

An Explainable Multi-Criteria Framework for Prioritizing Refactoring in Object-Oriented Software Using Understandability and Testability Indicators

Manoj Kumar¹ , Abdullah², and Mohit Bajpai³

Research Scholar, Department of Computer Science, Shubham University, Bhopal, Madhya Pradesh, India
Professor, School of Computing and Information Science, Chandigarh University, Unnao, Uttar Pradesh, India
Professor, Department of Computer Science, Shubham University, Bhopal, Madhya Pradesh, India

Correspondence should be addressed to Manoj Kumar; mnj76.kumar@gmail.com

Received: 17 Sep 2026; **1st Revision:** 30 Sep 2026; **2nd Revision:** 13 Oct 2026; **Accepted:** 24 Oct 2026;
Published Online: 30 Oct 2026

Copyright © 2026 Manoj Kumar et al. This is an open-access article distributed under the [Creative Commons Attribution 4.0 International License \(CC BY 4.0\)](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

ABSTRACT- Interacting design attributes affect the quality of object-oriented software. Our previous studies have developed with individual regression models on understandability and testability. Nonetheless, individual scores do not provide a maintainer with direct information of which classes should be reviewed or refactored initially when there is a shortage of time and resources. This article suggests a multi-criteria model that is explainable and based on understandability and testability indicators and can help to refactor on the class level. The framework maintains the two dimensions as two independent indicators, normalizes the values and incorporates project priorities which are configurable to and generates a ranked list of classes with an explanation which is of metric level. Contrary to an opaque aggregate score, the suggested method will reveal why a particular class is prioritized and which indicators are monitored that allow it to be ranked. The paper establishes the framework architecture, scoring formulation, prioritization algorithm, as well as an empirical validation procedure that relies on object-oriented systems and judging by independent experts. There are no additional experimental findings; the empirical validation and implementation are necessary and only after that, the effectiveness of the framework will become definite.

KEYWORDS- Software Quality, Understandability, Testability, Multi-Criteria Framework, Object-Oriented Design

I. INTRODUCTION

The functions of software maintenance entail error correction, system reconfiguring according to current conditions and enhancing current functions. These factors influence maintenance effort; the ease with which developers comprehend a component, and the ability of developers to create and execute tests on a component. There are similar concerns which are not interchangeable: a class can be easy to read and hard to test or can be testable and hard to understand. It can be more helpful to maintain the two dimensions distinct rather than to have one overall score on quality. Our earlier research introduced UEMOOD,

a regression-based model to estimate the understandability using of object-oriented design property, and TEMOOD, a regression-based model to estimate testability [1], [2]. The following research question is operational: in what cases classes have dissimilar understandability and testability issues they can be asked how a team should afford limited refactoring effort and why so? The present paper suggests a refactoring-prioritization multi-criteria framework that can be explained. It is not a replacement of the original assessment models but is complementary. It makes the notions of understandability and testability remain visible as distinct dimensions, produces a visible priority order, and describes the contribution of each dimension. Its framework conceptually correlates to ISO/IEC 25010:2023 [3] software product-quality.

It has contributions, such as: (1) the workflow where the independent quality measurements were linked to a refactoring queue to the class; (2) the transparent score, whose weights might be tuned and incomplete inputs were clearly handled; (3) the explanations of priorities were given at the metric level; and (4) the comparison of the framework with single-dimension baselines and expert rankings was made.

II. BACKGROUND AND RELATED WORK

A. Understandability, Testability

Our earlier research introduced a software Understandability, which was initially called UEMOOD, based on object-oriented design properties, such as polymorphism, coupling, cohesion and inheritance [1]. The second, AI Based Testability Evaluation: An Important Element in Integrity, suggested TEMOOD based on design features such as polymorphism, coupling and inheritance [2]. They both rely on regression-based evaluation and compare model estimates/ranks with known values. Understandability issues the extra work required to interpret the structure and behavior of a component, whereas testability is whether appropriate tests can be easily designed and run. Weighted methods per class, coupling, response complexity, and cohesion are object-oriented

design metrics that are used to place on the established methods to describe design structure [7], [17] [20]. Software-science measures and complexity measures provide complementary measures, but the interpretation of the measures must be pragmatic, that is, depending on the context of the language and measurement use [8], [9], [16]. Design quality models and engineering principles also encourage the approach of considering quality to be multidimensional as opposed to quality being a unidimensional count [10], [13]. The procedure of refactoring is a well-known technique of enhancing internal structure, although the impact of refactoring must be assessed instead of intended [11], [12], [14], [15]. Concepts and terminology of software-testing must be aligned with the testing standard used [20].

