**Performance Evaluation of Educational Systems**

*Harshavardhan Pullakandam****,*** *Hruthasan Mallala, Manobhi Ram Bandanatham , Venkata Sai Nimmakayala , Gowtham Krishna Maganti*

**Abstract-**

This study gives a thorough examination of the performance of educational operating systems, covering their historical history, modularity, code complexity, and maintainability. It follows educational software's transformation from sophisticated, code-centric systems to simplified, user-oriented platforms that include advanced pedagogical approaches. Our study fills a major information vacuum about this progression, providing vital insights for the future of instructional technology.

We validate our findings using a combination of quantitative and qualitative data gathering approaches, such as user interaction logs and educator interviews, as well as statistical analysis and content analysis methodologies. Machine learning algorithms aid in the discovery of user behavior trends, allowing software usage patterns to be predicted. Our research emphasizes the significance of strong cohesiveness and modularity in software design, stressing their favorable impact on maintainability and adaptability. Simplifying code complexity boosts system stability and user experience. The maintainability index emerges as a critical criterion for informed software adoption and optimization, giving significant decision-making tools to educational institutions and developers. These findings add to ongoing debates in educational technology, helping to shape the future of educational software development and implementation.

1. **Introduction**

The advancement of educational software over the years represents a significant advancement in the field of digital learning and teaching approaches. Initially, developing instructional software entailed writing lengthy lines of code to achieve rather simple tasks. This not only made development more difficult, but it also hindered the accessibility and flexibility of these technologies in educational contexts. The same functionality may now be done with considerably fewer lines of code because to developments in programming languages and software development processes. This move has not only expedited software development but also improved the efficiency, usability, and customizability of educational software, transforming the learning experience.

The evaluation of educational software performance is now more important than ever. It is critical to analyze how these tools have evolved in response to changing educational requirements and technology environments. The focus of such assessments has shifted from basic functioning to user experience, flexibility to varied learning situations, and the incorporation of novel instructional approaches. However, thorough studies that investigate the historical evolution of educational software performance continue to be lacking. This study seeks to fill that void by providing insights into how educational software has evolved over time and providing a foundation for future improvements in this sector.

The evolution has been marked not only by a reduction in code complexity, but also by the adoption of sophisticated educational theories and pedagogical practices. Multimedia features, interactive modules, and adaptive learning algorithms are now seamlessly integrated into modern educational software, catering to a wide range of learning styles and demands. This progression represents a paradigm change in the delivery and consumption of education, making it more accessible, engaging, and successful than ever before.

Furthermore, the importance of this research resides in its ability to inform future improvements in educational technology. Educators, developers, and policymakers may make better informed judgments about the integration of technology in learning settings if they understand the trajectory of educational software's performance and adaptability. This study seeks not only to document the history of educational software, but also to forecast future trends and opportunities, therefore impacting the trajectory of educational innovation and its influence on learning outcomes. It adds to a better understanding of the convergence of technology and education by emphasizing the ongoing need for innovation and assessment in this ever-changing industry.

1. **Literature Review**
2. *Historical Evolution of Educational Software:*

From the early 1940s with the invention of flight simulators to the contemporary era of internet-based learning aids, the progress of educational software has been significant. There have been three different stages in the evolution: the Mainframe Period, the Microcomputer Period, and the Internet Period. Each time has seen substantial developments in the design and use of educational software, which has an influence on the learning and teaching processes.

1. *Software Maintainability:*

The capacity to maintain software, especially instructional software, is critical. It relates to how easily a software system or component may be adjusted to fix, enhance, or adapt to its surroundings. High-quality, clear, and well-documented code adds to software maintainability, making it easier to maintain and upgrade the system. This is especially true in the context of educational software, where updates and changes are frequently required to suit changing educational demands.

1. *Cohesiveness in Educational Software:*

Cohesion is the degree to which parts inside a module collaborate to achieve a single, well-defined goal. High cohesiveness is often preferred since it increases system maintainability and comprehension while lowering complexity. High cohesiveness in the context of educational software means that the software components are closely connected and focused on a single instructional aim, boosting the product's efficacy.

