# Code complexity: an empirical study based on the index of internal effort

Alisson Antônio de Oliveira
alisson.oliveira@ifpr.edu.br. Instituto Federal do Paraná (IFPR), Curitiba, Paraná, Brasil

**Abstract**
Measuring the complexity of programming codes is a recurring challenge in projects and in computer science education. This study aims to verify the applicability of the Index of Internal Effort (IIE), a metric originally conceived to measure Explicit Intellectual Activities (EIA), in assessing the complexity of computational codes. The methodological approach consisted of analyzing two distinct datasets: (i) simple Arduino platform codes commonly used in electronics and embedded systems courses; and (ii) source codes from open-source projects published in public repositories such as GitHub. The study applied the IIE, in its complexity approximation form (IIEa), adapted for source code analysis, and compared its results with well-established software metrics such as LOC (Lines of Code), WMC (Weighted Methods per Class), and CC (Cyclomatic Complexity). Results showed a significant correlation between the IIE and traditional metrics, providing preliminary evidence of the IIE's applicability. Due to the lack of data on the number of developers in open-source projects, it was not possible to calculate the average team effort using IIE0, the second equation of the IIE framework. The comparison between IIE and previous studies revealed that complexity is strongly associated with the structural variables of code. It is concluded that the IIE is a Systems Engineering–based alternative that can be used to measure EIA, such as programming code complexity.
**Keywords:** complexity; computation; empirical study; software metrics; systems engineering.

## *Complexidade de códigos: um estudo empírico usando o índice interno de esforço*

***Resumo***
*A mensuração da complexidade de códigos de programação é um desafio recorrente em projetos e no ensino de computação. Este estudo tem como objetivo verificar a aplicabilidade do Índice Interno de Esforço (IIE), uma métrica originalmente concebida para mensurar Atividades Intelectuais Explicitadas (AIE), na mensuração da complexidade de códigos computacionais. A abordagem metodológica consistiu na análise de dois conjuntos distintos de dados: (i) códigos simples da plataforma Arduino comuns nos cursos de eletrônica, sistemas embarcados e afins; (ii) códigos-fonte de projetos open-source publicados em repositórios públicos, como GitHub. O estudo aplicou o IIE, em sua aproximação da complexidade (IIEa), adaptado para análise de códigos-fonte, e comparou seus resultados com métricas consagradas da área de software, como LOC (Lines of Code), WMC (Weighted Methods per Class) e CC (Cyclomatic Complexity). Os resultados mostraram correlação significativa entre o IIE e as métricas tradicionais, fornecendo evidências preliminares da aplicabilidade do IIE. Devido à falta de dados sobre a quantidade de desenvolvedores nos projetos open-source não foi possível calcular o esforço médio dos membros da equipe, usando o IIE0, uma segunda equação do framework do IIE. A comparação do IIE com estudos anteriores evidenciou que a complexidade está fortemente associada a variáveis estruturais dos códigos. Conclui-se que o IIE é uma alternativa da área de engenharia de sistemas que pode ser usada na mensuração de AIE, como, por exemplo, na complexidade de códigos de programação.*
***Palavras-chave:*** *complexidade; computação; engenharia de sistemas; estudo empírico; métricas de código.*



## 1 Introduction

Measuring code complexity is a critical aspect of software engineering, directly linked to software quality assurance, maintainability, and overall system performance (Pressman, 2011). In educational settings, providing students with meaningful indicators of code complexity can more effectively guide them through the progressive stages of learning programming (Oliveira; Pilatti, 2021).

The inherent complexity of a program significantly influences development time, the ease of identifying and fixing errors, how well other programmers understand the code, and ultimately, the total cost of the project (Sommerville, 2019). Ignoring complexity metrics may lead to systems that are difficult to maintain, more error-prone, and costly to modify, compromising productivity and reliability in professional environments (IEEE, 1998). In academic settings, projects that require high technical sophistication or

involve numerous design constraints may discourage less experienced students or those inadequately prepared.

Over the past decades, numerous metrics have been proposed to quantify various attributes of source code. In systematic reviews conducted by Ardito *et al*. (2020) and Vogel *et al*. (2020), more than 100 metrics related to coding and software engineering were cataloged. While each metric has specific use cases, some of the most widely used are generic indicators of effort or complexity, such as Lines of Code (LoC) and Cyclomatic Complexity (CC). Other well-known and more specific metrics include: Coupling Between Objects (CBO), Response For a Class (RFC), Depth of Inheritance Tree (DIT), Lack of Cohesion in Methods (LCOM), and Number of Children (NOC).

One of the most influential contributions in this area is the Weighted Methods per Class (WMC) metric, proposed by Chidamber and Kemerer (1994). This metric evaluates a class's complexity based on the sum of the complexities of its methods, using Cyclomatic Complexity (CC) or a similar measure as its base. According to Pressman (2011), classes with high WMC values tend to be highly specialized, which can hinder their reuse in other projects. Due to its conceptual relevance to complexity, WMC is the only object-oriented (O.O.) metric considered in this study.

