

Intelligent Test Case Scheduling Using Multi-Criteria Decision Making for Human–Machine Interface Testing

Gang Zheng

Juxing Xiang

Jun Xu

Qilin Li

Guida Tang

19158811523@163.com

Sichuan Technology and Business University

Research Article

Keywords: software quality assurance, utility-based evaluation, smart manufacturing, factory acceptance testing, test prioritization

Posted Date: October 9th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11137088/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Intelligent Test Case Scheduling Using Multi-Criteria Decision Making for Human–Machine Interface Testing

Gang Zheng, Juxing Xiang, Jun Xu, Qilin Li, Guida Tang

Sichuan Technology and Business University, Chengdu, Sichuan, China

Corresponding Author:

Guida Tang¹

No. 65 Tuanjie College Street, Pidu District, Chengdu, Sichuan, 611754, China

Email address: 19158811523@163.com

Gang Zheng²

No. 65 Tuanjie College Street, Pidu District, Chengdu, Sichuan, 611754, China

Email address: 13008150705@163.com

Abstract

Factory testing of industrial human–machine interface (HMI) system is typically conducted under stringent constraints: a fixed and non-extendable time budget, limited test resources, and zero tolerance for undetected critical failures. Under these conditions, the order and selection of test cases constitute a quality-risk decision rather than a mere efficiency concern. In practice, however, test case selection relies largely on engineers' experience and ad-hoc checklists, resulting in inconsistent coverage, inefficient resource use, and poor reproducibility across releases. To address this gap, we propose a Time-Weighted Utility Scoring Model (TWUSM) for automated test case prioritization and selection. The model evaluates each test case along three dimensions (risk impact on system safety, functional importance, and testing cost) and combines them into a single utility score that is updated with historical execution feedback. Test cases are then assigned to priority tiers: those above a predefined threshold are deterministically included as mandatory tests, while the remainder are admitted by probabilistic sampling with probability proportional to their normalized scores. This hybrid scheme enforces quality first by guaranteeing complete verification of critical functionalities, and then, subject to this constraint, maximizes fault detection per unit of testing time. Evaluated on an industrial HMI system, TWUSM achieves 100% detection of critical failures and an APFD of 0.9987. Compared with full test-suite execution (425.9 s), it reduces testing time to 174.9 s (–58.9%) without any loss of failure coverage. The results show that utility-based prioritization can provide a reproducible, resource-aware basis for inspection planning in safety-critical industrial software, assuring product quality while substantially reducing test duration.

Key Word: software quality assurance; utility-based evaluation; smart manufacturing; factory acceptance testing; test prioritization

I. Introduction

Currently, test case selection relies primarily on two conventional practices in factory testing of industrial human–machine interface (HMI) systems: full test-suite execution, which runs the entire set of test cases on every unit, and fixed filtering, which repeatedly executes a predefined subset of critical test cases derived from engineers' experience^[1]. Both practices have notable limitations. Full test-suite execution incurs substantial resource overhead by applying tests whose outcomes are stable across units. This overhead is particularly acute in high-volume production with short cycle times, where even marginal reductions in test duration translate into significant cost savings^[2]. Fixed filtering, while reducing the test scope, depends on subjective and incomplete human knowledge, which introduces two risks. First, high-risk failure modes outside the engineers' experience may be omitted, allowing critical failures to escape detection at the factory stage. Second, experience-based rules lack quantitative criteria and therefore adapt poorly to changes in product configuration, firmware revision, and test-equipment constraints^[3–5]. Critically, neither practice treats test selection as a quality-risk decision under a fixed, non-extendable time budget: the former ignores efficiency, and the latter provides no guarantee of coverage consistency or reproducibility across units and batches.

Recently, research on optimization methods for test cases has been relatively limited, particularly for systems that integrate both software and hardware; most existing studies focus instead on pure software testing^[6]. Reference^[7] proposes a reactive GRASP meta-heuristic algorithm for test case prioritization, which achieves excellent coverage performance with acceptable time overhead and high result stability. Reference^[8] presents a model-specific coverage criterion combined with a cost-aware, search-based test generation method. To evaluate the effectiveness of the proposed criteria and method, the authors introduce a mutation analysis technique at the model level and employ various failure detection methods to assess the quality of generated tests. Reference^[9] investigates a requirement clustering method integrated with traditional code analysis information, demonstrating that incorporating requirement

information into test case prioritization yields significant benefits. Reference ^[10] proposes a system-level test case prioritization model derived from software requirement specifications, aiming to improve the detection rate of severe failures and ultimately deliver high-quality, low-cost software with high user satisfaction. This model prioritizes system-level test cases based on six factors: customer priority, requirement volatility, implementation complexity, completeness, traceability, and failure impact. Reference ^[11] explores the application of greedy algorithms in automated testing, with a focus on the selection of fitness metrics, characterization of search space landscapes, and determination of optimal search strategies. Reference ^[12] introduces a novel supervised machine learning-based technique for manual system-level regression test prioritization, which significantly improves failure detection rates. In contrast, this work targets industrial-grade Human-Machine Interface (HMI) systems, which are characterized by stable software versions and a mature, fixed test suite accumulated over long-term operation. In such a context, where rapid adaptation and complex computation are less critical, lightweight and efficient algorithms are not only sufficient but also particularly suitable for practical and sustainable regression test optimization.

In the research and application of utility scoring models, this model has already been successfully applied in various fields such as finance, healthcare, construction, and the internet. Reference ^[13] proposed a novel method for selecting probability estimation models, which enables the comparison of market transactions under evolving transaction probability hypotheses over time. Reference ^[14] conducted an empirical assessment of a new recommendation system in terms of utility and willingness-to-pay estimation, and by comparing it with systems based on linear utility functions, derived pricing strategies for profit maximization. Reference ^[15] developed a scoring system and carried out a comparative study of different scoring strategies, aiming to indirectly compute utility values using the prostate-specific utility scale, with applications in clinical practice and research settings. Reference ^[16] established a scoring formula based on responses from 194 subjects, which employs a multiplicative multi-attribute utility function to help researchers record health states and assign corresponding preference scores. Reference ^[17] applied utility functions together with principal component analysis to perform risk assessment and prioritization of selected drug active compounds and investigated the impact of assigning equal versus unequal weights to different criteria. Reference ^[18] identified key parameters affecting user satisfaction in the construction domain and designed a weighting system that reflects both user requirements and the perspectives of construction parties, thereby establishing a scoring system based on a utility model. Reference ^[19] proposed a new service recommendation model based on the fuzzy analytic hierarchy process (FAHP) and ordinal utility functions, determining the optimal service by calculating the ordinal utility of candidate services. Reference ^[20] introduced a utility function-based strategy that eliminates the rank reversal phenomenon and significantly improves the quality of web ranking by comprehensively considering application and user requirements. Reference ^[21] proposed a utility function that allows group multi-criteria decision-making problems to be modeled as game-theoretic problems. The advantage of using game theory in this context is that it enables strategic evaluation of conflicts among decision-makers. It can be seen that utility scoring models have been widely applied across multiple industries. However, unfortunately, there are still few reports on their application in the field of automated testing for industrial equipment. Reference ^[22] proposed a test case prioritization technique based on a case-based machine learning algorithm for ranking test cases. This method acquires relative priority information through user-provided pairwise comparisons of test cases. During the iterative refinement of the test case order, the user input is integrated with multiple priority metrics. The model ranks system test cases based on six factors. These include customer priority, requirement volatility, implementation complexity, completeness, traceability, and failure impact. The work in Reference ^[22] provides certain guidance for the research presented in this paper. However, there are significant differences between the application scenarios considered in this study and those in Reference ^[22]. The industrial human-machine interface (HMI) systems addressed in this work are characterized by broad deployment, relatively low requirement volatility, and high safety-criticality. These characteristics lead to different sets of prioritization factors. Moreover, the model in Reference ^[22] does not incorporate testing cost as a consideration in the algorithm design, which limits its practicality in real-world production environments.