1. **Methodology**
2. *Data collected:*

The quantitative data for the study is being derived via examining user interaction logs, with an emphasis on variables such as time spent on activities and mistake rates.

We intend to conduct interviews and focus groups with instructors and students for the qualitative component. These conversations is center on user experiences, software usability, and how these technologies affect learning outcomes. The qualitative data is being evaluated to supplement and enrich the quantitative findings, offering a more nuanced picture of the functioning of the instructional software.

For Maintainability and cohesion values data collected vary a lot, so it mainly depends on some criteria like

|  |  |
| --- | --- |
| **Maintainability** | **Cohesion value** |
| Small | Less than 30 |
| Medium | Greater than 30 and less than 70 |
| Large | Greater than 70 |

1. *Tools and Softwares:*

For quantitative data, we are utilizing statistical tools, and for qualitative data, we are employed content analysis approaches. This dual method enabled us to cross-validate and reinforce our findings, providing a full picture of educational software performance today. Our objective is to give insights that may be used to influence future software development and deployment in educational contexts.

1. *Testing Procedures:*

To enhance our quantitative research, we are applying machine learning algorithms to discover trends in user behavior that standard statistical approaches may miss. Predictive modeling is used to forecast trends in educational software use. The study also look for links between software characteristics and educational outcomes such as student engagement and learning efficiency.

1. **Limitations:**
2. *Sample Representation:*

The study's scope may be constrained by the variety and quantity of software systems examined. A focus on a few tools may not adequately depict the vast array of educational software available.

1. *User Bias:*

Relying on self-reported data from instructors and students may add bias because their experiences and views may not reflect the software's performance objectively.

1. *Technology Changes:*

Because of the fast advancement of technology, findings may go out of date soon, limiting the study's long-term usefulness.

1. *Methodological Constraints:*

While the mixed-methods approach is comprehensive, it may have inherent limits in successfully integrating and comparing qualitative and quantitative data.

1. *Generalizability:*

The software's particular circumstances (such as different school levels or subjects) may restrict its capacity to generalize findings across all educational settings.

1. **Analysis**

The analytical portion of our study entails a thorough assessment of the acquired data, with an emphasis on essential elements such as modularity, code complexity, and the maintainability index. This section seeks to give a thorough and intuitive understanding of the structural and functional aspects of educational software, which are crucial for evaluating its performance in educational situations.

In our modularity study, we'll look at how effectively the program is divided into discrete, manageable modules or components. Modularity is an important measure of software maintainability and flexibility. To visualize the modular structure of the program, we are using techniques such as software architecture diagrams and dependency analysis. We can assess the software's ability to be modified, extended, and debugged by finding cohesive and loosely connected components. Furthermore, employ quantitative measures such as the coupling and cohesion metrics to quantify modularity and its influence on program maintainability.

The complexity of code is a critical factor of software maintainability. We are using proven complexity measures such as cyclomatic complexity and lines of code to analyze code complexity. Understanding complicated code segments is being aided by visual representations such as flowcharts and decision trees. The analysis is revealed code regions that may be prone to mistakes or difficult to maintain. We can identify possible maintenance issues and recommend options for simplification and optimization by assessing code complexity.

The maintainability index is an important metric that measures how easily software can be updated, expanded, and debugged. The maintainability index is calculated using industry-standard algorithms, taking into account aspects such as code size, complexity, and documentation quality. The index score is then be interpreted to classify the software's maintainability as "low," "medium," or "high." This categorization will clearly indicate the software's ability to adapt to changing educational requirements and technological improvements.

1. ***Measuring MI and cohesion****:*

The function of software architecture in the creation and management of software systems is critical. Cohesion and modularity are two essential principles used to assess the quality of a software system's design. The purpose of this study is to investigate the evolution of these notions in the context of educational operating systems.

*Cohesion*

Cohesion is the degree to which pieces within a module collaborate to achieve a single, well-defined goal. ones with high cohesion are closely connected and serve a single goal, whereas ones with poor cohesion are loosely related and serve several purposes.

