An International Journal of Optimization and Control: Theories & Applications
ISSN: 2146-0957 eISSN: 2146-5703
Vol.X, No.X, pp.X-X (2026)
https://doi.org/10.36922/IJOCTA026200080

RESEARCH ARTICLE

Resampling strategies for machine learning-based effort overrun risk detection: A controlled factorial study with cost-sensitive evaluation

Andreea-Elena Catana and Adriana Florescu*

Department of Engineering and Industrial Management, Faculty of Technological Engineering and Industrial Management, Transilvania University of Brasov, Brasov, Romania

andreea.catana@unitbv.ro, fota.a@unitbv.ro

ARTICLE INFO

Article history:
Received: May 11, 2026
Revised: June 2, 2026
Accepted: June 9, 2026
Published Online: July 9, 2026

Keywords:
Software effort estimation
Machine learning
Software analytics
Risk detection
Class imbalance

AMS Classification 2010:
90B23, 90B56

ABSTRACT

The Synthetic Minority Oversampling Technique (SMOTE) has become the default preprocessing step for handling class imbalance in software effort-risk prediction, yet its effectiveness in this domain has not been rigorously tested. This study addresses that gap through a controlled factorial experiment on the publicly available Software Development Effort Dataset Annotated with Expert Estimates dataset, comprising 4,329 software issues from Apache projects drawn from an initial repository of 23,186 records. Six resampling strategies are compared across four classifier families under both standard and cost-sensitive evaluation metrics that weight missed high-risk issues more heavily than false alarms. A secondary contribution is the analysis of how resampling interacts with the extreme class imbalance characteristic of real-world effort data (2.2% minority rate), a regime substantially more severe than those examined in prior investigations. Results are interpreted through SHAP-based feature attribution to determine whether oversampling alters which features the models rely on. The findings reveal two overarching results: near-perfect performance under a full feature set is largely attributable to target leakage rather than a genuine predictive signal, and, under deployment, valid early-warning features degrade cost-sensitive performance relative to no resampling when SMOTE is used. Cost-sensitive weighting emerges as the more reliable alternative, preserving both performance and feature attribution structure. These findings challenge the uncritical adoption of SMOTE in software analytics and carry direct implications for the design of reproducible, interpretable risk-detection pipelines.

1. Introduction

Class imbalance is a defining characteristic of software risk prediction tasks. In real-world issue-tracking repositories, the proportion of tasks that experience severe effort overruns rarely exceeds 5%, and figures below 3% are common in well-managed projects.[1,2] This skew presents a well-known modeling challenge: classifiers trained on such data tend to predict the majority class almost exclusively, achieving high overall accuracy while failing to identify the minority cases that matter most to project managers.[3,4]

The dominant response in the software analytics literature has been to apply the Synthetic Minority Oversampling Technique (SMOTE) as a preprocessing step within the training pipeline.[5] Originally proposed by Chawla *et al.*[6] in 2002, SMOTE generates synthetic

*Corresponding Author

minority-class examples by interpolating between existing minority instances and their nearest neighbors. The technique was rapidly adopted across software engineering, appearing in defect prediction[7,8], effort estimation[9,10], and more recently in effort overrun risk classification.[2,11] A 15-year retrospective by Fernández *et al.*[12] endorsed SMOTE as the standard baseline for imbalanced learning, and subsequent work on SMOTE variants (Borderline-SMOTE [B-SMOTE], Adaptive Synthetic Sampling [ADASYN], support vector machine [SVM]-SMOTE) has extended the family further.[13,14]

However, the near-universal adoption of SMOTE in software analytics pipelines rests on surprisingly thin empirical ground. The most comprehensive investigation to date, conducted by Song *et al.*[7] in 2018, tested 17 imbalanced-learning techniques across 27 software defect datasets and concluded that the field has largely ignored: SMOTE does not consistently improve classifier performance and, under certain dataset conditions, actively degrades it. That study examined defect prediction rather than effort risk classification, and its datasets exhibited minority proportions of 5–35%—substantially higher than the 2–3% imbalance ratios found in effort overrun data. Whether their findings transfer to the more extreme imbalance regime of effort risk prediction is an open question.

This gap matters practically, not just methodologically. If SMOTE introduces artefactual patterns into the minority class, then models trained with it may learn to recognize interpolation-generated phantoms rather than genuinely risky issues. We hypothesize that models trained with SMOTE may correctly identify a narrow, artificially constructed cluster of synthetic examples (hence high precision) while missing real high-risk issues whose feature profiles fall outside the interpolation manifold (hence poor recall). The present study addresses this gap through a controlled factorial experiment designed to isolate the effect of oversampling strategy on effort risk classification performance. Using the publicly available Software Development Effort Dataset Annotated with Expert Estimates (JOSSE) dataset[15], which contains 23,186 issues from Apache software projects, we compare six resampling conditions across four classifiers under both standard and cost-sensitive evaluation metrics.