To address the limitations identified above, namely low inspection efficiency, excessive test duration, and reliance on experience-driven case selection in the factory testing of industrial HMI devices, this paper proposes the Time-Weighted Utility Scoring Model (TWUSM) for automated test case prioritization and selection. The model constructs a single utility function that quantitatively evaluates each test case along three dimensions: risk impact on system safety, functional importance, and execution cost, with weights updated from historical execution feedback. Based on the resulting ranking, test cases are assigned to priority tiers: those above a predefined threshold are deterministically admitted as mandatory tests, while lower-tier cases are selected by probabilistic sampling proportional to their normalized scores. Experiments on an industrial HMI system show that TWUSM reduces testing time from 425.9 s to 174.9 s (−58.9%) while achieving an APFD of 0.9987 and 100% detection of critical failures. By replacing ad-hoc human judgment with a documented, score-based rule, the method improves both the objectivity and the reproducibility of the factory inspection process.

II. Method Design

A. Overall Design of the Testing System

In traditional testing of industrial HMI devices, test case selection is entirely dependent on the subjective experience of a test engineer. The engineer makes binary decisions, whether to test or skip each test case, based solely on personal judgment, without any multi-dimensional data evaluation, quantitative assessment of functional importance, or risk analysis. The selected test cases are manually compiled into a fixed test suite, which is then imported into the automated testing system. The testing equipment executes the test cases sequentially according to their predefined order in the suite, performs the tests on the target HMI, and finally aggregates the results into a test report upon completion. When a new version of the HMI is released, the test case update process is repeated. Despite the accumulation of numerous test reports, the engineer is unable to systematically analyze historical results, relying only on manual cognitive effort, subjective understanding of version changes, and qualitative interpretation of past outcomes. Consequently, decisions on adding, removing, or modifying test cases remain simplistic and experience driven. The updated test suite is then re-imported for automated execution on the new version. Throughout this process, test case selection lacks data-driven support and remains confined to the engineer's binary decisions. This leads to suboptimal test coverage, with critical functional paths prone to omission and low-risk scenarios unnecessarily retested. As a result, testing strategies fail to achieve dynamic optimization and cannot evolve in a scientific or sustainable manner.

To address the limitations in traditional testing of industrial HMI devices, namely, the heavy reliance on manual experience, the lack of quantitative criteria in decision-making, and the static, inflexible test strategy, this paper proposes an algorithm for automated test case selection based on a multi-dimensional utility scoring model. The algorithm integrates multiple data dimensions, including functional importance, risk impact on system safety, testing cost, and historical pass rate, to quantitatively score and dynamically prioritize test cases, thereby enabling a shift from experience-driven to data-driven testing. The algorithm replaces the manual test case selection step in the conventional workflow, where a test engineer subjectively decides whether to test or skip each case. Specifically, it eliminates the need for human binary judgment by automatically computing a comprehensive utility score for each test case and intelligently generating the set of test cases to be executed based on these scores. The resulting high-value test suite is then imported into the existing automated testing system for execution. Without modifying the current test execution framework, the proposed method fundamentally optimizes the test case selection mechanism, enabling dynamic test coverage and scientific allocation of testing resources.

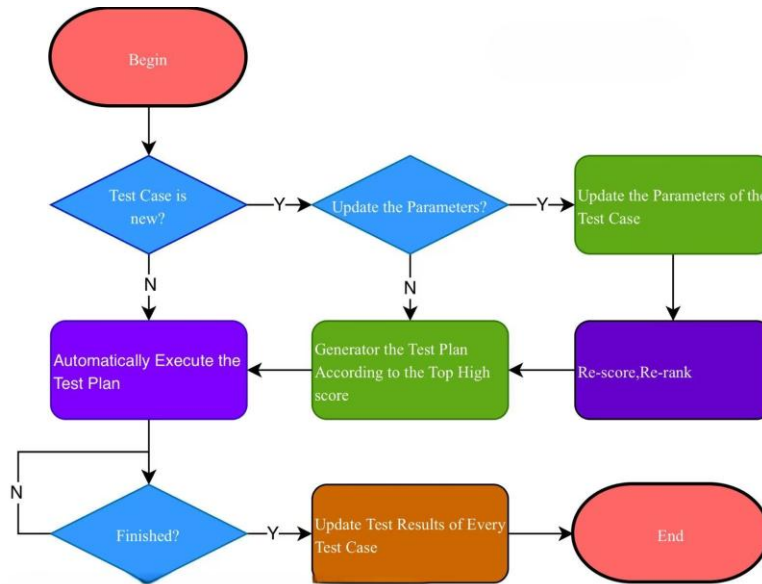


Figure 1. Workflow diagram of the proposed automated test case execution and optimization process based on the TWUSM model.

Figure 1 shows the complete workflow of the proposed automated test case execution and optimization process. The flow starts at the "Start" node, where the system first determines whether a new test batch is initiated. If so, it checks whether the factor parameters of the test cases require updating. If an update is needed, the parameters are revised, and all test cases are re-scored and re-prioritized using the utility function model. Otherwise, the existing scores are retained. Subsequently, test cases with higher utility scores are selected to form the automated test suite, which defines the set of test cases to be executed. The system then enters the automated execution phase, where test cases are performed sequentially or in parallel according to the testing environment. During execution, the system continuously checks whether the current test suite has been fully completed. If not, it proceeds with the next test case; otherwise, it updates the historical pass rate data of each test case based on the latest execution outcomes. This closed-

loop process concludes with the "End" node, but the updated pass rates are stored for use in future test batches. By feeding execution results back into the scoring model, the system enables dynamic test case optimization across iterations, thereby improving product inspection efficiency and quality over time.

B. Study on Time-Weighted Utility Scoring Model

This paper assumes that the decision to execute a test case is determined by three key factors—the risk impact of its associated function on system safety, the testing cost, and the functional importance. Based on this assumption, a utility scoring model is established to quantitatively evaluate and prioritize test cases, as is shown in Equation (1),

$$\text{Score}(i) = w_1 \cdot R(i) + w_2 \cdot I(i) - w_3 \cdot C(i), \quad (1)$$

where $R(i)$ denotes the risk function, with the risk factor representing the failure probability of the functional module corresponding to test case; $I(i)$ is the importance function, with the importance factor reflecting the criticality of the functionality covered by the test case; $C(i)$ denotes the cost function, with this factor indicating the resource consumption score required to execute the test case, and all the three metrics $R(i)$, $I(i)$, and $C(i)$ are scored on a percentage scale ranging from 0 to 100; and w_1, w_2, w_3 are the weighting coefficients for the respective functions, all non-negative and satisfying the constraint given in Equation (2),

$$w_1 + w_2 + w_3 = 1; \quad (2)$$

where $R(i)$ in Equation (1) measures the historical failure probability of the functional module corresponding to test case i , and the function selected in this paper is as shown in Equation (3),