B. Maintainability Research

The ISO/IEC 25010:2023 offers a model on product quality where software products are specifications on and assessment of software quality [3]. Previous literature has analyzed the object-oriented class maintainability, empirical maintainability data sets and information, as well as the information-driven prediction methods [4]-[6], [19]. The Goal Question Metric Methodology provides assistance in matching the measures to clear research questions [13], whereas well-established software-engineering review and experimentation advice advocates a fixed protocol, independent testing and reporting [14], [15]. Coupling and cohesion measurement as individual design properties were explored previously [17], [18] too.

C. Research Gap

Individual quality dimensions are defined in separate estimates but do not propose an action order per se when it comes to limited engineering resources. Research has been refactored to investigate the quality influence and practical difficulties in industry on a case-by-case basis [12], [14], [15]. Empirical research on understandability and maintainability recommends that quality indicators are seen

contextually and contrasted with the manifested evidence instead of presumed to be general predictors [4]-[6], [19]. The current framework fills this operational gap by creating the combination rule, weighting and dimension-wise contributions.

III. OBJECTIVE AND RESEARCH QUESTIONS

The objective is to design a transparent framework for prioritizing object-oriented classes for refactoring using understandability and testability assessments.

RQ1: Can separate understandability and testability indicators be combined into a transparent class-level priority score without hiding either dimension?

RQ2: Do the resulting priorities agree with independent expert judgments more closely than rankings based on either dimension alone?

RQ3: How sensitive are the priorities to the selected weights, normalization method, and uncertainty in input estimates?

RQ4: Can the framework provide actionable explanations of the design indicators contributing to each class's priority?

IV. PROPOSED FRAMEWORK

A. Architecture

The framework consists of six steps, namely (1) gather class-level metrics of design; (2) compute or import UEMOOD and TEMOOD estimates; (3) verify completeness and model applicability; (4) normalize both quality indicators; (5) compute a configurable refactoring-priority score; (6) generate a ranked queue with explanations. Other validated estimators can be used provided their outputs and constraints are reported. The entire architecture is shown in Figure 1 and the score-calculation and ranking path singled out in Figure 2 to enable the input transformations and decision steps to be viewed independently.