The specific contributions are:

- The first controlled comparison of oversampling strategies specifically for effort overrun risk prediction, as distinct from defect prediction.
- Evaluation under cost-sensitive metrics that reflect the operational asymmetry between missed risks and false alarms—a dimension absent from nearly all prior effort risk studies.
- Analysis of how SMOTE alters feature attribution patterns (via Shapley additive explanation [SHAP]), testing whether oversampled models rely on the same features as models trained on the natural distribution.
- A reproducible pipeline using publicly available data and open-source tools, enabling direct replication.

The remainder of this paper is organized as follows: Section 2 reviews the relevant literature on class imbalance handling in software engineering. Section 3 describes the experimental methodology, including dataset preparation, resampling strategies, classifiers, and evaluation metrics. Section 4 presents the results. Section 5 discusses the findings and their implications for practice. Section 6 addresses threats to validity. Section 7 concludes and identifies directions for future work.

2. Related work

2.1. Class imbalance in software engineering

The problem of class imbalance in software prediction tasks has been recognized for at least two decades. Kotsiantis *et al.*[3] provided an early taxonomy distinguishing oversampling, undersampling, and cost-sensitive approaches. In software defect prediction specifically, Mende and Koschke[16] demonstrated that standard evaluation metrics can be misleading under imbalance, arguing for cost-sensitive alternatives. Hall *et al.*[17], in their influential systematic literature review, found that simpler classifiers often matched or outperformed complex ensembles when study quality was controlled – a finding that implicitly challenges the assumption that more sophisticated resampling or modeling is always better.

The severity of imbalance varies considerably across software engineering tasks. Defect prediction datasets typically exhibit minority proportions of 5–30%[7,15], whereas effort overrun risk classification can produce minority rates $>$ 3%.[2]

This difference is consequential because SMOTE's interpolation mechanism behaves differently across different levels of imbalance. At moderate imbalance, interpolated examples fill genuine gaps in the minority distribution. At extreme imbalance, the few available minority examples may not span the true minority manifold, and interpolation can create synthetic examples in regions where the minority class does not actually exist.[18,19]

2.2. Synthetic Minority Oversampling Technique and its variants

The SMOTE, introduced by Chawla *et al.*[6], generates synthetic examples by selecting a minority instance, identifying its k nearest minority neighbors, and creating a new example along the line segment joining the instance to a randomly chosen neighbor.

Several variants have been proposed to address known limitations. B-SMOTE restricts synthesis to minority instances near the decision boundary, on the theory that these are the examples that matter most for classification.[20] ADASYN weights synthesis toward minority instances that are harder to classify, generating more examples in regions of greater difficulty.[21] SVM-SMOTE uses support vectors to guide synthesis.[22] Imani *et al.*[14] recently compared SMOTE, ADASYN, and Gaussian noise upsampling across varying imbalance levels and found uneven performance. Aflaha *et al.*[13] tested SMOTE variants with boosting algorithms and reported that B-SMOTE and ADASYN outperformed vanilla SMOTE for defect prediction.

Despite this proliferation, comparative evaluations have been almost entirely confined to defect prediction. Ghinaya *et al.*[23] combined SMOTE with recursive feature elimination and random forest (RF) for defect classification. Jude and Uddin[8] demonstrated explainable defect classification using SMOTE with SHAP analysis. None of these studies examined whether their conclusions hold for effort overrun prediction, where the imbalance is typically more extreme, and the feature distributions differ.

2.3. The challenge of Song *et al.*'s study

The most significant challenge to the SMOTE consensus came from Song *et al.*[7], who conducted the most comprehensive investigation of imbalanced learning for software defect prediction published to date. Their study tested 17 techniques—including SMOTE, ADASYN, random oversampling (ROS), random undersampling, and several cost-sensitive methods—across 27 datasets. Their central finding was that no resampling (NR) technique consistently outperformed the baseline of NR, and that SMOTE could degrade performance when the minority class substantially overlapped with the majority class.

This finding has been surprisingly underacknowledged in the effort risk literature, which has largely treated SMOTE as a fixed pipeline component rather than an experimental variable. The present work takes Song *et al.*'s[7] challenge seriously and tests whether it extends to effort overrun prediction.