Meirelles (2013) emphasizes the need to carefully select appropriate metrics and define priorities for their application. A common structural limitation is that traditional metrics are often tied to the paradigms of the programming languages used (Tiwari, 2017).

In the context of software engineering education, analyzing the cognitive load involved in reading and understanding code is especially relevant. According to Gonçales, Farias and Silva (2021), cognitive load refers to the mental effort required from developers, ranging from novice students to experienced professionals, and it may vary depending on the nature of the tasks performed. Cognitive load measurement has been applied for various purposes, such as improving productivity and software quality. Understanding its dimensions can help identify cognitive bottlenecks, optimize task distribution across teams, and contribute to the development of more readable, secure, and efficient systems.

This study is motivated by a gap identified in the literature: the lack of a generic and robust metric capable of broadly assessing both code complexity and the average effort required by development teams. Although traditional metrics are useful, they fall short in capturing the full spectrum of complexity involved in software creation and maintenance. In this context, we propose using a metric developed in the field of Systems Engineering, the Index of Internal Effort (IIE), as a broader alternative for quantifying intellectual activities expressed through source code.

Systems Engineering provides a valuable methodological foundation for this approach. Sheard and Mostashari (2010) proposed that the complexity of products and processes can be categorized into hierarchical structures, referred to as the Complexity Typology (CT), which includes six levels:

- Order-1 Structures - Related to quantity and size: number of components, processes, instances, etc.;
- Order-2 Structures - Related to connections and communication: number and type of links, interaction intensity, etc.;
- Order-3 Structures - Associated with architecture: patterns, heterogeneity, exceptions, branching, etc.;
- Order-4 Structures - Linked to fast dynamic behaviors: nonlinearity, feedback, stability, etc.;
- Order-5 Structures - Concerning long-term dynamic behaviors: self-organization, adaptation, evolution, etc.;
- Order-6 Structures - Related to social and political aspects: number of stakeholders, economic impacts, conflicting interests, and more.

Although this typology is generic, domain experts must conduct contextualized studies to identify, measure, and rank these structures. However, Sheard and Mostashari (2010) did not provide a method for integrating them into a single quantitative indicator.

To derive a quantitative complexity indicator inspired by Systems Engineering principles, this study employs the Index of Internal Effort (IIE) framework, previously applied to the measurement of industrial property patents by Oliveira *et al*. (2023a, 2023b), Oliveira, Santos and Pilatti (2024), and Pereira, Oliveira and Barbalho (2025). The IIE framework has already been tested in programming contexts under the following conditions: (i) small Arduino scripts evaluated by Digital Games students (Oliveira; Pilatti, 2021), and (ii) code used in the Brazilian Informatics Olympiad (*Olimpíada Brasileira de Informática – OBI*), where IIE results were compared to official problem difficulty levels (Oliveira, 2024a). However, the IIE has

not yet been applied to large open-source codebases, which represents, hypothetically, a research gap regarding this Systems Engineering-based metric.

The present study aims to compare the results of the IIE framework with well-established metrics commonly used in code evaluation. Since the IIE was originally designed as a generic metric for any Explicit Intellectual Activity (EIA) (Oliveira, 2024b), it is essential to test its boundaries and validate its applicability in software development contexts not yet explored in the scientific literature.

Inspired by the concept of Robust Control, in which systems can be effectively managed despite uncertainties or parameter variations (Dorf; Bishop, 2011), the IIE employs a methodology that tolerates local imprecision without compromising the global evaluation.

The specific objectives of this research are:

- To apply the IIE to codebases under extreme conditions, ranging from simple scripts to complex open-source projects;
- To compare IIE results with Cyclomatic Complexity, Lines of Code (LoC), and human evaluations using datasets composed of Arduino and open-source C/C++ projects;
- To statistically analyze the correlation between IIE, traditional metrics, and human evaluations using appropriate correlation tests;
- To assess the robustness and generalizability of the IIE across different development contexts in order to verify its large-scale utility in measuring intellectual activities represented in code.

This section has introduced the topic and theoretical background necessary to contextualize the study. The remaining sections are organized as follows: Section 2 presents the research methodology, including IIE modeling and case studies; Section 3 details the results and statistical correlation analyses; Section 4 provides a qualitative discussion and comparisons with existing literature; and finally, Section 5 contains the conclusions and recommendations for future work.

## 2 Methodology

To collect classical software metrics, we used the CCCC tool (2013), selected due to its direct integration with the Code::Blocks IDE. This tool was employed to measure three widely recognized metrics in C language code: Lines of Code (LoC), Cyclomatic Complexity (CC), and Weighted Methods per Class (WMC).