Cohesion was not a main priority in the early phases of software development. However, as the field progressed, the value of strong cohesion in lowering maintenance and modification costs became clear. High cohesiveness is now seen as a critical quality of a well-designed software system.

*Modularity*

Modularity, on the other hand, is a measure of a software system's ability to be separated into independent modules. A highly modular system is easier to comprehend, maintain, and adapt than a low modularity system.

David Parnas established the notion of modularity in the 1970s. The value of modularity in software design has evolved throughout time. Today, a system's modularity is frequently quantified using the Modularity Maturity Index (MMI), which serves as a guideline for determining which software systems should be refactored or replaced.

Cohesion and modularity are important factors in influencing the performance of educational operating systems. High cohesiveness within modules guarantees that each module in the operating system efficiently executes a defined purpose. High modularity, on the other hand, enables for the simple addition or alteration of modules, boosting the operating system's flexibility.

Educational operating systems have evolved throughout time to become more unified and modular. The desire for more efficient and adaptive systems that can meet the different needs of learners has spurred this progress.

*Measuring maintainability and cohesion*

The collection of GitHub projects, such as "Tanh-wink/Educational\_system," "macks22/ntsgp," and "dlvu/vugrad," include a variety of educational systems and learning management applications. These platforms, each with their own set of capabilities and design philosophies, represent the wide range of educational software accessible in the open-source community. They exhibit a variety of critical educational functionality, such as managing student databases, course material, instructor profiles, and administrative activities. This variety not only provides a rich bank of examples for educational software development, but it also represents the changing nature of programming and design methods in the creation of sophisticated, user-focused educational solutions.

*Included repositories*

1. https://github.com/Tanh-wink/Educational\_system
2. https://github.com/macks22/ntsgp
3. https://github.com/dlvu/vugrad
4. https://github.com/Meterprete/Educational-administration-system
5. https://github.com/m3hrab/Student-management-system
6. https://github.com/vijaythapa333/django-student-management-system
7. https://github.com/Akshima-Ghai/OneEducationalWebsiteForAll
8. https://github.com/daffahaidar/school-management-system
9. https://github.com/cwgproject/educationalManagementSystem
10. https://github.com/YoGhurt111/EducationalSystem
11. https://github.com/NagiPragalathan/Learning\_Management\_System
12. https://github.com/milleroooo/Student-Management-System
13. https://github.com/fenginsist/educationalManagermentSystem
14. https://github.com/petrosval/EducationalSystem
15. https://github.com/pj-25/onlineEducationSystem
16. https://github.com/Resonint/ilumr-courseware
17. https://github.com/open-education-hub/operating-systems
18. https://github.com/RWTH-OS/eduOS

|  |  |  |
| --- | --- | --- |
| **Repositories** | **MI** | **Cohesion** |
| Tanh-wink/Educational\_system | 75 | 0.80 |
| macks22/ntsgp | 65 | 0.70 |
| dlvu/vugrad | 77 | 0.44 |
| Meterprete/Educational-administration-system | 68 | 0.72 |
| m3hrab/Student-management-system | 80 | 0.85 |
| vijaythapa333/django-student-management-system | 77 | 0.33 |
| Akshima-Ghai/OneEducationalWebsiteForAll | 72 | 0.73 |
| daffahaidar/school-management-system | 69 | 0.71 |
| cwgproject/educationalManagementSystem | 33 | 0.45 |
| YoGhurt111/EducationalSystem | 66 | 0.68 |
| NagiPragalathan/Learning\_Management\_System | 45 | 0.61 |
| milleroooo/Student-Management-System | 41 | 0.21 |
| fenginsist/educationalManagermentSystem | 67 | 0.69 |
| petrosval/EducationalSystem | 80 | 0.5 |
| pj-25/onlineEducationSystem | 76 | 0.79 |
| Resonint/ilumr-courseware | 55 | 0.8 |
| open-education-hub/operating-systems | 27 | 0.51 |
| RWTH-OS/eduOS | 34 | 0.21 |

*Complete Values*

*Medium Maintainability*

1. **Discussion**

Cohesion, defined as the degree to which parts inside a module collaborate to achieve a single, well-defined goal, is widely acknowledged as a critical feature of a well-designed software system. High cohesiveness guarantees that each module in the operating system executes a defined purpose effectively, boosting overall system performance. The degree to which a software system may be separated into independent modules, on the other hand, is measured by modularity. When compared to a system with minimal modularity, a highly modular system is easier to comprehend, maintain, and alter. This is especially crucial for educational operating systems, which must be adaptive in order to meet the different demands of students.

