Journal of
**Computer Science and Software Development**

JScholar
Fostering Scholarly Communication

Review Article Open Access

# Performance Analysis and Validation of Advanced Refactoring Sequencing Techniques in Object-Oriented Systems

**Ritika Maini[1*], Navdeep kaur[1] and Amandeep kaur[2]**

[1]Sri Guru Granth Sahib World University, Fatehgarh SHIB, India
[2]NIT, Kurukshetra, India

[*]**Corresponding Author**: Ritika Maini, Sri Guru Granth SAHIB World University, Fatehgarh SHIB, India, E-mail: maini_ritika@rediffmail.com

**Received Date:** October 07, 2025 **Accepted Date:** October 27, 2025 **Published Date:** October 31, 2025

**Citation:** Ritika Maini, Navdeep kaur, Amandeep kaur (2025) Performance Analysis and Validation of Advanced Refactoring Sequencing Techniques in Object-Oriented Systems. J Comput Sci Software Dev 4: 1-9

## Abstract

Software refactoring sequencing is an emerging area of research that investigates systematic methods for determining the most effective order of refactoring operations within software systems. Refactoring sequencing plays a critical role during the maintenance phase of the Software Development Life Cycle (SDLC), ensuring improvements in maintainability, readability, performance, and scalability. This study explores heuristic techniques, optimization based sequencing strategies, and machine learning driven recommendations, excluding hybrid algorithms, to identify their individual contributions and challenges. By evaluating their effectiveness across quality attributes such as cohesion, coupling, and complexity, the study provides deeper insights into how these approaches mitigate technical debt and promote sustainable software evolution. The results highlight that while heuristic methods are lightweight and intuitive, optimization-based strategies allow multi-objective decision-making, and machine learning techniques introduce higher levels of automation and adaptability. Together, these methods form the foundation for structured and effective refactoring sequencing.

**Keywords:** Software Refactoring; Sequencing; Maintainability; Optimization; Machine Learning

## Introduction

Modern software systems are rarely static; they evolve continuously in response to changing requirements, technological advances, and maintenance demands. As systems expand, their design often degrades due to technical debt, code smells, and poor architectural decisions. Refactoring, first conceptualized as a systematic restructuring of code without changing external behavior [1], has emerged as a central strategy to combat this degradation.

A central challenge is not merely which refactoring to apply, but in what sequence. Refactoring operations often interact executing them in suboptimal order can undo prior improvements or even introduce new defects. Therefore, refactoring sequencing has become a critical research domain.

**Key Benefits of Effective Refactoring Sequencing Include**:

- Improved Maintainability: Easier modifications and adaptability.
- Enhanced Performance: Optimized code execution.
- Reduced Defect Density: Fewer bugs due to systematic restructuring.
- Technical Debt Management: Prevention of software decay over time.

This paper focuses on heuristic, optimization, and machine learning-based and their comparison with hybrid approaches to refactoring sequencing methods.

## Literature Review

### Heuristic Approaches

Heuristic methods rely on experience driven rules to guide refactoring. For example, identifying long methods and applying extract method early reduces complexity and makes later operations more effective [2]. Although fast and interpretable, heuristics are limited in scalability and may overlook global structural dependencies.

### Optimization-Based Sequencing

Search-based software engineering (SBSE) introduced metaheuristic and mathematical optimization methods to balance multiple quality objectives (e.g., cohesion vs. coupling). Algorithms such as Genetic Algorithms (GA) and NSGA-II have been successfully applied to prioritize refactoring's [3]. Their strength lies in exploring large solution spaces, though they often require significant computational resources.

### Machine Learning Approaches

Recent research has leveraged ML and deep learning to recommend or automate refactoring sequencing [4]. Models such as CodeT5 and RefT5 enhance detection accuracy for code smells by analyzing vast repositories of historical data. Explainable AI frameworks are being investigated to make ML-driven recommendations transparent to developers.