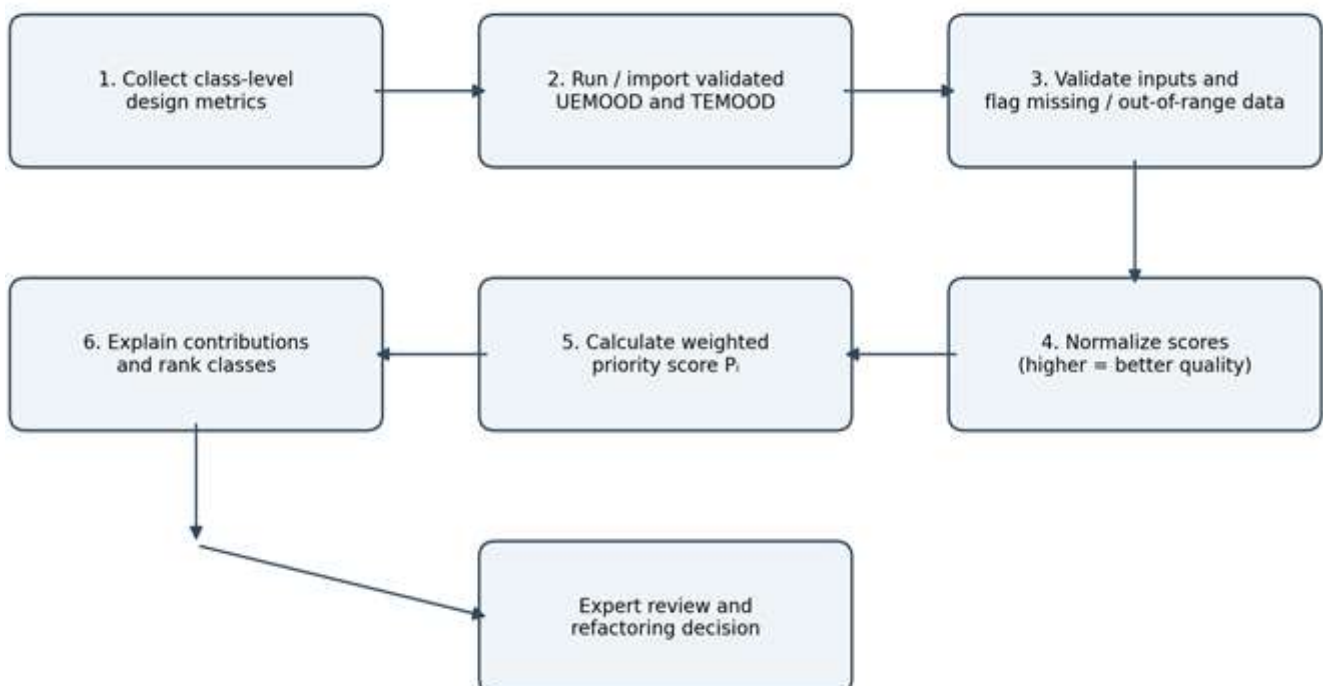


Figure 1: Architecture of the proposed explainable multi-criteria refactoring-prioritization framework.

Figure 1 shows how the framework proposed transforms the information of the object-oriented design into an explanation-capable refactoring-priority list. To gather the understandability and testability estimates of the original UEMOOD and TEMOOD models, a set of class-level design metrics will be gathered and utilized to estimate them. The completeness and applicability of the inputs are then verified to ensure the absence and inappropriateness of values to avoid scoring. As the two estimates might have varied scales, they are scaled to a standard range. The normalized scores are weighted together to give each class a score on priority; according to the suggested formulation, a low estimated quality level will result in a high refactoring priority. Lastly, the framework identifies the classes and displays the understandability and testability contributions of each score, which helps the developers understand why a class is of priority. The ranked list is not meant to substitute engineering judgment but assist it and is assessable to independent expert judgments.

B. Input and Model outputs.

Each class i must have the input record with the design metrics of the chosen UEMOOD and TEMOOD implementations, the resulting estimate of understandability U_i , the estimating estimate T_i , estimator versions, and a data-quality flag. Specific regression coefficients and metric definitions should be taken out of the original papers; there is no silent change of those coefficients that have been made in this paper. Recalibration of a new project population should be a separate training step reported.

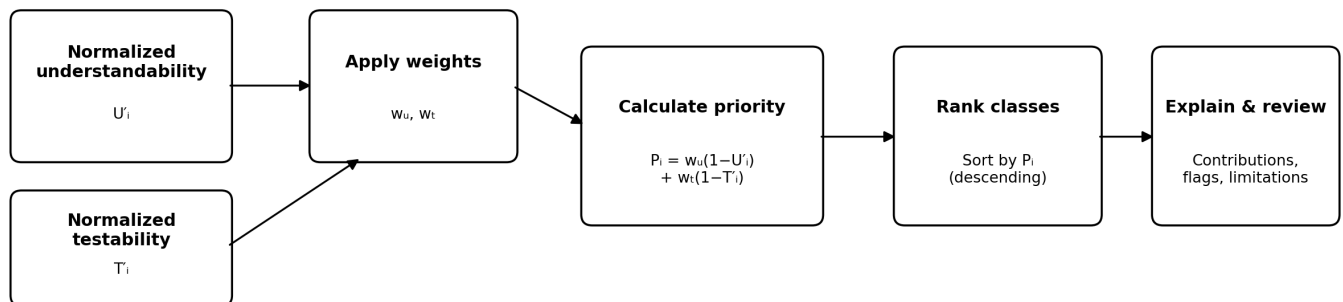


Figure 2: Workflow for calculating and ranking class-level refactoring-priority scores.

As shown in Figure 2, the framework first aligns the direction and scale of the two quality estimates, and then calculates the contributions of understandability and testability according to the pre-defined weights. The contributions are summed to get the class priority score. The eligible classes are ranked in descending order and linked to dimension-level explanations and data-quality flags. This figure is a description of the calculation sequence. It is not measured performance.