The Index of Internal Effort (IIE), being a generic framework designed to quantify intellectual activities, requires a more detailed procedure. As described by Oliveira, Santos and Pilatti (2024), applying the IIE to estimate either the approximate complexity (IIEa) or the average work effort (IIE0) of an Explicit Intellectual Activity (EIA) involves the following steps:

i) Identify the variables involved in the intellectual activity, distinguishing between internal variables (contained within the EIA) and external variables (such as execution context, resources used, outcomes achieved, etc.);

ii) Organize the potential variables according to the Complexity Typology (CT) proposed by Sheard and Mostashari (2010), using correlation tests to determine which variables are most relevant for measurement;

iii) Ensure robustness in estimation by using at least four variables (or groups of variables), minimizing the impact of extreme values or subjective bias. Whenever possible, include at least one quantitative and one qualitative variable;

iv) Group the variables into two sets: output variables (An) and input variables (Bm), each containing at least one variable. This grouping is essential for applying Equations 1 and 2;

v) Apply the IIEa equation (Equation 1) to the defined groups to compute an approximate measure of complexity. A square root correction factor is used to offset distortions caused by the variable ranges. The result is a dimensionless value, which facilitates summation and comparison among the An components;

vi) If the statistical analysis does not reveal a significant positive correlation between IIEa and other indicators, the process must be restarted from step (i). A practical stopping criterion is achieving an approximately normal distribution in the results, which typically depends on the diversity of the codes analyzed;

vii) Once a significant correlation for IIEa is confirmed, the primitive effort function (IIE0), represented by Equation 2, can be applied to estimate the average individual effort of developers, provided that data regarding task inputs are available.

Equation 1 provides an approximation of task (or process) complexity based on the CT. Equation 2 is used to calculate the average individual effort by incorporating not only observable outputs but also the task's inputs. The IIE0 is referred to as the primitive function because it represents the basic intellectual effort measurement and distinguishes itself from the IIEa complexity approximation.

$$IIEa = 1 + \sqrt[2]{A_1} + \sqrt[2]{A_2} + \sqrt[2]{A_3} + \sqrt[2]{A_4} + \ldots + \sqrt[2]{A_n} \quad (1)$$

$$IIE0 = \left( \frac{1+\sqrt[2]{A_1}+\sqrt[2]{A_2}+\sqrt[2]{A_3}+\sqrt[2]{A_4}+\ldots+\sqrt[2]{A_n}}{1+\sqrt[2]{B_1}+\sqrt[2]{B_2}+\ldots+\sqrt[2]{B_m}} \right) \quad (2)$$

To compare the performance of IIE with classical metrics applied to C/C++ code, two case studies were conducted. These studies were intentionally designed to differ in code size and complexity: (I) Case Study I: composed of short code samples similar to those used in introductory programming courses or small-scale projects; (II) Case Study II: based on larger code samples from open-source projects, requiring greater development effort.

Inspired by the empirical study conducted by Oliveira and Pilatti (2021), Case Study I involved students from a high school technical course in Electronics. Students were asked to subjectively rate the difficulty of Arduino codes they reviewed. Each participant gave a score from 0 to 10, proportional to their perceived difficulty. Two groups of students participated: (i) Group A2019: students with only one year of experience programming in standard ANSI C; (ii) Group A2018: more advanced students, already applying programming concepts in microcontrollers from the PIC family (Programmable Interface Controller), developed by Microchip.

Although Case Study I involved only 11 code samples, this small number was intentional. Evaluations were carried out during regular class hours, and the code set was kept manageable for the available time. According to Álvarez, Mozo and Durán (2021), the Arduino platform is considered an international standard due to its widespread use in educational and commercial contexts, which reinforces the study's external validity and facilitates future comparisons.

For Case Study II, source code was selected from open-source projects available on platforms such as GitHub (2025) and SourceForge (2025). Table 1 provides an overview of the codes used in both case studies. In the case of large open-source projects (Case Study II), which contain multiple files and functions, a single representative file was selected from each project. These selections are detailed later in Table 3.

To allow direct and consistent comparisons between the two studies, Case Study II also included 11 code samples, thus matching the sample size of Case Study I. This design choice enables the use of the same critical values for correlation coefficients (r), eliminating the need for auxiliary statistical tables during significance testing.

Table 1 – C language codes used in Case Study I and II

| Nº | Case study I: Arduino codes | Case study II: open-source codes |
|---|---|---|
| 1 | Analog Read Serial | Notepad++ |
| 2 | Bare Minimum | VLC media player |
| 3 | Blink | eMule |
| 4 | Digital Read Serial | 7-ZIP |
| 5 | Fade | Apache OpenOffice |
| 6 | Read Analog Voltage | OBS-Studio |
| 7 | Array | Google Level DB |
| 8 | If Statement – Conditional | Retro Arch |
| 9 | Switch Case | GIMP |
| 10 | While Statement – Conditional | Free CAD |
| 11 | BarGraph | DOOM Open Source Release |

Source: research data

To minimize the influence of subjective judgment on results, statistical correlation testing was employed to identify the presence of direct or indirect relationships among variables. The choice between Pearson's test (parametric) and Spearman's test (non-parametric) depends on the nature of the dataset: when data are continuous and normally distributed (Gaussian), Pearson's test is recommended; in cases involving ordinal data, non-normal distributions, or outliers, Spearman's test is more appropriate.