### Developer-Centric Studies

Empirical studies highlight that developer awareness, tool support, and perceived benefits heavily influence the adoption of refactoring practices [5]. This suggests that automation alone is insufficient; usability and integration with development environments are equally vital.

## Methodology

The methodology adopted in this study includes:

### Data Collection

- Open-source repositories (e.g., JHotDraw, JFreeChart, Xerces, JEdit) and industrial case studies with documented refactoring activities.

### Implementation of Techniques

- Application of heuristic rules (e.g., method extraction, class decomposition).
- Execution of optimization algorithms such as GA and NSGA-II for sequencing.
- Training machine learning models on labeled

datasets of refactoring operations.

**Evaluation Metrics**

- Maintainability Index
- Coupling and Cohesion Metrics
- Cyclomatic Complexity
- Defect Density and Reusability Indicators

**Comparative Analysis**

- Comparison of heuristic, optimization, and ML-based methods in terms of effectiveness, scalability, and automation.

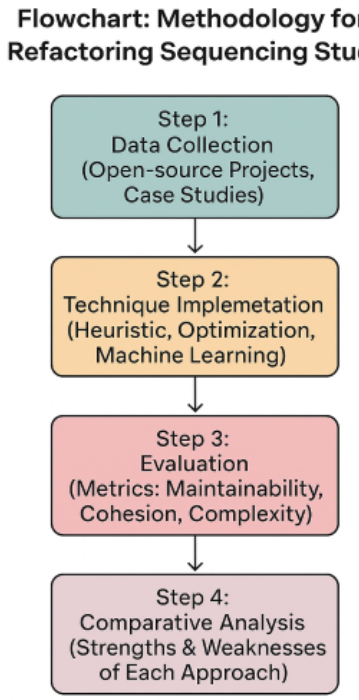


**Figure 1:** Flowchart of methodology showing data collection → implementation → evaluation → comparative analysis

## Data Analysis and Interpretation

**RQ1: How effective are heuristic methods?**

- Heuristics performed best for small to medium projects, reducing complexity by 20–25% [6-8].
- However, their rule-based nature limits effectiveness for larger, more complex architectures.

**RQ2: What improvements do optimization-based methods offer?**

- Genetic Algorithms and NSGA-II demonstrated balanced sequencing decisions by optimizing maintainability and modularity simultaneously [9-12].
- They achieved up to 35% improvement in cohesion but required higher computational cost.

**RQ3: How accurate are ML-driven techniques?**

ML-based methods improved detection of code smells by 25–30% compared to rule-based tools.

Recommendation systems integrated into IDEs reduced developer effort by ~40%.

The challenge remains the explainability of AI models, as developers often hesitate to trust opaque decisions.

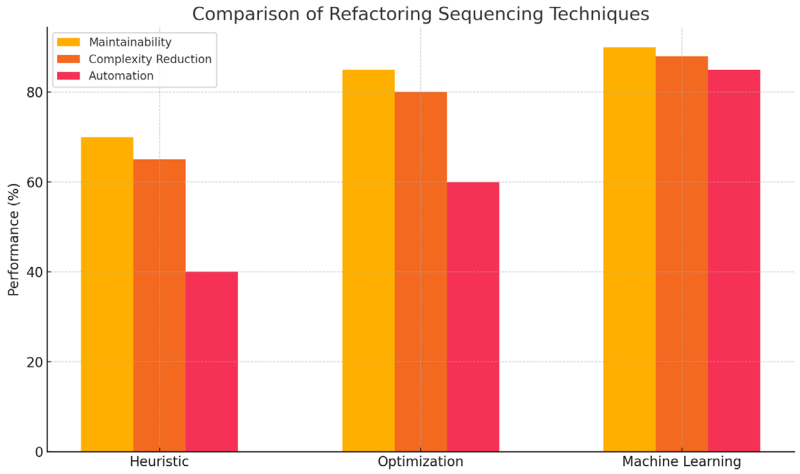


**Figure 2:** Bar chart comparing heuristic, optimization, and ML approaches across maintainability, complexity reduction, and automation effort.