E. Explanation Generation

Each class's explanation should show (1) its normalized understandability score; (2) its normalized testability score; (3) each dimension's weighted contribution; and (5) data-quality or extrapolation warnings. Define $C_{ui} = w_u(1 - U_i)$ and $C_{ti} = w_t(1 - T_i)$, so $P_i = C_{ui} + C_{ti}$. This arithmetic decomposition makes the score inspectable but does not establish causal explanations for the effects of individual design metrics. The mapping in Table 1 links these signals

C. Normalization

The two indicators need to be mapped to some common scale. Considering a higher value as high-quality, a minmax normalization of a specified calibration set is possible: This calibration process is consistent with general practice in measurement: The scale, direction, and calibration population should be specified consistently prior to making observations [13], [16].

$$U'_i = (U_i - U_{min}) / (U_{max} - U_{min})$$

$$T'_i = (T_i - T_{min}) / (T_{max} - T_{min})$$

The calibration ranges must be fixed before evaluating the test projects. If a source model produces a difficulty or risk score where higher values are worse, its direction must be reversed before aggregation. A zero denominator must be flagged rather than divided by zero. Robust scaling or percentile ranks may be evaluated in sensitivity analysis.

D. Refactoring-Priority Score

Let U'_i and T'_i range from 0 to 1, where larger values indicate better quality. Define:

$$P_i = w_u(1 - U'_i) + w_t(1 - T'_i), \text{ where } w_u \geq 0, w_t \geq 0, \text{ and } w_u + w_t = 1.$$

A higher P_i indicates greater combined concern under the selected weights. Equal weights may be used as a baseline, but are not claimed to be optimal. Teams may select weights to reflect project constraints; empirical studies must declare and test weight sensitivity. The separate U'_i and T'_i scores should always be displayed so that the aggregate does not conceal a severe problem in one dimension.

to possible review actions without treating a metric threshold as proof of a defect.

Table 1: Mapping Of Quality Indicators to Potential Refactoring Review Actions

Quality dimension	Indicator / signal	Potential concern	Review action (example)
Understandability	Low normalized U'_i	Class may be difficult to interpret or maintain	Review naming, method responsibility, cohesion, and class decomposition
Understandability	Out-of-range coupling or cohesion metric	Dependencies or responsibilities may hinder	Inspect dependency structure and consider responsibility-

		comprehension	focused refactoring
Testability	Low normalized T_i	Tests may be difficult to design, isolate, or execute	Review dependency injection, controllability, observability, and test seams
Combined assessment	High P_i from both dimensions	Multiple quality concerns may coincide	Prioritize for joint understandability and testability review
Combined assessment	High contribution from one dimension	One quality dimension primarily drives priority	Select a targeted review while retaining the other dimension in the report

Table 1 matches each quality dimension and indicator to a possible area of concern and a suggested review action. Engineering inspection triggers the actions, the actions themselves are not prescriptions of automatic refactoring, the developers are encouraged to confirm the root cause of the design issue and evaluate behavior-preservation constraints to make changes.

F. Missing Data and Uncertainty

An estimate out of the range supported by the model seen in a class with missing metrics or an approximation that simply misses a measure is not deserving of an apparently accurate priority. The implementation is to detect gaps in the fields and either not rank or run a pre-concerted imputation procedure tested on independent data. Do not construct confidence intervals when the underlying estimators are not uncertain about the measurement; measure the ability of the estimators to be stable by sensitivity analysis in terms of weight and normalization.

V. PRIORITIZATION ALGORITHM

Input: class-level design metrics; validated understandability estimator U ; validated testability estimator T ; weights w_u and w_t ; calibration ranges; review thresholds.

Output: ranked classes, separate quality indicators, priority scores, and explanations.

1. Validate metric definitions, units, completeness, and estimator versions.
2. Obtain U_i and T_i for each class using the specified estimators.
3. Flag missing values, unsupported ranges, and invalid estimates.
4. Normalize both indicators using fixed calibration ranges and align their direction so higher values mean better quality.
5. Calculate $C_{ui} = w_u(1 - U_i)$ and $C_{ti} = w_t(1 - T_i)$.
6. Calculate $P_i = C_{ui} + C_{ti}$ for eligible classes.
7. Apply pre-registered review flags to each dimension separately.
8. Sort eligible classes by descending P_i ; preserve ties and data-quality flags.