$$R(i) = (1 - \text{pass}(i))^\alpha, \quad (3)$$

where $\text{pass}(i)$ is the historical pass rate of test case i , which is automatically updated after each test case execution; α is the amplification coefficient used to adjust the sensitivity to test case failures. A value of $0 < \alpha < 1$ non-linearly amplifies the risk score of test cases with low historical pass rates, thereby prioritizing potentially unstable modules. Conversely, $\alpha > 1$ would compress the risk scores, reducing the differentiation between high-risk and low-risk test cases. In this study, we set $\alpha = 0.045$ to strongly amplify the risk of less stable test cases. $I(i)$ in Equation (1) is used to reflect the criticality of the functionality covered by the test case within the system, and the function selected in this paper is as shown in Equation (4),

$$I(i) = w_p \cdot P(i) + w_s \cdot S(i) + w_f \cdot F(i) + w_c \cdot \text{Comp}(i), \quad (4)$$

where $P(i)$, $S(i)$, $F(i)$, and $\text{Comp}(i)$ denote the normalized scores of the respective metrics, with $P(i)$ representing the functional priority score used to determine whether the test case belongs to a core business process, $S(i)$ representing the module stability score quantified based on historical failure frequency, $F(i)$ representing the user access frequency score used to determine whether it belongs to a high-usage path, and $\text{Comp}(i)$ representing the security and compliance score used to determine whether the test case involves sensitive operations or regulatory compliance requirements, and the weighting coefficients must satisfy the constraint given in Equation (5),

$$w_p + w_s + w_f + w_c = 1, \quad (5)$$

where w_p, w_s, w_f, w_c are the weighting coefficients corresponding to the respective metrics, all non-negative and satisfying the aforementioned constraint.

To transform the raw utility score into a valid probability for the sampling process, we apply a normalization step. The raw score calculated in Equation (1) is normalized to the range $[0, 1]$ using Min-Max normalization, ensuring that the resulting value can serve as a selection probability, as is shown in Equation (6),

$$P_{\text{base}}(i) = \frac{\text{Score}(i) - \min(\text{Score})}{\max(\text{Score}) - \min(\text{Score})} \quad (6)$$

where $P_{\text{base}}(i)$ denotes the normalized base selection probability, and $\min(\text{Score})$ and $\max(\text{Score})$ represent the minimum and maximum raw utility scores within the current test case set, respectively. Furthermore, in this paper, the computed result of $P_{\text{base}}(i)$ is used as the selection probability for sampling, and let X_i denote whether test case i is selected for execution, then X_i must satisfy the constraint given in Equation (7),

$$X_i = \begin{cases} 1, & \text{selected for testing with probability } P_{\text{base}}(i) \\ 0, & \text{not selected with probability } 1 - P_{\text{base}}(i) \end{cases}, \quad (7)$$

after each batch of testing, the historical pass rate $\text{pass}(i)$ of each test case is automatically updated, and $R(i)$ and $P_{\text{base}}(i)$ are recalculated.

C. Evaluation index design

a. Metric of Average Percentage of Failures Detected (APFD)

APFD is the most classical evaluation metric in the field of test case prioritization, used to measure the average speed at which a test case sequence detects failures. Its calculation formula is shown in Equation (8),

$$\text{APFD} = 1 - \text{average of } \left(\frac{T_f(i)}{n}\right), \quad (8)$$

where n is the total number of test cases; $T_f(i)$ is the number of test cases that have been executed when the i -th failure is first detected (counting starts from 1).

b. Metric of Top-N

Top-N detection rate is a metric that measures the proportion of failures detected within the top N test cases ranked by utility score. Its definition is given in Equation (9),

$$\text{Top} - N = \frac{D_N}{D_T} \times 100\%, \quad (9)$$

where, D_N is Number of failures detected by the first N test cases in the prioritized sequence. D_T is total number of failures present in the system under test.

c. Metric of Precision@K

Precision@K measures the proportion of test cases among the top K prioritized test cases that effectively detect products with real failures. Its mathematical definition is given in Equation (10),

$$\text{Precision@K} = \frac{N_{\text{effective}}}{K} \times 100\%, \quad (10)$$

where, $N_{\text{effective}}$ denotes the number of test cases within the top K that successfully detect products with real failures (i.e., test cases that fail and whose corresponding products are confirmed to have manufacturing failures), and K is the number of top-ranked test cases considered.

d. Metric of Recall@K

Recall@K measures the proportion of the total number of defective products detected with regard to the first K test cases out of the total number of all defective products. Its mathematical definition is given in Equation (11),

$$\text{Recall@K} = \frac{D_{\text{detected@K}}}{D_{\text{total}}} \times 100\%, \quad (11)$$

where, $D_{\text{detected@K}}$ denotes the number of distinct defective products detected by at least one test case among the top K , and D_{total} is the total number of defective products in the test batch.

To address the concern regarding the alignment between the task setting and evaluation metrics, we formally define our problem as a "Budgeted Test Case Selection" task. While the evaluation employs the Average Percentage of Failures Detected (APFD) metric—commonly used in prioritization studies—it is utilized here to assess the ranking efficacy of the underlying scoring model. Specifically, a high APFD indicates that the utility scores effectively concentrate on high-risk failures at the top of the list. However, the actual execution strategy follows a Selection paradigm constrained by the fixed time budget of the industrial production line. After ranking, we apply a threshold-based selection (as described in Section B) to form the final test suite. Consequently, the primary validation of our method relies on selection-specific metrics: Precision@K and Recall@K. These metrics evaluate the effectiveness of the selected subset in detecting failures within the strict time constraints of the manufacturing environment, ensuring the method is optimized for practical deployment rather than theoretical ranking alone.

The algorithm proposed in this paper was tested over six thousand times on industrial HMI systems. To enable a systematic analysis, the 6,000 independent test executions were partitioned into five groups (Group 1 to Group 5), each comprising 1,200 executions. Group 1 was designated for the in-depth analysis presented in the following subsections, while Groups 2–5 were randomly assigned from the remaining executions to ensure statistical independence. In our experimental setup, each complete test execution consists of 52 distinct test cases. The reported results are based on 6,000 independent runs of this full test suite, partitioned into five groups of 1,200 runs each, where K and N are both set to 1,200. This assignment mitigates sequence-induced bias and ensures statistical comparability across groups. We first performed an in-depth analysis in Group 1, examining both process-level metrics and outcome metrics. The process metrics included testing time, $R(i)$, $I(i)$, $C(i)$ and $\text{Score}(i)$. The outcome metrics analyzed included APFD, Top-1200 detection rate, Precision@1200, Recall@1200, and total execution time, providing a comprehensive understanding of the algorithm's behavior. To investigate the consistency and potential variability across different testing executions, we then compared the outcome metrics from Group 1 with those from the remaining four groups (Groups 2–5). The scoring focused on key outcome metrics including the overall test score, APFD, Top-1200 detection rate, Precision@1200, and Recall@1200. By comparing the results across all five groups—where Group 1, designated for process analysis, served as the baseline and Groups 2–5, randomly assigned for stability verification, contributed to outcome comparison—we were able to examine both the internal dynamics and the cross-execution stability of the algorithm. In addition, to facilitate the evaluation of the metrics APFD, Top-1200 detection rate, Precision@1200, and Recall@1200, this paper also provides their formal definitions.

III. Results and Analysis

A. Experimental Parameter Settings

To ensure the objectivity and practicality of the evaluation, the weight parameters in the TWUSM model are not arbitrarily assigned. Instead, they are derived from domain expert knowledge. Specifically, we invited five senior test engineers from the production environment to determine the weights. Each engineer was asked to rate the relative importance of the four factors in Equation (5) and the three factors in Equation (1) on a scale of 1 to 10. The final weight values used in this experiment were calculated by normalizing the average scores provided by these experts. This approach ensures that the model reflects the actual priorities of the industrial testing process rather than theoretical assumptions. The specific values determined by this expert assessment process are summarized in Table 1.