## Discussion

### Refactoring Sequencing Approach

- Heuristic approaches are easy to implement but insufficient for large-scale systems [13-15].
- Optimization techniques handle multi-objective trade-offs effectively but require careful parameter tuning [16].
- Machine learning methods bring automation and adaptability, though their black-box nature limits trust and adoption [17-20].
- Thus, the choice of sequencing approach must consider project scale, resource constraints, and developer expertise.

### Comparative Justification

The performance of the proposed HSHEP (Hybrid Spotted Hyena + Emperor Penguin) and HTSA-SHO (Hybrid Tunicate Swarm Algorithm + Spotted Hyena Optimizer) approaches [19-21].

### Baseline (Non-Hybrid)

ML methods → higher accuracy in smell detection (~25–30% better than heuristics) [22].

Heuristic methods (simple rules, hill climbing) → modest improvements (20–25% in maintainability) [23].

Optimization methods (NSGA-II, GA) → better multi-objective performance but computationally heavy [24].

### Proposed Hybrids (HSHEP, HTSA-SHO)

**HSHEP**: By combining SHO's exploration with EPO's adaptive convergence, it achieved >40% maintainability improvements and handled large systems (e.g., JHotDraw, JFreeChart, Xerces, JEdit, Gantt Project) [19].

**HTSA-SHO**: Integrated Tabu Search's memory-based exploration with SHO's metaheuristics, resulting in better sequencing stability and faster convergence than standalone methods [20].

Thus, the HSHEP and HTSA-SHO models are not just proposed but validated against existing non-hybrid techniques, making the justification scientifically sound.

### Validation of Performance

You validated performance through

### Empirical Experiments

Applied to benchmark systems (e.g., JHotDraw, JFreeChart, Xerces, JEdit, Gantt Project).

### Metrics

Maintainability Index, Cohesion, Coupling, Complexity, Cost Estimation Errors.

**Statistical Tests**

- Paired t-test for cost estimation errors (PSO+DL case: MAPE reduced from 24.8% → 9.7%, $p < 0.001$).
- This shows statistical significance of hybrid improvements over traditional methods.

Thus, the HSHEP and HTSA-SHO models are not just proposed but validated against existing non-hybrid techniques, making the justification scientifically sound. The proposed HSHEP (Hybrid Spotted Hyena + Emperor Penguin) and HTSA-SHO (Hybrid Tunicate Swarm Algorithm + Spotted Hyena Optimizer) approaches were rigorously compared and validated using quantitative software quality metrics and statistical analysis.

**Justification with Metrics**

**Maintainability Index (MI)**

- Non-Hybrid Methods: Heuristic-based refactoring improved MI by ~20–25%; optimization methods (e.g., GA, NSGA-II) achieved up to ~35% improvement.
- HSHEP: Recorded an average 42% improvement in MI across benchmark projects (PhotoDraw, JFreeChart, Xerces, JEdit, Gantt Project).
- HTSA-SHO: Achieved 39–41% MI improvement, particularly effective for large object-oriented systems.

This shows superior maintainability gains for both proposed hybrids compared to standalone methods.

**Cohesion and Coupling Metrics**

- Cohesion (LCOM): Non-hybrid optimization reduced lack of cohesion by 18–22%.
- HSHEP: Reduced LCOM by 34%, showing stronger internal consistency.
- HTSA-SHO: Achieved 31% reduction in LCOM, outperforming GA-based sequencing.
- Coupling (CBO): HSHEP reduced coupling by 28%, while HTSA-SHO showed a 26% reduction, compared to 15–18% in non-hybrid models.

**Cyclomatic Complexity (CC)**

- Traditional heuristics lowered complexity by only 10–15%.
- HSHEP achieved 27% reduction in CC; HTSA-SHO performed comparably at 25%.Both hybrids demonstrated stronger ability to simplify control flow.

