13th International Conference on Applied Informatics
Eszterházy Károly Catholic University
Eger, Hungary, February 19–21, 2026

Assessing software metrics and complexity in student and open-source projects using static analysis

Péter János Császár[a], Máté Cserép[b]

[a]Eötvös Lóránd University, Faculty of Informatics
csaszarpeter@inf.elte.hu

[b]Eötvös Lóránd University, Faculty of Informatics
mcserep@inf.elte.hu

Abstract

Maintaining code quality remains a persistent challenge not only in the software engineering industry but also in larger student projects. While conventional static analysis tools have proven effective in identifying syntax errors, type mismatches, and some well-known anti-patterns [4, 6], they often fall short in capturing deeper, structural issues that affect code quality, such as complexity, maintainability, and readability. By quantifying specific properties such as cyclomatic complexity [8], cohesion, and parameter count in functions, metrics can reveal patterns that may not be immediately obvious through traditional analysis methods [11, 13].

For the TMS (Task Management System) project [3], previous works have already developed tools supporting automatic static code analysis [5, 12]. Building on this foundation, my research aims to extend these tools to enable complexity- and cohesion-based analysis of student projects written in C#, using various software metrics, and focusing on the automatic detection of problematic code segments and code smells. The metrics examined and implemented during the research include the *Lack of Cohesion of Methods (LCOM)* [7], the *Bumpy Road* [2], *Functional arguments*, the *Maintainability Index* [9], *Class coupling* [1], and *Cyclomatic Complexity* [8]. The implementation is based on the well-known .NET Compiler Platform SDK (Roslyn) [10]. These metrics can be used to determine the complexity and cohesion of the code. By continuously monitoring these metrics, it becomes possible to track their fluctuations throughout the development lifecycle and to evaluate the coherence of the system at the module and type levels.

The analyzers were applied to student projects from the *Software Technology* course at *ELTE FI*. Projects from previous semesters, written in C# were evaluated by calculating several metrics on them. The analyzers were executed on three major project milestones, and the results were compared across these stages. Although a gradual increase in complexity and cohesion was expected as the projects progressed, some cases exhibited sharp increases, suggesting the presence of code that is difficult to read and excessively complex. The presented tool is capable of locating these problematic segments, which can then be reviewed by the project leader. Another objective was to implement a tool that enables the automatic download and milestone-based analysis of the course projects from a given year, facilitating the automated identification of code segments that may warrant closer inspection based on metric results. This research has the potential to improve the quality of student projects and facilitate the early identification of potential problems during the development phase.

Furthermore, to verify the effectiveness and general applicability of the implemented code metrics, they have also been calculated on open-source projects. Analyzing open-source software can provide valuable insights into how these metrics behave across different development environments, with varying coding standards, team structures, and project scales. This broader evaluation can assist validating the robustness and applicability of the implemented metrics. Due to the wide range of targeted .NET versions and dependency configurations across projects, the analysis was conducted using a Docker-based environment to provide controlled and reproducible execution contexts.

The results demonstrate that metric-based static analysis can effectively support the early detection of problematic code segments and provide actionable insights to improve software quality. The approach is adaptable and shows potential for integration into continuous development workflows in both academic and real-world environments.

References

[1] L. Briand, P. Devanbu, W. Melo: *An investigation into coupling measures for C++*, in: Proceedings of the 19th International Conference on Software Engineering, ICSE '97, Boston, Massachusetts, USA: Association for Computing Machinery, 1997, pp. 412–421, isbn: 0897919149, doi: 10.1145/253228.253367, url: https://doi.org/10.1145/253228.253367.

[2] CodeScene: *The Bumpy Road Code Smell: Measuring Code Complexity by its Shape and Distribution*, url: https://codescene.com/engineering-blog/bumpy-road-code-complexity-in-context/, Accessed: 2026.01.07.

[3] M. Cserép, P. Kaszab: *Task Management System*, url: https://tms-elte.gitlab.io, Accessed: 2026.01.07.

[4] I. V. Gomes, P. Morgado, T. Gomes, R. M. L. M. Moreira: *An overview on the Static Code Analysis approach in Software Development*, in: 2009, url: https://api.semanticscholar.org/CorpusID:9373127.

[5] P. Kaszab: *Automated evaluation of programming assignments with static code analysis*, url: https://tms-elte.gitlab.io/theses/kaszab_peter_tdk.pdf, Accessed: 2025.02.01.

[6] P. Kaszab, M. Cserép: *Detecting Programming Flaws in Student Submissions with Static Source Code Analysis*, Studia Universitatis Babeș-Bolyai Informatica 68 (July 2023), pp. 37–54, doi: 10.24193/subbi.2023.1.03.

[7] E. N. H. Kirğil, T. E. Ayyildiz: *Analysis of Lack of Cohesion in Methods (LCOM): A Case Study*, in: 2021 2nd International Informatics and Software Engineering Conference (IISEC), 2021, pp. 1–4, doi: 10.1109/IISEC54230.2021.9672419.

[8] T. McCabe: *A Complexity Measure*, IEEE Transactions on Software Engineering SE-2.4 (1976), pp. 308–320, doi: 10.1109/TSE.1976.233837.

[9] Microsoft: *Code metrics - Maintainability index range and meaning*, url: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning?view=vs-2022, Accessed: 2025.04.27.

[10] Microsoft: *The .NET Compiler Platform SDK*, url: https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/, Accessed: 2026.01.07.

[11] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, C. Soubervielle-Montalvo: *Source code metrics: A systematic mapping study*, Journal of Systems and Software 128 (2017), pp. 164–197, issn: 0164-1212, doi: https://doi.org/10.1016/j.jss.2017.03.044, url: https://www.sciencedirect.com/science/article/pii/S0164121217300663.

[12] B. D. Orosz: *Automated validation of design and architectural patterns on student assignments with static code analysis*, url: https://tms-elte.gitlab.io/theses/orosz_balint_dominik_tdk.pdf, Accessed: 2026.01.07.

[13] L. H. Rosenberg, T. Hammer, J. G. Shaw: *SOFTWARE METRICS AND RELIABILITY*, in: 1998, url: https://api.semanticscholar.org/CorpusID:59798472.