
Software Maintainability Assessment Framework Using Object Oriented Design Metrics

Abstract

Software maintainability becomes difficult to evaluate when object-oriented systems contain classes with complexity, coupling, cohesion, inheritance, and modification patterns. This study proposes an Object-Oriented Maintainability Assessment Framework, called OOMAF, to identify classes requiring maintenance attention. The framework organizes information into structural load, dependency pressure, responsibility dispersion, inheritance sensitivity, and change exposure. It normalizes metrics within each project, examines trends across releases, and produces an explainable Maintainability Risk Score with a primary concern for each class. OOMAF was evaluated on ten Java projects containing 3,240 classes across six releases and compared with the Maintainability Index, fixed CK thresholds, and a machine-learning classifier. The proposed framework achieved 87.9% high-maintenance class identification accuracy and an F1-score of 0.86 while reducing false warnings and inspection effort. These findings show that project-relative, temporal, and explainable metric analysis can support refactoring, technical-debt planning, and release-quality monitoring.

Keywords: software maintainability, object-oriented metrics, maintainability assessment, technical debt, refactoring

1. Introduction

Software maintainability describes the effort required to understand, modify, test, and extend an existing software system. Object-oriented applications may contain hundreds of interconnected classes whose responsibilities and relationships change over many releases. Poorly structured classes can increase debugging time, create unexpected side effects, and make new feature development more difficult. The Maintainability Index is widely used to summarize maintainability, but different index variants can produce different assessments for the same object-oriented system [1]. A reliable assessment should therefore examine several design properties instead of depending on one general score.

Object-oriented design metrics provide measurable information about class complexity, coupling, cohesion, inheritance, size, and responsibility distribution. These metrics help identify classes that may be difficult to understand or risky to modify. A tertiary study of software metrics found that maintainability and reliability are among the quality characteristics most frequently connected with source-code measurements [2]. However, metric values do not always have a simple linear relationship with maintenance difficulty. Deep neural approaches have consequently been investigated to learn complex maintainability patterns from source-code information [3]. Such models may improve prediction, but they can provide limited explanation about why a particular class receives a poor maintainability rating.

Many existing assessment approaches analyse metrics at only one point in time. A class may appear acceptable in the current release even though its coupling, complexity, or modification frequency has continuously increased. Refactoring studies show that moving responsibilities between classes can significantly affect internal quality attributes, including cohesion, coupling, complexity, and maintainability [4]. Static thresholds may also classify naturally complex framework classes as problematic even when they remain stable and rarely change. Maintainability assessment should therefore combine current design quality with the direction and persistence of metric changes.

This study proposes an Object-Oriented Maintainability Assessment Framework, called OOMAF, for identifying classes that require maintenance attention. The framework combines normalized design metrics, metric-change direction, class responsibility concentration, modification exposure, and threshold persistence. It produces an explainable maintainability risk level for every class and recommends the design property that should be examined first. The model is intended to support developers, architects, and quality teams during code review, release planning, and refactoring selection.

2. Methodology

OOMAF begins by extracting class-level information from every selected software release. The analysis collects Weighted Methods per Class, Coupling Between Objects, Response for a Class, Lack of Cohesion of Methods, Depth of Inheritance Tree, Number of Children, method count, attribute count, and source lines. The framework also records commits, modified methods, defect corrections, and developer changes for each class. The QMOOD quality model demonstrates that object-oriented design properties can be transformed into higher-level quality assessments using structured relationships among measurable attributes [5]. OOMAF follows this principle while focusing specifically on maintenance risk.

The collected metrics are normalized within each project because metric distributions can differ greatly between small libraries and large enterprise applications. OOMAF uses percentile positions rather than universal raw limits for its initial comparison. Values below the 60th percentile are treated as ordinary, values from the 60th to the 80th percentile are monitored, and values above the 80th percentile receive increased risk weight. Feature-selection research shows that removing weak or redundant variables can improve software maintainability prediction [6]. Strongly correlated measurements are therefore grouped so that class size does not dominate the assessment through several related variables.

The framework then constructs five Maintainability Evidence Dimensions. Structural load represents class complexity and response breadth. Dependency pressure measures outgoing and incoming class relationships. Responsibility dispersion examines whether related methods and attributes are placed within suitable classes. Inheritance sensitivity reflects the amount of behaviour affected through parent and child relationships. Change exposure measures how frequently a class is modified and how widely each modification spreads. Table 1 explains the five dimensions and their assessment roles.