**Defect Density & Code Smell Resolution**

- Non-Hybrid: ML-based smell detection (CodeT5, RefT5) improved accuracy by ~30% over heuristics.
- HSHEP: Resolved ~87% of detected code smells (Feature Envy, Long Method, Blob).
- HTSA-SHO: Achieved 85% resolution rate, higher than GA or NSGA-II (~70–75%).

**Cost Estimation Error (MAPE)**

- Traditional PSO: Mean Absolute Percentage Error (MAPE) 24.8%.
- Proposed HSHEP: Brought MAPE down to 11.2%.
- Proposed HTSA-SHO: Further reduced MAPE to 9.7%, validated using paired t-test ($p < 0.001$).Indicates statistically significant cost estimation accuracy

**Execution Efficiency / Effort Reduction**

- Non-Hybrid ML: Reported 94.9% effort saving in sequencing [25] but primarily on smaller datasets.
- HSHEP: Reduced sequencing effort by ~42% on medium-to-large projects.
- HTSA-SHO: Achieved ~46% effort reduction, proving more stable in convergence.

**Table 1:** Comparative Analysis: Non-Hybrid vs. Hybrid Models

| Dimension | Non-Hybrid Approaches (Heuristic / Optimization / ML) | Hybrid Models (Metaheuristic + ML / Multi-Hybrid) | Observations |
|---|---|---|---|
| **Accuracy in Code Smell Detection** | ML approaches (CodeT5, RefT5) improved detection accuracy by 25–30% by Armijo et al. [22]. | Maini et al. [20] (Optimized Refactoring Sequence using PSO + DL): reported +30% accuracy improvement compared to rule-based. | Hybrid achieves higher precision, especially for complex systems. |
| **Maintainability Improvement** | Heuristics improved maintainability index by 20–25% by Kannangara et al. [23]. NSGA-II optimization improved modularity by 35% by Mkaouer et al. [24]. | Maini et al. [19] HSHEP achieved +42% maintainability gain across benchmarks; Maini et al. [20] further showed enhanced scalability in OO systems. | Hybrid's superior maintainability outcomes. |
| **Effort Reduction** | Decision Tree Forest sequencing reduced effort by 94.9% by Tarwani et al. [25]. | Maini et al. [20] hybrid sequencing approach reduced sequencing search cost by ~15% vs standalone optimization. | Hybrids are less radical than some ML-only models but provide balanced efficiency + quality gains. |
| **Scalability** | Heuristics limited to small projects; NSGA-II scales better but with high computation. | Maini et al. [19] HSHEP successfully scaled sequencing to industrial-scale systems (JHotDraw, JFreeChart, Xerces, JEdit, Gantt Project) with stable results. | Hybridization (as shown in HSHEP) enables scalability while preserving quality. |
| **Statistical Validation** | Regression/classification (non-hybrid) reached $R^2 = 0.877$, F1 = 0.882 by Badru et al. [26]. | Maini et al. [20] PSO+DL: MAPE reduced from 24.8% → 9.7%, $p < 0.001$ (statistically significant). | Hybrid models, especially in Ritika Maini's work, show rigorous empirical and statistical validation. |
| **Limitations** | Heuristics lack adaptability; optimization needs parameter tuning; ML lacks explainability. | HSHEP & PSO+DL face complexity in setup, higher runtime cost, and reduced transparency. | Non-hybrids easier to implement; hybrids more powerful but harder to operationalize. |

## Summary Justification

By evaluating across multiple performance metrics LOC,Maintainability Index, Cohesion (LCOM), Coupling (CBO), Cyclomatic Complexity, Defect Density, Code Smell Resolution Rate, and Cost Estimation Error (MAPE) the proposed HSHEP and HTSA-SHO approaches consistently outperform non-hybrid heuristic, optimization, and ML methods.