## 2.1 Application of the methodology to the dataset

As in the case study described by Oliveira (2024a), the data used to measure code complexity based on the Index of Internal Effort (IIE) framework were categorized according to the Complexity Typology (CT) as follows:

A4) Order-4 Structures of CT: iteration and control flow structures, such as while, do while, for, goto, and continue;

A3) Order-3 Structures of CT: decision-making and conditional branching structures, including if-else, switch-case, and the use of break;

A2) Order-2 Structures of CT: logical and relational operators, such as &&, ||, !, <, <=, ==, >=, >, and !=;

A1) Order-1 Structures of CT: organizational language elements, such as curly braces {}, square brackets [], and parentheses ().

The order adopted for analyzing these structures follows the didactic sequence commonly used in teaching the C language, as can be observed in both contemporary works (e.g., Backes, 2022) and classic references such as Oualline (1997) and Kernighan and Ritchie (1988).

Based on these C language structures, Table 2 presents the results of applying the IIEa framework to the Case Study I dataset. Subjective evaluations of code comprehension difficulty were conducted by students from the Technical Course in Electronics in the years 2018 and 2019. The variation in assigned scores can be explained by the experience level of each group: students from 2018 (Group A2018) were in their third and final year of the program, having already applied their programming knowledge in practical courses involving microcontrollers. In contrast, students from 2019 (Group A2019) had only completed an introductory course in algorithms and programming, with some students even retaking the course due to prior academic failure.

Table 2 – Code metric results for Case Study I

| Nº | LoC | CC | A2018 | A2019 | IIEa |
|---|---|---|---|---|---|
| 1 | 11 | 0 | 2,48 | 1,45 | 3,83 |
| 2 | 2 | 0 | 0,71 | 0,27 | 3,00 |
| 3 | 12 | 0 | 2,10 | 1,5 | 4,0 |
| 4 | 13 | 0 | 3,71 | 2,15 | 4,0 |
| 5 | 16 | 2 | 4,76 | 3,05 | 6,73 |
| 6 | 11 | 0 | 3,57 | 2,37 | 3,83 |
| 7 | 23 | 3 | 5,86 | 3,90 | 9,36 |
| 8 | 20 | 1 | 4,80 | 2,39 | 7,16 |
| 9 | 26 | 5 | 5,15 | 3,33 | 6,74 |
| 10 | 33 | 3 | 7,40 | 5,13 | 10,05 |
| 11 | 25 | 3 | 6,14 | 4,10 | 10,14 |

Source: research data

In Case Study II, the results of the collected metrics are shown in Table 3. Unlike the previous study, this stage did not include student evaluations; however, the Weighted Methods per Class (WMC) metric was added. The "Software" column contains three pieces of information: the name of the application, the source file analyzed, and its version. The "Size" column indicates the total number of lines of code in each file, including blank lines and comments. These details may be useful for comparisons with future studies.

Table 3 – Metrics of the selected projects in Case Study II

| Nº | Software | Size * | LoC | CC | WMC | IIEa |
|---|---|---|---|---|---|---|
| 1 | Notepad++ (winmain.cpp) (versão 8.2) | 708 | 289 | 85 | 15 | 49,3 |
| 2 | VLC media player (player_controller.cpp) (versão 3.0.16) | 1955 | 773 | 100 | 53 | 82,1 |
| 3 | eMule (ircMain.cpp) (versão 0.50a) | 1650 | 812 | 175 | 14 | 86,9 |
| 4 | 7-ZIP (main.cpp) (versão 21.07) | 1482 | 973 | 197 | 21 | 75,9 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | Apache OpenOffice (DomainMapper_Impl.cxx) (versão 4.1.11) | 3898 | 1000 | 157 | 47 | 97,4 |
| 6 | OBS-Studio (Main.cpp) (versão 0.659b) | 852 | 282 | 60 | 12 | 57,2 |
| 7 | Google LevelDB (versão 1.23) (version_set.cc) | 1569 | 131 | 34 | 19 | 82,0 |
| 8 | RetroArch (menu_setting.c) (versão 1.9.14) | 21188 | 5373 | 1177 | 205 | 195,4 |
| 9 | GIMP (main.c) (versão 2.99) | 1071 | 375 | 64 | 15 | 51,4 |
| 10 | FreeCAD (MainGui.cpp) (versão 0.19.3) | 448 | 209 | 15 | 10 | 38,9 |
| 11 | DOOM Open Source Release (r_main.c) (versão 1.10) | 1171 | 536 | 93 | 17 | 58,6 |

Source: research data

Due to the unavailability of detailed information about the projects or their development processes, it was not possible to calculate the equation used to estimate the developers' average work effort (referred to as IIE0). This calculation typically requires data such as development time, number of developers involved, and total financial investment, variables commonly used in productivity and cost assessments in software engineering. By analogy, in the context of patent development, the study conducted by Pereira, Oliveira and Barbalho (2025) was able to perform this type of analysis thanks to the availability of such data.