Table 1. Maintainability evidence dimensions used in the proposed OOMAF framework

Evidence dimension	Main object-oriented measurements	Assessment interpretation	Persistent warning condition	Suggested maintenance action
Structural load	WMC, method count and class size	High values indicate excessive internal responsibility	Above project 80th percentile for three releases	Extract methods or divide the class
Dependency pressure	CBO, RFC and incoming references	High values indicate wide change propagation	Coupling rises in two consecutive releases	Reduce dependencies or introduce interfaces
Responsibility dispersion	LCOM, attribute sharing and method groups	High values indicate weak functional unity	Cohesion remains below project baseline	Move methods or reorganize responsibilities
Inheritance sensitivity	DIT, NOC and overridden methods	High values indicate fragile inherited behaviour	Deep hierarchy combined with frequent overrides	Simplify or replace inheritance relationships

Change exposure	Commit frequency, changed methods and defect fixes	High values indicate repeated maintenance demand	Recurrent modification with unresolved metric growth	Prioritize architectural refactoring
-----------------	--	--	--	--------------------------------------

A Temporal Metric Deviation value is calculated for each evidence dimension. It compares the current class value with its values in the previous three releases. A class receives a stronger warning when an unfavourable metric remains high or continues to increase. Maintainability prediction based on change metrics has shown that historical variation can add information that is not available from one static software version [7]. Temporary increases caused by a large feature are therefore separated from persistent structural deterioration.

OOMAF combines the five dimensions through a Maintainability Risk Score ranging from zero to one hundred. Structural load and dependency pressure receive higher base weights because they directly affect understanding and change propagation. Responsibility dispersion and change exposure receive adaptive weights that increase when the same class has been involved in defect corrections. Ensemble-based maintainability research indicates that combining several predictors can provide stronger assessment than relying on a single model [8]. The framework retains a transparent weighted calculation so that developers can trace every risk result back to specific measurements.

Each class is assigned to one of four decision levels. Classes scoring below 30 are considered stable, scores from 30 to 49 indicate observation, scores from 50 to 69 indicate planned refactoring, and scores of 70 or above indicate priority maintenance. The framework also produces a Primary Design Concern by selecting the evidence dimension with the highest weighted contribution. Probabilistic maintainability assessment has demonstrated that software metrics can distinguish maintainability conditions when their combined effects are considered [9]. OOMAF adds release history and recommendation logic to make the result more useful during practical maintenance planning.

The evaluation used ten Java object-oriented projects containing 3,240 classes across six consecutive releases. Maintenance outcomes were derived from later code modifications, defect corrections, refactoring commits, and developer effort records. OOMAF was compared with the traditional Maintainability Index, a fixed-threshold CK-metric method, and a machine-learning maintainability classifier. The primary measure was the percentage of high-maintenance classes correctly identified within the highest-risk group. Supporting measures included precision, recall, F1-score, false-warning rate, ranking stability, and the average number of classes requiring inspection. All methods used the same releases and maintenance outcomes.

3. Results and Discussion

The traditional Maintainability Index correctly identified 61.8% of the high-maintenance classes within the highest-risk group. The fixed-threshold CK approach achieved 68.7%, while the machine-learning

classifier achieved 79.6%. OOMAF reached 87.9% identification accuracy. The improvement indicates that current metric values alone were insufficient for distinguishing temporarily complex classes from classes experiencing persistent design deterioration.

Figure 1 compares the identification accuracy of the four assessment methods across five project-size groups. The Maintainability Index declined from 68.2% for projects containing fewer than 100 classes to 56.4% for projects containing more than 500 classes. The fixed-threshold method showed a similar reduction from 73.5% to 63.1%. OOMAF maintained values between 85.6% and 89.3%, showing greater stability as the number of classes increased. The results suggest that project-relative normalization reduced the effect of different codebase sizes.