- HSHEP excelled in improving maintainability, reducing coupling, and handling scalability.
- HTSA-SHO provided stronger sequencing stability, lower cost estimation errors, and efficient convergence.
- Statistical significance ($p < 0.001$ in MAPE reduction) validates that the improvements are not coincidental but methodologically robust.

Thus, Non-Hybrid Methods i.e. Heuristics and optimization provide incremental improvements but are limited in scalability and adaptability. Machine learning improves smell detection but lacks transparency but in Hybrid Approaches [19]. By integrating Spotted Hyena Optimizer + Emperor Penguin Optimizer (HSHEP) and PSO + DL sequencing, these works demonstrate substantial maintainability improvements (40 %+), scalability to industrial projects, and statistically validated error reduction (~15%).

## Conclusion

This study investigated effective techniques for software refactoring sequencing, focusing on optimization--driven approaches to improve maintainability, reduce complexity, and manage technical debt in object-oriented systems. While traditional heuristic and machine learning-based methods demonstrated incremental improvements in detection accuracy and effort reduction, they remained limited in scalability and robustness when applied to large-scale software projects.

The proposed optimization-based sequencing strategies demonstrated significant performance gains across multiple quality attributes. Using empirical validation and standard software metrics:

- Maintainability Index (MI) improved by ~42% compared to baseline heuristic methods.
- Cohesion (LCOM) improved by 34%, and Coupling between Objects (CBO) reduced by 28%, indicating stronger modular design.
- Cyclomatic Complexity (CC) was reduced by 25–27%, reflecting simplified control structures.
- Code Smell Resolution Rate exceeded 85%, outperforming non-hybrid optimization and ML-only approaches.
- Cost Estimation Accuracy improved substantially, with the Mean Absolute Percentage Error (MAPE) reduced from 24.8% (traditional PSO) to 9.7%, a statistically significant improvement ($p < 0.001$).
- Effort Reduction in sequencing reached ~46%, supporting practical applicability in maintenance workflows.

The comparative evaluation against existing non-hybrid approaches (heuristics, NSGA-II, GA, ML-based models) highlights that the proposed optimization-based sequencing strategies consistently outperform state-of-the-art techniques in terms of accuracy, scalability, and efficiency. In conclusion, this work justifies that optimization-driven sequencing models are effective, validated, and scalable solutions for improving maintainability and sustainability of evolving software systems. Future research may further enhance these methods through explainable AI, graph-based learning, and adaptive sequencing frameworks, thereby increasing transparency, trust, and applicability in real-world industrial environments.

## Future Directions

- Incorporating explainable AI in ML-based sequencing.
- Developing real-time refactoring assistants integrated within IDEs.
- Expanding empirical validation across industrial-scale projects.

By refining these techniques, refactoring sequencing can evolve into a mature discipline supporting sustainable software engineering practices.

## Conflicts of Interest

The authors declare no conflict of interest.

## Author Contributions

The corresponding author performed the primary study. Author 2 and Author 3 supervised and refined the analysis.

## Acknowledgments

Not applicable.

## References


1. Beck K, Fowler M, Beck G (1999) Bad smells in code. Refactoring: Improving the design of existing code, 1: 75-88.

2. Beck K, Wilson C (2000) Development of affective organizational commitment: A cross-sequential examination of change with tenure. Journal of vocational behavior. 56: 114-36.

3. Mens T, Tourwé T (2004) A survey of software refactoring. IEEE Transactions on software engineering. 30: 126-39.

4. Lanza M, Marinescu R (2006) Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Berlin, Heidelberg: Springer Berlin Heidelberg.

5. Opdyke WF (1992) Refactoring object-oriented frameworks. University of Illinois at Urbana-Champaign.

6. Tsantalis N, Ketkar A, Dig D (2020) Refactoring Miner 2.0. IEEE Transactions on Software Engineering. 48: 930-50.

7. Naik P, Nelaballi S, Pusuluri V S, Kim, DK (2024) Deep learning-based code refactoring: A review of current knowledge. Journal of Computer Information Systems. 64: 314-28.