9. Generate explanations showing dimension scores, contributions, metric flags, and limitations.

10. Return the ranked list for expert review; do not automatically refactor code.

VI. EMPIRICAL VALIDATION PROTOCOL

A. Study Design

Test framework on observation of a number of object-oriented software systems on a class-level basis. Regional estimation and TEMOOD estimation Use projects not used to estimate or calibrate UEMOOD and TEMOOD. Choose systems based on pre-determined inclusion criteria, document versions and provenance of the source, and rationalize the sample size. Set programming languages, extraction tools and exclusion rules prior to data collection.

B. Expert Assessment by independent expert.

A stratified sample of classes should be evaluated using a written rubric of understandability, testability, and refactoring urgency concerns by at least two independent reviewers with relevant software-engineering experience. Framework priorities should not be viewed in the initial assessment by reviewers. Agreement with records, document the process of aggregation, and report inter-rater agreement.

C. Baselines

Compare the suggested structure to (1) UEMOOD-only ranking; (2) TEMOOD-only ranking; (3) equal-weight aggregation; (4) project-weighted aggregation and (5) a simple, pre-registered design-metric baseline. Assuming that there are enough labels available, a supervised model can be added, but when possible, train and test partitions should be provided within a project.

D. Evaluation Measures

Rank agreement: Use Spearman's rank correlation, and Kendall tau. When experts detect high-priority classes, the top-k precision, recall, and NDCG can be used to assess how the classes are ranked in the top. In the case of continuous estimates, continuous representations of MAE and RMSE can be also reported to specific reference labels. When sufficient size sample allows, report confidence limits and relevant statistical comparisons. Picking measures prior to checking final results. Inspecting the results should be specified in the evaluation plan before reporting, in enough detail to facilitate replication [14], [15]. NDCG k (Normalized Discounted Cumulative Gain at k) uses how well the ranking ranks expert-identified high-priority classes at the top; based on relevant classes showing lower in the list, the result is discounted then divided by an ideal ranking. The k value has to be determined beforehand and the relevance labels that are assigned by the experts need to compute this measure. NDCGk (Normalized Discounted Cumulative Gain at k) is used to understand whether high-priority classes suggested by the experts can be found at the top of the ranking. It discounts pertinent classes that are ranked lower in the listing and normalizes the increase against the ideal ranking. The relevance labels given to the experts have to be pre-specified and the value of k is necessary.

E. Sensitivity and Reproducibility

Compare normalization schemes and remove a single dimension, vary w and w_0 over a grid, and compare normalization schemes. Report the frequency with which classes remain in the high-priority. Publish extraction scripts, estimator versions, configuration files, expert-rating rubric and analysis code under licensing were allowed by licensing.

VII. ILLUSTRATIVE DESCRIPTIVE AND SENSITIVITY ANALYSIS

Since it was not provided with any observed project data, this section describes a reproducible simulation consisting of 30 hypothetical classes. The normalized concepts of understandability and testability scores were produced demonstratively, with a fixed random seed. These values do not come as a measurement of a software repository, or expert review or implementation of UEMOOD/TEMOOD. This analysis thus demonstrates the way in which descriptive statistics and weight sensitivity can be reported; it is not an attempt at demonstrating the empirical relevance of the solution framework.

The score of the hypothetical classes was assigned a priority score, $P_n = w_U (1-U) + w_N (1-T)$. The equal weights (w) ranking was considered with a weight of 0.5 (w). Sensitivity analysis took w_0, w_1, w_2 and so on, 0.1 at a time, $w_e = 1$ w . In every setting, Spearman rank disagreement was used to evaluate the rank agreeableness with the equal-weight rank, and the overlap between the five highest-priority classes measured.

The justifications on simulated results would be considered as a strict interpretation as a methodological demonstration. They demonstrate the workings of the scoring rule and how the rankings can be tested to be weight sensitive. They should not be quoted to be quantitative proofs that the integrated framework works better than either of the baselines. The empirical test should re-do the analysis using actual data about classes and, in the case of NDCG at k , or expert-agreement assertions, get independent expert relevance labels.