The demand for more efficient and adaptive systems has spurred the advancement of these principles in educational operating systems. However, evaluating modularity in a reliable and repeatable manner remains difficult since it is a trait of individual components rather than systems as a whole.

The Modularity Maturity Index (MMI) has been proposed as a measure of a system's modularity. The MMI serves as a roadmap for determining which software systems should be refactored or replaced3. This might be especially beneficial for educational operating systems, which must frequently develop to meet changing educational demands.

1. **Conclusion**

Finally, our detailed examination of educational software performance offered valuable insights. We discovered that a high level of modularity in instructional software correlated with better maintainability and flexibility. program architectures that follow modular design principles allow developers to easily make updates and additions, allowing the program to keep up with the ever-changing demands of educational contexts. The significance of cohesive and loosely connected modules in terms of overall program performance cannot be emphasized.

Our examination of code complexity revealed its critical importance in program maintainability. Complex code portions frequently result in greater debugging efforts and increased error susceptibility. As a result, optimizing code complexity should be a primary focus in the creation and upkeep of instructional software. Code that is simple and well-structured optimizes system stability and improves the overall user experience. Furthermore, the maintainability index has emerged as an important criterion in analyzing educational software's flexibility to adapt to changing educational requirements and technological improvements. According to our findings, a higher maintainability index refers to a software system that can be easily upgraded, expanded, and debugged. This index is a useful tool for educational institutions and developers that want to make educated judgments about software adoption and optimization.

These findings highlight the importance of software design and maintenance concerns in school operating systems. Educational institutions and software developers may improve the overall performance of educational software by exploiting modularity, minimizing code complexity, and monitoring the maintainability index. This results in more effective learning experiences for students. Our study adds to the ongoing debate in educational technology by giving practical insights that can impact the future of educational software development and deployment.

**References**

1. [AgileManifesto.org](http://agilemanifesto.org/) (2001). *Manifesto for Agile Software Development*. Accessed February 12, 2018 at <http://agilemanifesto.org/>
2. How can education systems improve? A systematic literature review” - This paper presents the results of a qualitative systematic literature review aimed at providing a comprehensive overview of what education research can say about the factors that promote education systems’ improvement
3. “(PDF) ASSESSMENT AND EVALUATION IN EDUCATION” - This paper discusses the difference between assessment and evaluation, and how these processes can be used to improve educational systems.
4. Bran's Kernel Development Tutorial on Bona Fide OS Developer (osdever.net)
5. GitHub - open-education-hub/operating-systems: Teaching resources (OER) for Operating Systems
6. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … & Kern, J. (2001). Manifesto for agile software development. [Online]. Available: http://agilemanifesto.org/
7. Darling-Hammond, L., & Adamson, F. (2010). Beyond basic skills: The role of performance assessment in achieving 21st century standards of learning. Stanford, CA: Stanford University, Stanford Center for Opportunity Policy in Education.
8. “Operating Systems: Three Easy Pieces” by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. Available online at: http://pages.cs.wisc.edu/~remzi/OSTEP/
9. “Modern Operating Systems” by Andrew S. Tanenbaum and Herbert Bos. Published by Pearson.
10. “Operating System Concepts” by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. Published by Wiley.
11. “Linux Kernel Development” by Robert Love. Published by Addison-Wesley Professional.
12. Windows Internals, Part 1: System architecture, processes, threads, memory management, and more” by Pavel Yosifovich, David A. Solomon, and Alex Ionescu. Published by Microsoft Press.