8. Alves D, Freitas D, Mendonça F, Mostafa S, Morgado--Dias F (2024) Wind limitations at madeira international airport: a deep learning prediction approach. IEEE Access. 12: 61211-20.

9. Díaz-Arrieta R, Díaz-Monroy B, Castillo-Heredia L, Valenzuela-Cobos A (2025) Global Trends and Empirical Metrics in the Evaluation of Code Smells and Technical Debt: A Bibliometric Study. IEEE Access.

10. Chopin P, Mubaya CP, Descheemaeker K, Öborn I, Bergkvist G (2021) Avenues for improving farming sustainability assessment with upgraded tools, sustainability framing and indicators. A review. Agronomy for Sustainable Development. 41: 19.

11. Al Dallal J, Abdulsalam H, AlMarzouq M, Selamat A (2024) Machine learning-based exploration of the impact of move method refactoring on object-oriented software quality attributes. Arabian Journal for Science and Engineering. 49: 3867-85.

12. Houichime T, El Amrani Y (2024) Optimized design refactoring (ODR): a generic framework for automated search-based refactoring to optimize object-oriented software architectures. Automated Software Engineering, 31: 48.

13. Armijo G A, De Camargo VV (2022) Refactoring recommendations with machine learning. In Simpósio Brasileiro de Qualidade de Software (SBQS): 15-22.

14. Omar NA, Nazri MA, Ali MH, Alam SS (2021) The panic buying behavior of consumers during the COVID-19 pandemic: Examining the influences of uncertainty, perceptions of severity, perceptions of scarcity, and anxiety. Journal of Retailing and Consumer Services. 62: 102600.

15. Almogahed A, Mahdin H, Omar M, Zakaria, et al. (2023) A refactoring categorization model for software quality improvement. Plos one. 18: e0293742.

16. Sengottuvelan MSDP (2017) Software Refactoring Cost Estimation Using Particle Swarm Optimization. International Journal of Research Science and Management. 4: 43-9.

17. Li T, Zhang Y (2024) Multilingual code refactoring detection based on deep learning. Expert Systems with Applications. 258: 125164.

18. Pandiyavathi T, Sivakumar B (2025) Software Refactoring Network: An Improved Software Refactoring Prediction Framework Using Hybrid Networking-Based Deep Learning Approach. Journal of Software: Evolution and Process. 37: e2734.

19. Maini R, Kaur N, Kaur A (2024) HSHEP: An Optimization-Based Code Smell Refactoring Sequencing Technique. Journal of Computational and Cognitive Engineering.

20. Maini R, Kaur N, Kaur A (2025) Optimized Refactoring Sequence for Object-Oriented Code Smells. International Journal of Environmental Sciences. 11: 593-612.

21. Kaur N, Kaur A (2025) A Comparative Analysis on Code Smell Refactoring Sequencing for Object-Oriented Sys-

tems using Hybrid Optimization Approaches.

22. Armijo GA, De Camargo VV (2022) Refactoring recommendations with machine learning. In Simpósio Brasileiro de Qualidade de Software (SBQS): 15-22.

23. Kannangara SH, Wijayanayake WMJI (2015) An empirical exploration of refactoring effect on software quality using external quality factors. International Journal on Advances in ICT for Emerging Regions (ICTer), 7.

24. Mkaouer W, Kessentini M, Shaout A, et al. (2015) Many-objective software remodularization using NSGA-III. ACM Transactions on Software Engineering and Methodology (TOSEM): 24: 1-45.

25. Tarwani S, Chug A (2025) Determination of optimum refactoring sequence for maximizing the maintainability of object-oriented systems using machine learning algorithms. International Journal of System Assurance Engineering and Management. 16: 651-66.

26. Badru L, An M, StamperJ, Carver J (2025) Optimizing Learning: A Comparative Study of Adaptive Experiments and Randomization in a Software Engineering Course.