFRGS. p. 93. Disponível em: < http://hdl.handle.net/10183/28733 >. Julho.

Zuse, H. (1999), “Thirty Years of Software Measurement”, in DUMKE, Reiner; ABRAN, Alain. (Eds.) Software Measurement: Current Trends in Research and Practice. Series: Information Engineering und IV-Controlling. Deutscher Universitätsverlag. 1999. ISBN: 978-3-8244-6876-8.

Pressman, R. S. (2011), “Engenharia de software”: uma abordagem profissional. Porto Alegre: AMGH, 2011. 780 p. ISBN 978-85-63308-33-7.

Flater, D. (2018), “Software Science revisited”: rationalizing Halstead’s system using dimensionless units. National Institute of Standerds and Tecnology (NIST – USA). 2018. Disponível em:< https://doi.org/10.6028/NIST.TN.1990 >. Junho de 2021.

CCCC, (2021), “C and C++ Code Counter”. Software. Disponível em: < https://sourceforge.net/projects/cccc/ >.

Oliveira A. A. (2016), “Mensures”. Software desenvolvido para mensurar atividades intelectuais, baseado na metodologia do Índice Interno de Esforço. 2016. BR 51 2016 001521-7. RPI 2405.

Cheng, B. H. C., Lemos, R., Inverardi, P. e Magee, J. (2009) (Eds.). “Software Engineering for Self-Adaptive Systems. Springer”. ISSN 0302-9743.

Zhou, Y. e Leung, H. (2006), "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," in IEEE Transactions on Software Engineering, vol. 32, no. 10, pp. 771-789, Oct. 2006, doi: 10.1109/TSE.2006.102.

Thirumalai, C. S., Reddy, P. A. e Kishore, Y. J. (2017), “Evaluating software metrics of gaming applications using code counter tool for C and C++ (CCCC)”, 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), 2017, pp. 180-184, doi: 10.1109/ICECA.2017.8212790.

Badri, M. e Toure, F. (2021), "Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes," Journal of Software Engineering and Applications, Vol. 5 No. 7, 2012, pp. 513-526. doi: 10.4236/jsea.2012.57060.

Triola, M. F. (2013), “Introdução à estatística”: atualização da tecnologia. Rio de Janeiro, LTC, xxviii, 707 p.