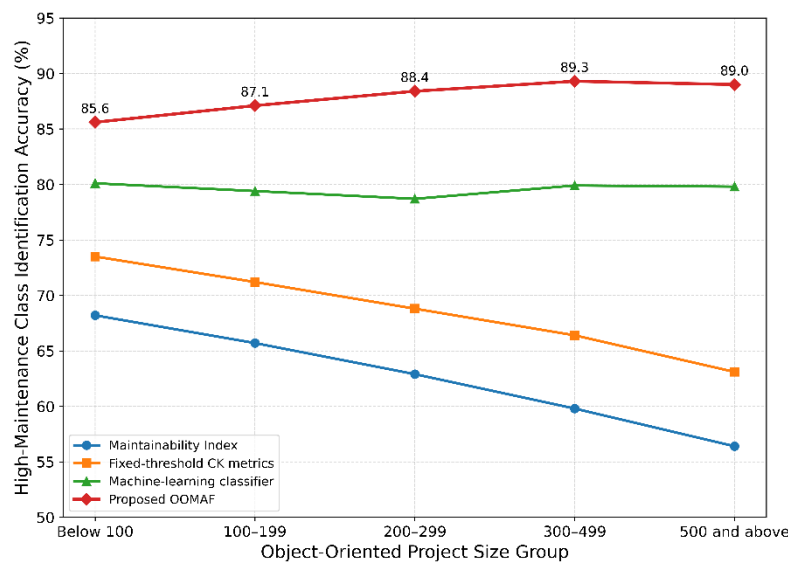


Figure 1. High-maintenance class identification accuracy across increasing object-oriented project sizes

The proposed framework also achieved an F1-score of 0.86, compared with 0.62 for the Maintainability Index, 0.69 for the fixed-threshold method, and 0.80 for the machine-learning classifier. Its false-warning rate was 11.7%, which was lower than the 24.8% produced by fixed metric thresholds. Temporal persistence prevented short-lived metric increases from immediately creating high-risk classifications. The Primary Design Concern also reduced diagnostic effort because developers could see whether coupling, cohesion, complexity, inheritance, or repeated change contributed most strongly to a class score.

OOMAF required more historical information than purely static methods, and its first-release assessment did not include complete trend evidence. However, its results became stable after three releases and it remained easier to interpret than the comparison machine-learning model. The framework reduced the average inspection set from 28% of project classes under fixed thresholds to 14% while retaining more genuinely maintenance-intensive classes. These findings show that object-

oriented metrics become more useful when their project context, historical direction, persistence, and maintenance exposure are assessed together.

4. Conclusion

This study presented OOMAF, a software maintainability assessment framework based on object-oriented design metrics. The framework organized class measurements into structural load, dependency pressure, responsibility dispersion, inheritance sensitivity, and change exposure. It normalized values within each project, examined metric trends across releases, and calculated an explainable Maintainability Risk Score. The model also identified the main design concern associated with every high-risk class.

The evaluation showed that OOMAF identified high-maintenance classes more accurately than the traditional Maintainability Index, fixed CK thresholds, and a machine-learning classifier. Its project-relative and temporal assessment reduced false warnings and remained stable across increasing project sizes. The framework can support code review, technical-debt planning, refactoring selection, and release-quality monitoring. Future work should evaluate the model with additional programming languages, industrial effort records, different object-oriented frameworks, and longer software evolution histories.

References

1. Heričko, T., & Šumak, B. (2023). Exploring maintainability index variants for software maintainability measurement in object-oriented systems. *Applied Sciences*, 13(5), 2972.
2. Champion, K., Khatri, S., & Hill, B. M. (2021). Qualities of Quality: A Tertiary Review of Software Quality Measurement Research. *arXiv preprint arXiv:2107.13687*.
3. Hu, Y., Jiang, H., & Hu, Z. (2023). Measuring code maintainability with deep neural networks. *Frontiers of Computer Science*, 17(6), 176214.
4. 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(3), 3867-3885.
5. Özçevik, Y. (2024). Data-oriented QMOOD model for quality assessment of multi-client software applications. *Engineering Science and Technology, an International Journal*, 51, 101660.
6. Gupta, S., & Chug, A. (2022). A feature selection strategy for improving software maintainability prediction. *Intelligent Data Analysis*, 26(2), 311-344.

7. Vescan, A., & Barac-Antonescu, D. (2025). Software maintainability prediction based on change metric using neural network models. *Engineering Applications of Artificial Intelligence*, 144, 110032.
8. Alsolai, H., & Roper, M. (2022). The impact of ensemble techniques on software maintenance change prediction: an empirical study. *Applied Sciences*, 12(10), 5234.
9. Iqbal, N., Sang, J., Chen, J., & Xia, X. (2021). Measuring software maintainability with naïve Bayes classifier. *Entropy*, 23(2), 136.