The data collected in both case studies (Tables 2 and 3) were first subjected to the Kolmogorov-Smirnov (KS) non-parametric test to verify the normality of the distributions. Upon statistical confirmation of normality, the Pearson correlation test was applied, and the results are consolidated in Tables 4 and 5.

According to the statistical parameters described by Triola (2013), when correlation analysis is conducted with a sample size of 11, the correlation values (r) must fall within the following ranges to be considered statistically significant: (i) For a 99% confidence level (alpha = 0.01): $-0.735 < r > 0.735$; (ii) For a 95% confidence level (alpha = 0.05): $-0.602 < r > 0.602$.

Correlation tests between variables were carried out using an all-against-all cross-analysis format, following the approach adopted by Badri and Toure (2012) in their study on correlations between software testability and object-oriented (OO) programming practices. Highlighted values in Tables 4 and 5 indicate statistically significant correlations at the 0.05 significance level ($r > 0.602$).

## 3 Results

Tables 4 and 5 present the results of the correlation tests applied to Case Studies I and II, respectively. The highlighted values in both tables indicate statistically significant positive correlations among the analyzed variables.

In Case Study I, which involved short code samples and students at different stages of technical education, a strong mutual correlation was observed between the metrics LoC (Lines of Code), CC (Cyclomatic Complexity), A2018 (student evaluation from the final year), and IIEa. This finding suggests that the more experienced students were better able to accurately assess code complexity, as their judgments aligned with the objective metrics. In contrast, evaluations from group A2019, comprising students with limited training and experience in programming, showed no significant correlation with any of the other metrics. This absence of correlation reinforces the hypothesis that technical knowledge plays a critical role in a student's ability to judge code complexity, a point that will be further discussed in the next section.

Table 4 – Correlation matrix among metrics in Case Study I

| | **LoC** | **CC** | **A2018** | **A2019** | **IIEa** |
|---|---|---|---|---|---|
| LoC | 1 | 0,827 | 0,951 | 0,385 | 0,901 |
| CC | | 1 | 0,760 | 0,106 | 0,773 |
| A2018 | | | 1 | 0,472 | 0,924 |

| | | | |
|---|---|---|---|
| A2019 | | 1 | 0,217 |
| IIEa | | | 1 |

Source: research data

In Case Study II, which analyzed large-scale source code from real software applications, subjective evaluations were intentionally excluded due to the impracticality of requesting human assessments for large codebases. Instead, established computational metrics such as WMC, along with LoC, CC, and IIEa, were used. The data in Table 5 indicate that all metrics showed significant positive correlations in this dataset, demonstrating strong consistency between IIEa and well-established software engineering indicators. This supports the hypothesis that IIEa is capable of capturing relevant aspects of software complexity, even at scale, without contradicting traditional metrics.

Table 5 – Correlation matrix among metrics in Case Study II

| | **Size** | **LoC** | **CC** | **WMC** | **IIEa** |
|---|---|---|---|---|---|
| Size | 1 | 0,989 | 0,988 | 0,987 | 0,948 |
| LoC | | 1 | 0,996 | 0,979 | 0,945 |
| CC | | | 1 | 0,968 | 0,937 |
| WMC | | | | 1 | 0,945 |
| IIEa | | | | | 1 |

Source: research data

The central focus of this study is the comparison between the Index of Internal Effort (IIE) framework, developed based on systems engineering principles, and traditional software complexity metrics. The primary goal is to test the consistency of IIEa in both academic code and production-level software projects. The results from both case studies do not refute the applicability of IIEa as a reliable complexity assessment tool and show coherence with established metrics such as LoC, CC, and WMC.

As a generic framework designed to robustly measure Explicit Intellectual Activities (EIA), IIE requires validation in specific domains, such as software development, to identify potential limitations or operational boundaries.

Due to compilation issues in the Code::Blocks environment with the code samples from Case Study II, resulting from the partial use of source files, the initial metrics extracted using the CCCC software (2013) were cross-validated against data generated by RSM software (M Squared Technologies, 2001). Although the absolute values differed, the correlation between the two tools for LoC and CC exceeded 0.96, thus supporting the use of CCCC as the primary tool in this analysis. The RSM tool also provides a Function Points (FP) metric; however, because it showed a strong correlation with LoC, the FP values were deemed redundant and excluded from this study.

## 4 Discussion

The tests carried out in Case Study I were intentionally simple and selected for two main reasons: (i) to enable a comparison between the results of the IIE framework and human evaluations conducted by students with introductory knowledge in programming, and (ii) to make it easier for novice programmers to reproduce the codes, compile them, and run their own comparative tests. The Arduino platform was chosen due to its widespread use, significant variety of code examples, and, to some extent, its status as an international standard in teaching and prototyping (Álvarez; Mozo; Durán, 2021).