Table 2: Minimum Data Requirements for Empirical Evaluation and Reproducibility

Measure	Required reporting	Measured result
Dataset / projects	Number of projects, classes, languages, and inclusion criteria	Not measured yet; report the actual number of projects, classes, languages, and inclusion criteria after dataset selection.
Expert assessment	Number of reviewers; rubric; inter-rater agreement	Not measured yet; report reviewer count, assessment rubric, and inter-rater agreement after independent review.
Rank agreement	Spearman's ρ and Kendall's τ with confidence intervals	[Compute from held-out data]
Top-k prioritization	Precision@k, Recall@k, and/or NDCG@k	[Compute from held-out data]
Baseline comparison	UEMOOD-only, TEMOOD-only, equal-weight, proposed framework	Not measured yet; report the same metrics for UEMOOD-only, TEMOOD-only, and the combined framework.
Robustness	Weight-sensitivity results and rank stability	Not measured yet; populate after empirical evaluation.
Uncertainty / validity	Missing data, exclusions, effect sizes, statistical tests	[Document transparently]

Table 2 defines the minimum information required to make the future empirical evaluation interpretable and reproducible. The measured-result column states what must be calculated after implementation; it does not contain experimental findings.

Table 3: Planned Comparison of Prioritization Methods for Empirical Evaluation

Method	Ranking input	Role in evaluation	Required comparison	Key reporting consideration
UEMOOD-only	Normalized understandability estimate	Single-dimension baseline	Compare ranking agreement and top-k relevance against expert labels.	Use the same held-out projects and relevance labels as all other methods.
TEMOOD-only	Normalized testability estimate	Single-dimension baseline	Compare ranking agreement and top-k relevance against expert labels.	Use the same held-out projects and relevance labels as all other methods.
Combined framework (equal weights)	Understandability and testability with equal weights	Transparent combined baseline	Evaluate whether joint information improves agreement, relevance, and ranking stability.	Declare normalization and equal-weight settings before evaluation.

Combined framework (project-defined weights)	Both dimensions with pre-declared project weights	Sensitivity / policy variant	Compare with equal weights and report weight sensitivity.	Explain the basis for the weights; do not tune them on the test set.
Design-metric baseline	Pre-specified object-oriented design metrics	Simple comparison baseline	Evaluate against the same projects, expert labels, and measures.	Specify metrics and ranking rule before inspecting outcomes.

Table 3 presents the comparison of 5 prioritization methods: UEMOOD-only, TEMOOD-only, combined framework with equal weights, combined framework with project-defined weights, and design-metric baseline. It specifies the input used by each method, its role in the evaluation and the comparison or reporting consideration to be applied. All methods should be evaluated on the same held-out software projects, with the same expert relevance labels and pre-specified metrics. The table describes the evaluation design; it does not report measured performance or establish that any method is superior.

Table 4: Descriptive Statistics for the Simulated Class-Level Data

Variable	Mean	Std. dev.	Minimum	Median	Maximum
Understandability	0.599	0.173	0.198	0.588	0.950
Testability	0.576	0.211	0.100	0.591	0.950
Priority equal 0.5 0.5	0.412	0.180	0.050	0.433	0.851

Table 4 reports summary statistics for the simulated normalized quality scores and the equal-weight priority score. The values describe only the generated hypothetical dataset; they are not empirical measurements.

Table 5: Weight-Sensitivity Analysis on the Simulated Data

w_u	w_t	Spearman's ρ vs. equal weights	Top-5 overlap
1.0	0.0	0.865	0.60
0.9	0.1	0.914	0.80
0.8	0.2	0.948	0.80
0.7	0.3	0.980	0.80
0.6	0.4	0.992	0.80
0.5	0.5	1.000	1.00
0.4	0.6	0.996	1.00
0.3	0.7	0.980	1.00
0.2	0.8	0.966	1.00
0.1	0.9	0.947	1.00
0.0	1.0	0.942	1.00

Table 5 shows how changing the relative weight assigned to testability affects the priority ranking compared with the equal-weight baseline. Spearman's ρ measures overall rank agreement, while Top-5 overlap is the proportion of classes shared by the two highest-priority sets. These are computed on the simulated dataset only.