Table 1: Weight values for Equation (1) and Equation (5) determined by expert assessment.

Factor Category	Symbol	Component Name	Final Weight
Equation (1) Weights	w_1	Risk Weight	0.1
	w_2	Importance Weight	0.5
	w_3	Cost Weight	0.4
Equation (5) Weights	w_p	P(i)	0.3
	w_s	S(i)	0.2
	w_f	F(i)	0.2
	w_c	Comp(i)	0.3

In addition, Regarding the risk amplification coefficient α in Equation (3), we set $\alpha=0.045$ in this experiment. This setting is chosen to non-linearly amplify the risk score of test cases with low historical pass rates, thereby prioritizing potentially unstable areas of the software, which aligns with the zero-tolerance safety requirement in industrial testing.

B. Results and Analysis of Group 1

a. Total execution time of test cases ordered by the TWUSM algorithm

The test cases are ordered by the TWUSM algorithm, and their cumulative execution times are recorded. As the number of executed test cases increases, the total testing time grows progressively, executing all test cases for each HMI under test results in a total time of 425.9 seconds. A test engineer assigned percentage-based scores to the three metrics R(i), I(i), and C(i) and provided the analytical assessment.

b. The Score of R(i)

Figure 2(a) illustrates the distribution of risk factor scores R(i) across 52 test cases. The horizontal axis represents the test case indices ranked in descending order of their final priority scores Score(i), while the vertical axis shows the corresponding R(i) values. As shown in the figure, the majority of test cases have a risk score of zero, indicating low inherent risk. However, a small number of test cases exhibit significantly higher R(i) values, marked as red dots. These include test cases ID:28 and ID:53 (R(i)=8.25), ID:36 (R(i)=7.68), ID:2 (R(i)=8.70), ID:44 (R(i)=8.3), ID:45 and ID:46 (R(i)=8.25), ID:11 (R(i)=8.18), and ID:7 (R(i)=7.44). Notably, the highest risk score is observed in test case ID:2 (R(i)=8.70). These high-risk test cases are sparsely distributed across the ranking, suggesting that they represent critical points requiring prioritized testing.

c. The Score of I(i)

Figure 2(b) and Figure 2(c) present a detailed breakdown of the importance of scoring components for the top 26 and bottom 26 test cases, respectively, ranked by their final scores. Each stacked bar represents a test case, with the total height indicating the overall importance score, which is computed as a weighted average of five key dimensions: Security/Compliance Score (blue), User Access Frequency Score (purple), Module Stability Score (green), Function Priority Score (orange), and the final Importance Factor Score (red). In Figure 2(b), the higher-ranked test cases exhibit balanced and substantial contributions across multiple dimensions, particularly in Security/Compliance, Function Priority, and User Access Frequency, indicating their high functional and operational significance. Notably, some high-priority test cases, such as ID 54 and ID 16, achieve high overall scores due to strong performance across all dimensions, especially full or near-full scores in Function Priority, Security/Compliance, and User Access Frequency. In contrast, Figure 2(c) reveals that lower-ranked test cases are often dominated by the Module Stability Score (green), frequently reaching the maximum value of 100.0, while other dimensions like Function Priority and User Access Frequency remain low, often below 60.0, as seen in test cases such as ID 48 and ID 30. This suggests that despite being stable modules, these test cases are less critical in terms of user impact or business functionality.

d. The Score of C(i)

Figure 2(d) presents the cost factor scoring results for all 52 test cases, calculated using the utility function C(i), where C(i) represents the cost associated with executing test case i. The horizontal axis shows the test case ID. As illustrated, the cost scores vary significantly across test cases, with several sharp peaks indicating high-cost test cases—particularly around IDs 48, 13, 26, 47, and 40—suggesting these cases require substantial resources or time to execute. In contrast, many test cases exhibit relatively low-cost scores, clustering around the lower end of the scale, implying they are less resource intensive. This distribution highlights the heterogeneity in execution costs among test cases and underscores the importance of incorporating cost factors into test prioritization strategies.

e. The Rank of Test Case Score

Figure 2(e) presents the final scoring results of 52 test cases ranked in descending order based on the algorithm proposed in this paper. The ranking reflects the priority for testing, where higher scores indicate a greater need for testing. The horizontal axis represents the test case IDs, and the vertical axis shows the final scores on a percentage scale. As shown in the figure, the test cases are arranged in descending order of their scores, with the highest score being 86.2 (with regard to test case ID 16) and the lowest score being 8.0 (with regard to test case ID 48). Based on the score distribution observed in this Chengdu production line experiment, a threshold of 65% is established. Test cases with scores above this threshold are identified as high-priority mandatory tests. Reference lines at 70 and 100 points are also marked in the figure to facilitate a visual comparison of the relative performance levels of the test cases.

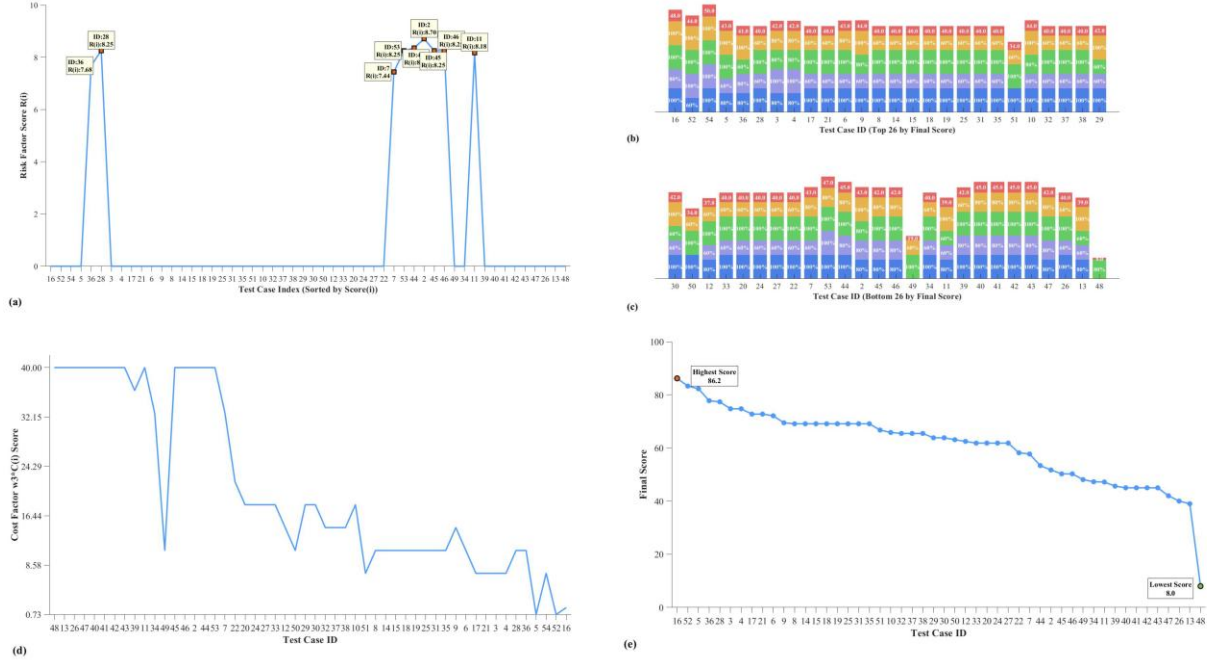


Figure 2. Analysis of scoring components for 52 test cases under the TWUSM model: (a) distribution of risk factor scores...; (b) importance scoring breakdown for the top 26 test cases; (c) importance scoring breakdown for the bottom 26 test cases; (d) distribution of cost factor scores; (e) final utility scores ranked in descending order.