At the opposite end, Case Study II involved larger and more complex codebases, contrasting with the simplicity of Case Study I. Moreover, this second study allowed the introduction of the WMC metric, enriching the analysis.

In both studies, a significant correlation was observed between IIEa and the established metrics in software engineering (see Tables 4 and 5). This is a remarkable result, as IIE is a generic method aimed at measuring Explicit Intellectual Activities (EIA), whereas the other metrics applied are specifically tailored to the software development context.

The consistency of the results obtained using IIEa reinforces its ability to measure the cognitive effort involved in coding, aligning well with traditional metrics in the field. Additionally, the IIE has unique features, such as its independence from code compilation and specific project configurations or paradigms. This allows the metric to detect even small code modifications.

According to Ardito *et al*. (2020), the number of lines changed in a class is a maintainability metric that can be used to compare different versions of the same software. In this sense, IIE can complement such metrics by automatically or semi-automatically indicating the effort involved in the changes made.

The IIE framework aligns with the quality criteria for software metrics proposed by Mills (1988), which include: (i) simplicity and clarity in the definition of what is being measured; (ii) objectivity in data collection; (iii) ease of obtaining values; (iv) validity of the metric in measuring what it claims to measure, with attention to approximations and subjectivity; and (v) robustness in guiding improvements in processes or products.

Additionally, IIE conforms to the complexity typology from Systems Engineering, as proposed by Sheard and Mostashari (2010), underscoring its versatility and potential for application in various complex domains.

Two central aspects of the IIE methodology are worth highlighting. The first is the establishment of a hierarchy of importance among the variables used in measurement, which aligns with the six levels of the complexity typology. The second is the use of a compression (or dampening) process applied to the data range via the square root function (Equation 1), which minimizes the impact of potential distortions during data collection.

To mitigate measurement bias, the IIE relies on two important premises. First, it incorporates at least four variables (or groups of variables), distributing effort across different dimensions. This approach significantly differs from McCabe's (1976) Cyclomatic Complexity (CC) metric. Second, the concept of effort (IIE0), incorporated in Equation 2, enables consideration of each team member's relative contribution. This is particularly useful for analyzing productivity in teams of varying sizes.

In the context of programming education, a noticeable discrepancy was observed between the evaluations from groups A2018 and A2019. Students in A2019, who had taken only an introductory course on algorithms, struggled to assess the complexity of the proposed codes. Their evaluations diverged considerably from technical metrics and from those of more experienced students. This evidence reinforces the importance of including, even at a basic level, content related to code measurement in the curriculum. Doing so can help reduce the perception of mismatch between exam evaluations and classroom exercises, a common complaint among students.

One limitation of the present study is the lack of information regarding the number of developers per code (variable B1), which prevented the calculation of IIE0 in the case studies. Similarly, no data were collected on development time or financial resources invested, which could have enriched the analysis. In other applications of IIE, such as scientific articles, books, or patents, the identification of authors is explicit, allowing for a clearer distinction between individual effort (IIE0) and general complexity (IIEa).

It is important to note that, in the field of software engineering, some metrics treat complexity as a negative attribute, associated with software maintainability issues (Pressman, 2011). However, this perspective was not applicable to the IIE framework in Case Studies I and II.

In line with Popper's (2001) principle of scientific falsifiability, it is essential that IIE be tested across different programming languages, paradigms, and databases to uncover potential limitations. If such limitations are not found, IIE may be considered a robust metric and potentially adopted as a design or procurement criterion in the public sector, similar to the Function Points metric (Brasil, 2024).

A relevant future research agenda includes comparing IIEa with metrics based on biological measurements of cognitive effort (biometrics), such as those proposed by Lee *et al*. (2017), among others.

In summary, the IIE emerges as a viable alternative for measuring EIA, such as developing C/C++ code. Its distinguishing feature lies in Equation 2, which enables evaluation of the average effort per team member, making it suitable for use in research projects, final course assignments, or other collaborative academic contexts.



## 5 Conclusions

This study aimed to evaluate the applicability of the IIE for measuring the complexity of programming code in different contexts. The results showed that the estimated complexity (IIEa) exhibited a significant correlation with well-established software engineering metrics such as WMC, LoC, and CC, both in simple student-produced code and in more complex open-source projects. Additionally, it was observed that students with limited programming experience (A2019) provided assessments that diverged significantly from objective metrics.

These findings clearly address the central research question: whether the IIE can be used to measure the complexity involved in software code production. The strong correlation between IIEa and traditional

metrics suggests that the method, though generic and originally developed within the field of Systems Engineering, is capable of capturing the complexity inherent in software development tasks.

Among the most significant contributions of the study is the application of an interdisciplinary metric, originating from Systems Engineering, to the field of programming. The incorporation of IIEa provides a new perspective on assessing cognitive effort in EIA, allowing instructors to more accurately evaluate the complexity of student-developed code. A particularly noteworthy outcome is the ability to measure even small changes in non-compilable code, expanding the usability of the IIE to contexts where other metrics fall short.