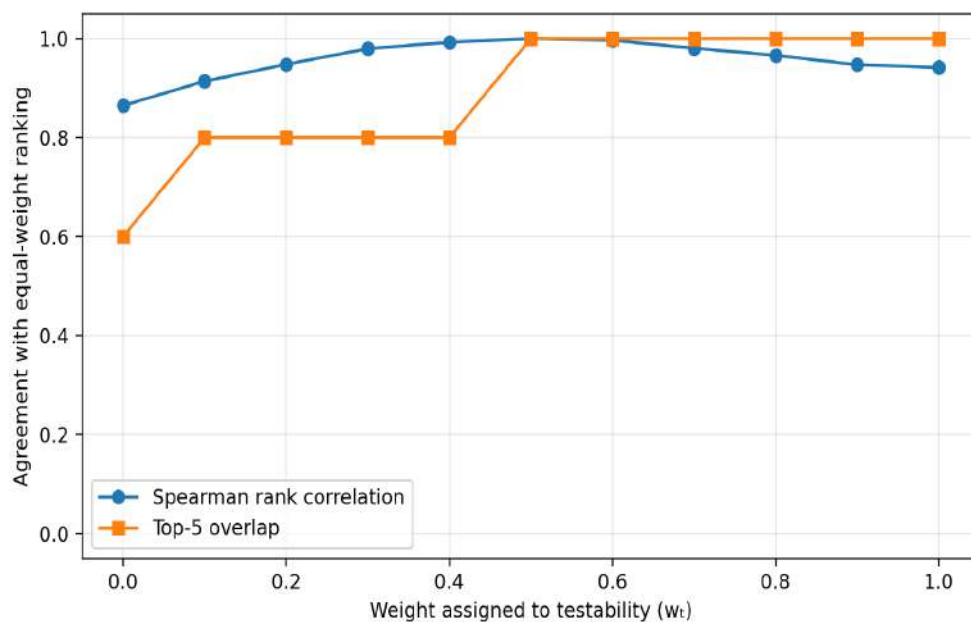


Figure 3: Weight sensitivity of ranking agreement and top-five overlap on simulated data.

The sensitivity results of Table 5 are shown in Figure 3. Horizontal axis varies the testability weight from 0 to 1. The understandability weight is the complementary value.

Agreement is measured against the equal weight ranking. The plot is illustrative in that it uses simulated inputs, not observed project data.

VIII. THREATS TO VALIDITY

Construct validity: expert ratings do not necessarily reflect all issues of understandability, testability or real maintenance effort. Label all constructs and employ homogeneous rubric.

Internal validity: performance may be inflated by internal data leakage, variable metric extraction, project calibration, or post-hoc weight selection. Independent calibration and testing, with holdout on a project level when possible.

External validity: outcomes of a weak sample of languages or repositories might not be generalizable. Describe project characteristics of report and do not make blanket statements.

Conclusion validity: Differences can be unstable when using small samples, when using dependent observations, when making multiple comparisons. Assume effects, report assumptions and confidence intervals.

Model validity: the framework is a limitation inheriting model of UEMOOD and TEMOOD. Reuse: Check their coefficients, provenance data, sample sizes and applicability. Identify any recalibration as compared to the original models.

IX. CONCLUSION AND FUTURE WORK

The paper suggests a multi-criteria model, which can be explained as linking individual understandability and testability measurements to a workflow that engages in refactoring-priorities. It maintains the two dimensions and reveals their accomplishments to an integrated plus score and sets data-quality controls, review flags, and an empirical protocol on comparing priorities with independent expert ratings.

The offered equations and algorithm set up a testable algorithm rather than empirical demonstration of efficacy. The implementation of the workflow, confirmation of initial UEMOOD and TEMOOD specifications, gathering of independent observations at the level of classes, and performing of baseline, sensitivity, and expert-agreement analyses are the next steps. Additional dimensions of maintainability, e.g. analysability, modifiability, can be involved into future work in case reliable measures and appropriate data on their validation can be found.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