f. Comparative Analysis of APFD

Figure 3(a) shows a comparison of the ranking results obtained by the utility function-based weighted scoring model proposed in this paper with those of random ordering, ordering by the number of failures, and the baseline of executing the full test suite. As can be seen from the calculation results in Figure 3(a), the APFD of reverse ordering is the lowest, at 0.5287. The APFD of the TWUSM algorithm is 0.9987, which is approximately 8.53% higher than that of random ordering (0.9134), demonstrating a substantial improvement in the APFD performance metric. On the other hand, the APFD of sorting purely by the number of failures is 0.9988, marginally higher than that of the proposed algorithm. However, this method considers only the number of failures and failures to incorporate practical application factors such as functional importance and the security impact of individual test cases. As a result, it lacks executability and does not align with real-world production scenarios.

g. Top-1200 Analysis

Figure 3(b) shows the Top-1200 detection rate curve obtained by the TWUSM algorithm, illustrating the failure detection rate under different numbers of test cases. As can be seen from Figure 3(b), once 16 test cases are executed, the failure detection rate for the HMI under test exceeds 95%, requiring 133.7 seconds of testing time; once 20 test cases are executed, the detection rate exceeds 99%, requiring 165.9 seconds; and once 23 test cases are executed, the detection rate reaches 100%, with a total testing time of 174.9 seconds. Analysis of this data reveals that many test cases have a 100% historical pass rate, and these cases can be sampled with low probability or even omitted during routine production testing. On the other hand, based on the curve's trend, engineers can balance detection coverage against testing time to determine an appropriate number of test cases to execute. Overall, the TWUSM algorithm reduces the required number of test cases, thereby saving substantial testing time while maintaining high failure detection rates.

h. Precision@1200 Analysis

Figure 3(c) shows the Precision@1200 curve obtained by the TWUSM algorithm, illustrating the precision of the HMI under test at different numbers of executed test cases. As can be seen from Figure 3(c), once 12 test cases are

executed, the precision rate for the HMI under test exceeds 90%, with a required testing time of 111.5 seconds; once 16 test cases are executed, the precision rate exceeds 95%, requiring 133.7 seconds; and once 23 test cases are executed, the precision rate reaches 100%, with a total testing time of 174.9 seconds. Analysis of this data reveals that over half of the test cases have a 100% pass rate, indicating that these cases could be sampled at low probability or even excluded from routine production testing without compromising the overall quality assurance process. On the other hand, based on the trend indicated by the curve, engineers can balance precision against testing time to find an optimal number of test cases to execute. This involves identifying a point where further increases in the number of executed test cases yield diminishing returns in terms of precision improvement relative to the additional testing time required. Overall, the algorithm designed in this paper significantly reduces the necessary number of test cases, thereby saving considerable testing time while ensuring high detection rates and maintaining precision. This method optimizes both efficiency and effectiveness in the testing process.

i. Recall@1200 Analysis

Figure 3(d) shows the Recall@1200 curve obtained by the TWUSM algorithm, illustrating the recall rate of the HMI under test at different numbers of executed test cases. As can be seen from Figure 3(d), once 12 test cases are executed, the recall rate for the HMI under test exceeds 90%, with a required testing time of 111.5 seconds; once 15 test cases are executed, the recall rate exceeds 95%, requiring 123.7 seconds; and once 23 test cases are executed, the recall rate reaches 100%, with a total testing time of 174.9 seconds. Analysis of this data reveals that over half of the test cases have a 100% pass rate, indicating that these cases could potentially be sampled at low probability or even excluded from routine production testing without compromising overall quality assurance. On the other hand, based on the data provided by the curve, engineers can balance recall against testing time to find an optimal number of test cases to execute. This involves identifying a point where the increase in recall becomes less significant relative to the additional testing time required. By doing so, they can ensure high detection rates while minimizing unnecessary testing duration. Overall, the algorithm designed in this paper significantly reduces the necessary number of test cases, thereby saving considerable testing time while maintaining high recall rates. This method optimizes both efficiency and effectiveness in the regression testing process for industrial HMI systems.

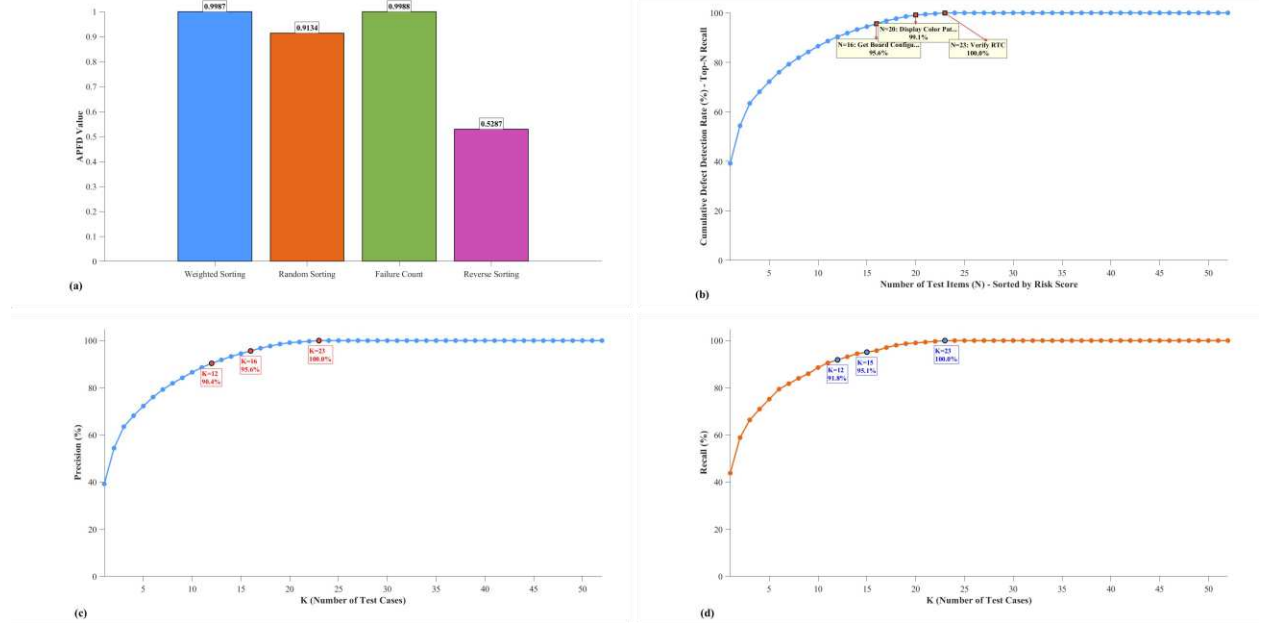


Figure 3. Performance evaluation of the TWUSM algorithm on the industrial HMI system: (a) APFD comparison among different sorting methods; (b) Top-1200 cumulative defect detection rate curve; (c) Precision@1200 curve; (d) Recall@1200 curve.

j. Ablation study-comparison with single-factor baselines

To further validate the effectiveness of the proposed TWUSM and address the concern regarding baseline comparison, we conducted an ablation study. We compared TWUSM against three simplified single-factor variants:

- (1) Risk-Only: A baseline that prioritizes test cases solely based on their risk scores, ignoring importance and cost.
- (2) Importance-Only: A baseline that ranks test cases purely by their functional importance, neglecting risk and execution cost.
- (3) Cost-Only: A baseline that prioritizes test cases with the lowest execution cost, maximizing the number of executed cases but ignoring failure detection potential.