From a practical standpoint, the results can be applied to project assessment in programming courses, especially when developed in teams, enabling the estimation of the average individual effort per person based on Equation 2.

One limitation of the present study was the lack of data on the number of developers per codebase, which hindered full application of the IIE0 metric. Furthermore, variables such as development time and financial resources were not measurable, limiting the analysis in terms of total effort. Future research should seek more comprehensive datasets and test the IIE in a wider variety of programming languages, paradigms, and educational contexts.

For the scientific community, this work expands the possibilities for measuring software complexity by proposing a framework that can be adapted to various scenarios and has proven useful even in educational environments. It is hoped that these results will inspire further studies, methodological refinements, and practical applications.

In summary, the IIE has demonstrated to be a robust and versatile alternative for measuring code complexity of programming code. Its key innovation lies in its ability to estimate the average effort per developer in collaborative projects, as well as to assess incremental changes in code, making it particularly valuable in teamwork settings. These characteristics reinforce the importance of adopting broader and more interdisciplinary metrics in programming.

**Acknowledgments**

I would like to express my gratitude to everyone who contributed directly or indirectly to this research. In particular, I extend my sincere thanks to my colleagues Fábio Albini, Eduardo Tondin, Sheila Freitas, Charles Fung, and Cleverton Vicentini for their valuable feedback during the preliminary review of this article.

**Funding**

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

**Conflict of interest**

The author declares no conflict of interest.

**References**

ÁLVAREZ, J. L.; MOZO, J. D.; DURÁN, E. Analysis of single board architectures integrating sensors technologies. **Sensors**, v. 21, n. 18, 6303, 2021. DOI: http://doi.org/10.3390/s21186303.

ARDITO, L.; COPPOLA, R.; BARBATO, L.; VERGA, D. A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review. **Scientific Programming**, v. 2020, n. 1,8840389, 2020. DOI: https://doi.org/10.1155/2020/8840389.

BACKES, A. **Linguagem C - Completa e Descomplicada**. 2. ed. Rio de Janeiro: GEN LTC, 2022. 424 p. In Portuguese.

BADRI, M.; TOURE. F. Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes. **Journal of Software Engineering and Applications**, v. 5, n. 7, p. 513-526, 2012. DOI: http://doi.org/10.4236/jsea.2012.57060.

BRASIL. Ministério da Gestão e da Inovação em Serviços Públicos. **PE 08/2023** – Ponto de Função – CODIFICAGOV. Brasília, DF: MGI, 2024. Available at:

https://www.gov.br/gestao/pt-br/assuntos/central-de-compras/transparencia/arp/2024/pe-08-2023-ponto-de-funcao-codificagov. Accessed on: 1 Jan. 2025. In Portuguese.

CCCC. **C and C++ Code Counter**. Software. 2013. Available at: https://sourceforge.net/projects/cccc/. Accessed on: 1 Jan. 2023.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 476-493, 1994. DOI: https://doi.org/10.1109/32.295895.

DORF, R. C.; BISHOP, R. H. **Modern control systems**. 12th ed. New Jersey: Prentice Hall. 2011.

GITHUB. **Trending**. 2025. Available at: https://github.com/trending. Accessed on: 1 Feb. 2025.

GONÇALES, L. J.; FARIAS, K.; SILVA, B. C. Measuring the cognitive load of software developers: an extended systematic mapping study. **Information and Software Technology**, v. 136, 106563, 2021. DOI: https://doi.org/10.1016/j.infsof.2021.106563.

IEEE – INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **1061-1998 - IEEE Standard for a Software Quality Metrics Methodology**. [*S.l.*]: IEEE, 1998. DOI: https://doi.org/10.1109/IEEESTD.1998.243394.

KERNIGHAN, B. W.; RITCHIE, D. M. **The ANSI C Programming Language**. 2nd ed. New Jersey: Prentice Hall, 1988.

LEE, S.; HOOSHYAR, D.; JI, H.; NAM, K.; LIM, H. Mining biometric data to predict programmer expertise and task difficulty. **Cluster Computing**, v. 21, n. 1, p. 1097-1107, 2017. DOI: https://doi.org/10.1007/s10586-017-0746-2.

M SQUARED TECHNOLOGIES. **What's Inside Your Source Code?** 2001. Available at: https://www.ic.unicamp.br/~eliane/Cursos/RSM/rsm_descricao.htm. Accessed on: 5 Jan. 2025.

MCCABE, T. J. A Complexity Measure. **IEEE Transactions on Software Engineering**, v. SE-2, n. 4, p. 308-320, 1976. DOI: https://doi.org/10.1109/TSE.1976.233837.

MEIRELLES, P. R. M. **Monitoramento de métricas de código-fonte em projetos de software livre**. Tese (Doutorado em Ciência da Computação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013. DOI: https://doi.org/10.11606/T.45.2013.tde-27082013-090242. In Portuguese.