- [1] M. Kumar, Abdullah, and M. Bajpai, "AI Based Understandability Assessment: A Key Contributor to Functionality," in Proc. 2025 International Conference on Metaverse and Current Trends in Computing (ICMCTC), 2025. Available from: <https://doi.org/10.1109/ICMCTC62214.2025.11196552>
- [2] M. Kumar, Abdullah, and M. Bajpai, "AI Based Testability Evaluation: An Important Element for Integrity," in Proc. 2025 International Conference on Metaverse and Current Trends in Computing (ICMCTC), 2025. Available from: <https://doi.org/10.1109/ICMCTC62214.2025.11196726>
- [3] ISO/IEC 25010:2023, Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Product Quality Model. Geneva, Switzerland: International Organization for Standardization, 2023. Available from: <https://www.iso.org/standard/78176.html>
- [4] J. Al Dallal, "Object-oriented class maintainability prediction using internal quality attributes," Information and Software Technology, vol. 55, no. 11, pp. 2028–2048, 2013. Available from: <https://doi.org/10.1016/j.infsof.2013.07.005>
- [5] M. Schnappinger, A. Fietzke, and A. Pretschner, "Defining a software maintainability dataset: Collecting, aggregating and analysing expert evaluations of software maintainability," in Proc. 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2020, pp. 278–289. Available from: <https://doi.org/10.1109/ICSME46990.2020.00035>
- [6] Azhir, M. Zakeri, Y. Abedini, and M. M. Ghahfarokhi, "QualCode: A data-driven framework for predicting software maintainability based on ISO/IEC 25010," Science of Computer Programming, vol. 250, Art. no. 103399, 2026. Available from: <https://doi.org/10.1016/j.scico.2025.103399>
- [7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, Jun. 1994. Available from: <https://doi.org/10.1109/32.295895>
- [8] T. J. McCabe, "A complexity measure," IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308–320, Dec. 1976. Available from: <https://doi.org/10.1109/TSE.1976.233837>
- [9] M. H. Halstead, Elements of Software Science. New York, NY, USA: Elsevier, 1977. Available from: <https://books.google.com/books?id=rRipAQAAMAAJ>
- [10] R. C. Martin, Agile Software Development: Principles, Patterns, and Practices. Upper Saddle River, NJ, USA: Prentice Hall, 2002. Available from: <https://search.worldcat.org/title/Agile-software-development-%3A-principles-patterns-and-practices/oclc/224957893>
- [11] M. Fowler, Refactoring: Improving the Design of Existing Code, 2nd ed. Boston, MA, USA: Addison-Wesley, 2018. Available from: <https://www.pearson.com/en-us/subject-catalog/p/refactoring-improving-the-design-of-existing-code/P200000000254/9780134757599>
- [12] T. Mens and T. Tourwé, "A survey of software refactoring," IEEE Transactions on Software Engineering, vol. 30, no. 2, pp. 126–139, Feb. 2004. Available from: <https://doi.org/10.1109/TSE.2004.1265817>
- [13] V. R. Basili, G. Caldiera, and H. D. Rombach, "The Goal Question Metric approach," in Encyclopedia of Software Engineering, vol. 1. New York, NY, USA: Wiley, 1994, pp. 528–532. Available from: <https://www.ecs.csun.edu/~rlingard/COMP587/gqm.pdf>
- [14] B. Kitchenham and S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, EBSE Technical Report EBSE-2007-01, Keele University and Durham University, 2007. Available from: <https://tinyurl.com/zd2vf638>
- [15] Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in Software Engineering, 2nd ed. Cham, Switzerland: Springer, 2012. Available from: <https://doi.org/10.1007/978-3-642-29044-2>
- [16] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd ed. Boston, MA, USA: PWS Publishing, 1997. Available from: <https://search.worldcat.org/title/Software-metrics-%3A-a-rigorous-and-practical-approach/oclc/1055171425>
- [17] L. C. Briand, J. W. Daly, and J. K. Wüst, "A unified framework for coupling measurement in object-oriented systems," IEEE Transactions on Software Engineering, vol. 25, no. 1, pp. 91–121, Jan./Feb. 1999. Available from: <https://doi.org/10.1109/32.748920>
- [18] L. C. Briand, J. W. Daly, and J. K. Wüst, "A unified framework for cohesion measurement in object-oriented systems," Empirical Software Engineering, vol. 3, no. 1, pp. 65–117, 1998. Available from: <https://doi.org/10.1023/A:1009783721306>

- [19] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, 1993. Available from: [https://doi.org/10.1016/0164-1212\(93\)90077-B](https://doi.org/10.1016/0164-1212(93)90077-B)
- [20] ISO/IEC/IEEE 29119-1:2022, *Software and Systems Engineering—Software Testing—Part 1: General Concepts*. Geneva, Switzerland: International Organization for Standardization, 2022. Available from: <https://www.iso.org/standard/81291.html>