The comparison was performed using the APFD and Precision@1200 metrics to evaluate the performance degradation when removing specific factors from the model.

As shown in Table 2, the ablation study results based on Group 1 data provide compelling evidence of the inherent limitations of single-factor strategies in test case prioritization. Specifically, relying on a single dimension often leads to suboptimal performance, failing to achieve the balance between detection speed and resource efficiency realized by TWUSM through multi-factor synergy. In detail, the Risk-Only method achieves an APFD of 0.9628, which is lower than TWUSM's 0.9987. More critically, the third column indicates the number of test cases required to reach the target precision level established by our algorithm. For the Risk-Only strategy, this value is 35.8, meaning it requires executing approximately 7.5 times more test cases than TWUSM (which requires only 4.8) to achieve the same confidence level. This severe inefficiency suggests that while risk scores capture failure likelihood, ignoring execution cost and functional importance leads to substantial resource waste. Similarly, the Cost-Only method (APFD = 0.9602; cases required = 27.2) prioritizes cheap test cases but lacks the ability to detect defects as efficiently as the proposed method. The Importance-Only method performs the worst in terms of detection speed (APFD = 0.9336), demonstrating that static importance metrics alone cannot effectively guide dynamic prioritization. In contrast, TWUSM achieves the best overall performance with the highest APFD (0.9987) and the lowest execution requirement (4.8). This combination proves that by integrating risk, cost, and importance, TWUSM not only accelerates failure detection but also minimizes the testing effort required to reach a specific confidence level. It successfully breaks the trade-offs inherent in single-factor baselines, validating the necessity and superiority of the multi-factor fusion mechanism.

Table 2: Ablation Study Results (TWUSM vs. Single-Factor Baselines)

Method	APFD	Test Case Number for TWUSM Precision
TWUSM	0.9987	4.8
Risk-Only	0.9628	35.8
Importance-Only	0.9336	9.8
Cost-Only	0.9602	27.2

k. Comparison among 5 groups

To evaluate the stability of the proposed algorithm when faced with scoring results from different test engineers under the same evaluation criteria, this paper constructs five groups of test data for comparative analysis. Specifically, the first group consists of the 1,200 test executions previously used to validate the correctness of the algorithm. The remaining 4,800 records are randomly partitioned into four additional groups, resulting in a total of five groups. These five groups are then evaluated using several metrics, including the overall test score, APFD, Top-1200 detection rate, Precision@1200, and Recall@1200. To evaluate the reasonableness of the aforementioned metrics, this paper first conducted a statistical analysis of the scores assigned by five engineers to 52 test cases across three dimensions: importance, risk, and cost. The results are shown in Figures 4(a) to Figures 4(d).

Figure 4(a) presents the evaluation results of the 52 test cases along the risk factor dimension. The blue line represents the mean scores assigned independently by five engineers, while the orange triangular markers indicate the corresponding standard deviations. As shown in the standard deviation distribution, the vast majority of test cases exhibit standard deviations close to zero, indicating a high level of agreement among engineers regarding the risk assessment of these test cases. Only a small number of test cases show relatively higher standard deviations, with the maximum value approximately 4.0 and the minimum approaching 0. This demonstrates that the engineers' ratings on the risk factor are generally highly consistent. The limited discrepancies occur primarily in specific complex or boundary-case scenarios, and the magnitude of variation falls within the range of reasonable cognitive variance expected in expert assessments in software engineering practice. Consequently, these minor inconsistencies do not undermine the overall reliability of the scoring results.

Figure 4(b) presents the evaluation results of the 52 test cases along with the importance factor dimension. The blue line represents the mean scores assigned independently by five engineers, and the orange triangular markers indicate the corresponding standard deviations. As shown in the standard deviation distribution, the vast majority of test cases have standard deviations below 3.0, with most falling within the range of 2.0 to 3.0. The minimum standard deviation is close to 0, while the maximum is approximately 5.5. This indicates that the five engineers demonstrated high consistency in their importance ratings for most test cases, with only a few exhibiting noticeable disagreements. Overall, more than 90% of the test cases have standard deviations below 4.0, demonstrating good inter-rater consistency. The magnitude of variation falls within the normal range of cognitive variance expected among software engineering professionals when assessing product quality attributes and therefore does not compromise the overall reliability of the evaluation results.

Figure 4(c) presents the evaluation results of the 52 test cases along the cost factor dimension. The blue line represents the mean scores assigned independently by five engineers, and the orange triangular markers indicate the corresponding standard deviations. As shown in the standard deviation distribution, the vast majority of test cases have standard deviations below 3.0, with most falling within the range of 1.0 to 3.0. The minimum standard deviation is close to 0, while the maximum is approximately 4.5. This indicates that the five engineers demonstrated high

consistency in their cost assessments for most test cases, with only a few exhibiting noticeable disagreements. Overall, more than 85% of the test cases have standard deviations below 3.0, demonstrating good inter-rater consistency. The magnitude of variation falls within the normal range of cognitive variance expected among software engineering professionals when assessing product quality attributes and therefore does not compromise the overall reliability of the evaluation results.

Figure 4(d) presents the mean and standard deviation of the composite scores for 52 test cases. These scores were calculated based on independent ratings from five groups of engineers on three dimensions (importance, cost, and risk) using the weighting scheme proposed in this paper. As shown in the results, the standard deviation across test cases ranges from approximately 0 to 8.0, with most test cases having standard deviations between 3 and 6. Although this level of dispersion appears relatively large numerically, such variability is expected in tasks involving expert subjective judgment. This is particularly true when the evaluation dimensions include abstract and context-dependent quality attributes, where engineers make reasonable but not identical judgments based on their own understanding of system architecture, business logic, and historical failures. Further observation shows that, although there is some dispersion in the scores, its magnitude falls within the normal range of cognitive errors among technical professionals regarding product quality attributes in software engineering practice. This variation is not random noise, but rather reflects reasonable judgments made by professionals based on experience and context in real-world projects, which also validates the necessity of introducing a multi-expert evaluation and weighted fusion mechanism.