MILLS, E. E. **Software Metrics**. Seattle: Software Engineering Institute, Carnegie Mellon University, 1988. 43 p. Available at: https://resources.sei.cmu.edu/asset_files/CurriculumModule/1988_007_001_15608.pdf. Accessed on: 10 Jan. 2023.

OLIVEIRA, A. A. Assessing Programming Difficulty and Effort: Statistical Correlations with the Index of Internal Effort. *In*: ESCOLA REGIONAL DE INFORMÁTICA DE GOIÁS (ERI-GO), 12., 2024, Ceres. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2024a. p. 21-30. DOI: https://doi.org/10.5753/erigo.2024.4795.

OLIVEIRA, A. A. **Índice interno de esforço**: uma proposta para a mensuração robusta de artefatos intelectuais desenvolvidos por servidores públicos. 103 f. 2024. Tese (Doutorado em Engenharia de Produção) – Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2024b. Available at: http://repositorio.utfpr.edu.br/jspui/handle/1/35490. Accessed on: 5 Jan. 2025. In Portuguese.

OLIVEIRA, A. A.; FUNG, C. W. H.; BURKARKER, E.; PILATTI, L. A.; SANTOS, C. B. Metrificação de patentes: uma análise entre qualidade, complexidade e esforço. *In*: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 43., 2023, Fortaleza. **Anais** [...]. Fortaleza: ABEPRO, 2023a. DOI: http://doi.org/10.14488/enegep2023_tn_st_404_1989_45333. In Portuguese.

OLIVEIRA, A. A.; GUIMARÃES, T. A.; AVILA, C. A.; PILATTI, L. A.; SANTOS, C. B. Metrificação de patentes desenvolvidas no serviço público: um estudo acerca do índice interno de esforço. *In*: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 43., 2023, Fortaleza. **Anais** [...]. Fortaleza: ABEPRO, 2023b. DOI: http://doi.org/10.14488/enegep2023_tn_wpg_404_1986_45338. In Portuguese.

OLIVEIRA, A. A.; PILATTI, L. A. Mensuração da complexidade de códigos em C com o método do Índice Interno de Esforço. **Anais do Encontro Anual de Tecnologia da Informação**, v. 10, n. 2, 2021. Available at: http://anais.eati.info/eati/article/view/64. Accessed on: 10 Sept. 2023. In Portuguese.

OLIVEIRA, A. A.; SANTOS, C. B.; PILATTI, L. A. Bridging the gap in patent assessment: The Index of Internal Effort framework for pharma innovations. **Journal of Pharmacy & Pharmacognosy Research**, v. 12, n. 5, p. 852-869, 2024. DOI: https://doi.org/10.56499/jppres23.1859_12.5.852.

OUALLINE, S. **Practical C programming**: why does 2+2=5986. 3rd ed. Sebastopol: O'Reilly, 1997. 504 p.

PEREIRA, S. A.; OLIVEIRA, A. A.; BARBALHO, C. R. S. Gestão estratégica de patentes em instituições públicas: avaliando o Índice Interno de Esforço como ferramenta para otimizar o portfólio. **P2P e Inovação**, Rio de Janeiro, RJ, v. 11, n. 2, e-7458, 2025. DOI: http://doi.org/10.21728/p2p.2025v11n2e-7458. In Portuguese.

POPPER, K. R. **A lógica da pesquisa científica**. 9. ed. São Paulo: Cultrix, 2001. 567 p. In Portuguese.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. Porto Alegre: AMGH, 2011. 780 p. In Portuguese.

SHEARD, S. A.; MOSTASHARI, A. A. Complexity Typology for Systems Engineering. **Incose International Symposium**, v. 20, n. 1, p. 933-945, 2010. DOI: https://doi.org/10.1002/j.2334-5837.2010.tb01115.x.

SOMMERVILLE, I. **Engenharia de software**. 10. ed. São Paulo: Pearson Prentice Hall, 2019. 756 p. In Portuguese.

SOURCEFORGE. **Top Software Downloads on SourceForge**. 2025. Available at: https://sourceforge.net/top. Accessed on: 13 Jan. 2025.

TIWARI, U. K. Conventional to component-based software: a critical survey on interaction and integration complexities. **International Journal of Advanced Research in Computer Science**, v. 8, n. 8, p. 135-139, 2017. DOI: https://doi.org/10.26483/ijarcs.v8i8.4617.

TRIOLA, M. F. **Introdução à estatística**: atualização da tecnologia. Rio de Janeiro: LTC Publisher, 2013. 707 p. In Portuguese.

VOGEL, M.; KNAPIK, P.; COHRS, M.; SZYPERREK, B.; PUESCHEL, W.; ETZEL, K.; FIEBIG, D.; RAUSCH, A.; KUHRMANN, M. Metrics in automotive software development: A systematic literature review. **Journal of Software: Evolution and Process**, v. 33, n. 2, e2296, 2020. DOI: http://doi.org/10.1002/smr.2296.