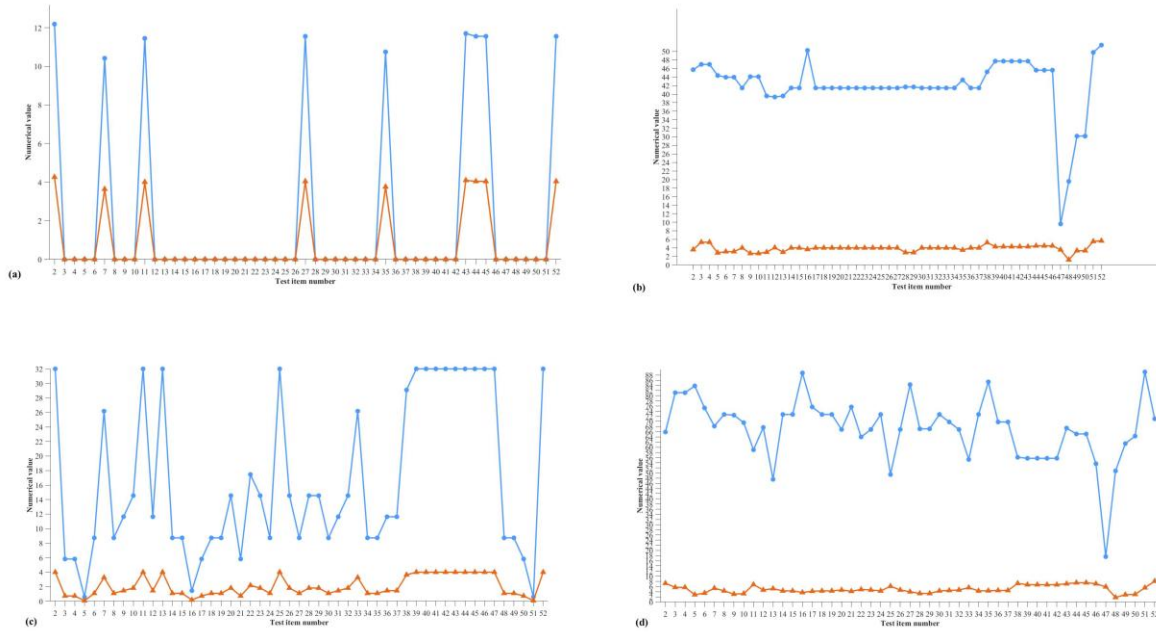


Figure 4. Inter-rater consistency analysis among five test engineers: (a) risk factor scores with mean and standard deviation; (b) importance factor scores with mean and standard deviation; (c) cost factor scores with mean and standard deviation; (d) composite score mean and standard deviation across 52 test cases.

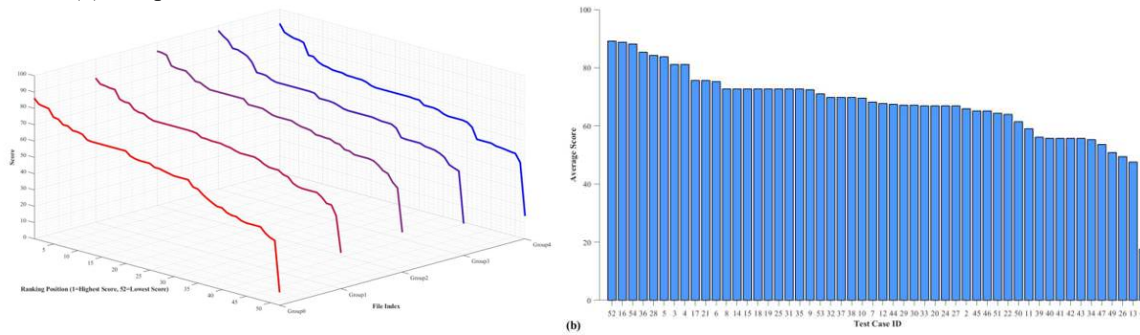


Figure 5. Cross-group scoring comparison for five experimental groups: (a) scoring results of five groups across different test case rankings in 3D visualization; (b) average scores for each test case across five groups.

Figure 5(a) shows the scoring results of five groups of data across different test cases and Figure 5(b) presents the average scores for each test case within the five groups. From these three figures, we can observe that although the scores were given by different test engineers, the degree of dispersion is relatively low due to their similar understanding of the product. Therefore, theoretically, this should have minimal impact on the final metric evaluation results.

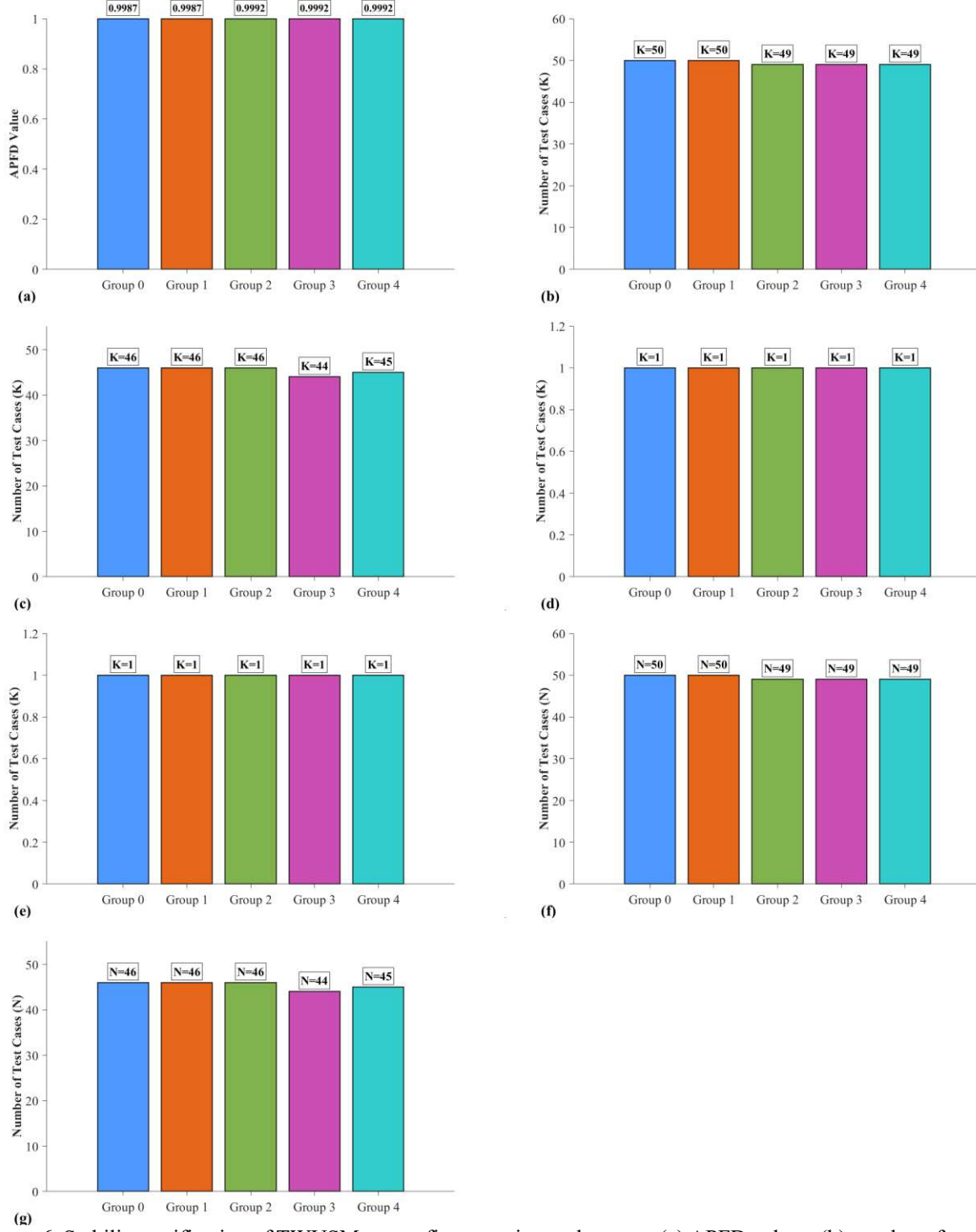


Figure 6. Stability verification of TWUSM across five experimental groups: (a) APFD values; (b) number of test cases required to achieve 99% precision; (c) number of test cases required to achieve 95% precision; (d) number of test cases required to achieve 99% recall; (e) number of test cases required to achieve 95% recall; (f) number of test cases needed to achieve 99% detection rate; (g) number of test cases needed to achieve 95% detection rate.

Figure 6(a) shows the APFD values for five groups of data, with an average APFD value of 0.9990, indicating that the data are highly concentrated and stable. As can be seen from the figure, the standard deviation of APFD for each group is very small, indicating that even if different professionals assign scores to test cases with certain subjective biases, the impact on this metric remains relatively minor.

Figure 6(b) shows the number of test cases with respect to 99% precision, Figure 6(c) shows the number of test cases with respect to 95% precision, Figure 6(d) shows the number of test cases with respect to 99% recall, Figure 6(e) shows the number of test cases with respect to 95% recall, Figure 6(f) shows the number of test cases needed to achieve 99%, Figure 6(g) shows the number of test cases needed to achieve 95%. From the figures, we can observe that there is considerable variation in the number of selected test cases across the five groups. This difference arises due to variations in test case execution scenarios. For this reason, the workflow design in Figure 1 has been carefully considered: after a certain number of tests have been executed, the test case execution plan needs to be re-adjusted accordingly.

IV. Conclusions

This paper proposes the Time-Weighted Utility Scoring Model (TWUSM) for automated test case prioritization and selection in the factory testing of industrial HMI systems. The method replaces experience-driven case selection with a quantitative, data-driven ranking mechanism that integrates three dimensions: risk impact on system safety, functional importance, and execution cost. The utility score is updated with historical execution feedback, and the resulting hybrid rule deterministically admits all cases above a threshold as mandatory tests while sampling the remainder proportionally to their normalized scores. This yields a lightweight, adaptive inspection suite that enforces complete verification of critical functionality under a fixed, non-extendable time budget. Evaluated on an industrial HMI system using APFD, Top-N detection rate, Precision@K, and Recall@K, TWUSM reduces testing time from 425.9 s to 174.9 s (a reduction of 58.9%) while achieving an APFD of 0.9987 and 100% detection of critical failures. Compared with baseline methods, TWUSM does not rely on a single dominant criterion; instead, it delivers balanced performance across multiple evaluation metrics, which makes it better suited to budget-constrained factory inspection than schemes optimized for APFD alone. Performance remains consistent across multiple independent test groups, confirming the stability and reproducibility of the approach. In summary, TWUSM transforms test case selection from an ad-hoc, experience-based practice into a documented, score-based procedure. It provides a systematic and resource-aware basis for inspection planning in safety-critical industrial software, assuring product quality while substantially reducing test duration.

Declarations

- a. Funding
Not applicable
- b. Conflicts of interest/Competing interests
Not applicable
- c. Authors' contributions

All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Gang Zheng, Juxing Xiang, and Jun Xu. Algorithm training and experimental verification were conducted by Gang Zheng and Qilin Li. The first draft of the manuscript was written by Gang Zheng, and all authors commented on previous versions of the manuscript. Guida Tang was responsible for data validation and figure organization. All authors read and approved the final manuscript.

References

1. Sultan Z, Abbas R, Bhatti S N, Shah A A. Analytical review on test cases prioritization techniques: An empirical study[J]. *International Journal of Advanced Computer Science and Applications*, 2017, 8(2).
2. Bagherzadeh M, Kahani N, Briand L. Reinforcement learning for test case prioritization[J]. *IEEE Transactions on Software Engineering*, 2021, 48(8): 2836-2856.
3. S.Yoo, M.Harman, "Regression testing Minimisation, Selection and Prioritization: A Survey" *Wiley InterScience*
4. Gupta S, Singhal A, Gosain A. "A novel approach for test case prioritization." *International Journal of Computer Science, Engineering and Applications* 2.3 (2012): 53.
5. Siripong Roongruangsuwan and Jirapun Daengdej. *Test Case Prioritization Techniques*. JATIT, 2005-2010.
6. Pan R, Bagherzadeh M, Ghaleb T A, Briand L. Test case selection and prioritization using machine learning: a systematic literature review[J]. *Empirical Software Engineering*, 2022, 27(2): 29.
7. C. L. B. Maia, R. A. F. do Carmo, F. G. De Freitas, G. A. L. de campos and J. T. De Souza, "Automated test case prioritization with reactive GRASP," In *Proceedings of Advances in Software Engineering*, pp.1-18,2010.
8. Bahrapour A, Rafe V (2020b) Using search-based techniques for testing executable software models specified through graph transformations. *Int J Mach Learn Cybern* 11(12):2743–2770
9. M. J. Arafeen and H. Do, "Test Case Prioritization Using Requirements-Based Clustering," 2013 IEEE Sixth

- International Conference on Software Testing, Verification and Validation, Luxembourg, Luxembourg, 2013, pp. 312-321
10. Z. Li, M. Harman and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization," in IEEE Transactions on Software Engineering, vol. 33, no. 4, pp. 225-237, April 2007
 11. P. Tonella, P. Avesani and A. Susi, "Using the Case-Based Ranking Methodology for Test Case Prioritization," 2006 22nd IEEE International Conference on Software Maintenance, Philadelphia, PA, USA, 2006, pp. 123-133
 12. Vecer J. Dynamic scoring: Probabilistic model selection based on utility maximization[J]. Entropy, 2019, 21(1): 36.
 13. R. Lachmann, S. Schulze, M. Nieke, C. Seidl and I. Schaefer, "System-Level Test Case Prioritization Using Machine Learning," 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), Anaheim, CA, USA, 2016, pp. 361-368
 14. Scholz M, Dorner V, Franz M, Hinz O. Measuring consumers' willingness to pay with utility-based recommendation systems[J]. Decision Support Systems, 2015, 72: 60-71.
 15. Tomlinson G, Bremner K E, Ritvo P, Naglie G, Krahn M D. Development and validation of a utility weighting function for the Patient-Oriented Prostate Utility Scale (PORPUS)[J]. Medical Decision Making, 2012, 32(1): 11-30.
 16. Torrance G W, Feeny D H, Furlong W J, Barr R D, Zhang Y, Wang Q. Multiattribute utility function for a comprehensive health status classification system: Health Utilities Index Mark 2[J]. Medical care, 1996, 34(7): 702-722.
 17. Golbaz S, Yaghmaeian K, Isazadeh S, Zamanzadeh M. Environmental risk assessments of multiclass pharmaceutical active compounds: selection of high priority concern pharmaceuticals using entropy-utility functions[J]. Environmental Science and Pollution Research, 2021, 28(42): 59745-59770.
 18. Mostavi E, Asadi S, Boussaa D. Development of an Integrated Quality Function Deployment and Utility Theory Weighting System to Improve Occupants' Satisfaction[J]. Journal of Architectural Engineering, 2019, 25(4): 04019020.
 19. Li L, Liu M, Shen W, Cheng G. Recommending mobile services with trustworthy QoS and dynamic user preferences via FAHP and ordinal utility function[J]. IEEE Transactions on Mobile Computing, 2019, 19(2): 419-431.
 20. Senouci M A, Hoceni S, Mellouk A. Utility function-based TOPSIS for network interface selection in heterogeneous wireless networks[C]//2016 IEEE international conference on communications (ICC). IEEE, 2016: 1-6.
 21. Leoneti A B. Utility function for modeling group multicriteria decision making problems as games[J]. Operations Research Perspectives, 2016, 3: 21-26.
 22. Krishnamoorthi R, Mary S A S A. Factor oriented requirement coverage based system test case prioritization of new and regression test cases[J]. Information and Software Technology, 2009, 51(4): 799-808.