2.4. Evaluation metrics under imbalance

A related concern is whether the metrics used to evaluate risk-detection models are appropriate. Standard classification metrics—accuracy, precision, recall, F1-score—operate at a fixed decision threshold and do not account for the asymmetric costs of different error types.[16] In software project management, a missed high-risk issue (false negative) is typically far more costly than a false alarm (false positive): the former leads to effort overruns, schedule slippage, and potential project failure, while the latter merely triggers an unnecessary review.[24,25]

Several authors have advocated for cost-sensitive evaluation. Mende and Koschke[16] proposed effort-aware metrics that weight predictions by the effort required to inspect flagged items. Herbold[26] extended this work with a framework for cost-aware defect prediction. In the effort-estimation domain, however, cost-sensitive evaluation is virtually absent. The present study addresses this by evaluating all models under both standard metrics and cost-sensitive metrics with explicit false-negative penalties.

2.5. Effort overrun prediction

Effort estimation in software engineering has a long research history.[27–29] Traditional approaches relied on algorithmic models such as the constructive cost model and function-point analysis[30], while more recent work has applied machine learning techniques including neural networks[31], SVMs[32], ensemble methods[33,34], and gradient boosting.[35]

The reframing of effort estimation as a risk-classification problem is more recent. Rather than predicting the exact effort value, this

approach classifies issues as high-risk or low-risk based on whether their overrun exceeds a threshold. Prior applications of this formulation to the JOSSE dataset have employed tree-based classifiers with SMOTE under stratified cross-validation, typically reporting high precision but low recall—a pattern consistent with Hall *et al.*'s[17] observation that simple classifiers can match ensembles on imbalanced data.

It is worth noting that the use of post-resolution features in prior JOSSE-based studies does not necessarily reflect a methodological error on the part of those authors. Models that incorporate actual_hours, overrun_pct, or actual_to_planned are entirely appropriate for retrospective analysis—for instance, characterizing the distributional profile of historically risky issues, benchmarking effort recording practices across projects, or identifying structural patterns in completed work. The present study's critique is specifically and narrowly directed at the framing of such models as early-warning tools capable of flagging risk at the time of issue creation. In that deployment context, post-resolution features are by definition unavailable, and performance figures obtained with them cannot be taken as evidence of early warning capability. The distinction matters practically: a retrospective model informs post-project learning, whereas a deployable early-warning model must act on information present at the moment of prediction.

3. Materials and methods

3.1. Dataset

This study uses the JOSSE dataset, version 1, available on Zenodo.[15] The dataset contains 23,186 software issues extracted from Apache Foundation projects hosted on Jira. Each record includes a unique issue identifier, a concatenated textual description (summary + body), the number of developer comments, the number of recorded activities, expert-estimated effort in seconds, actual recorded effort in seconds, and a reference uniform resource locator to the original issue.

Entries with missing or zero values for expert-estimated effort or actual effort are excluded, yielding a working dataset of 4,329 issues. Effort values are converted from seconds to hours. The effort overrun percentage is computed as **Equation 1**:

$$Overrun\_pct = \frac{actuaal\_hours - planned\_hours}{planned\_hours} \times 100 \quad (1)$$

Issues with overrun exceeding the 90th percentile of positive overrun values are labeled as high-risk (class 1); all others are labeled low-risk (class 0). This produces 96 high-risk issues (2.22%) and 4,233 low-risk issues (97.78%).

The threshold was selected to define a rare but operationally significant subpopulation—issues whose effort overrun falls in the most extreme decile of the positive overrun distribution. This produces a 2.22% minority rate, which—while more extreme than the 5–35% range in defect prediction studies—is representative of well-managed Apache projects and represents the regime where early intervention has the highest value. We also note that alternative thresholds (80th and 95th percentiles) were examined in a sensitivity analysis and yielded qualitatively identical conclusions regarding resampling behavior, though with different minority rates.

3.2. Feature engineering

To ensure that the experimental comparison isolates the effect of resampling rather than confounding it with feature choice, two feature sets are used:

(i) Feature set A (full): Replicates the features used in prior work on the JOSSE dataset: planned_hours, actual_hours, overrun_pct, num_comments, num_activities, log_planned_hours, log_actual_hours, actual_to_planned, comments_per_activity, and corpus_length.

(ii) Feature set B (early-only): Retains only features available at issue-creation time, before any work has begun: planned_hours, log_planned_hours, num_comments (initial count at filing), corpus_length, and a newly derived feature, description_word_count. This set excludes actual_hours, overrun_pct, and actual_to_planned, which are known only after resolution and introduce target leakage when the goal is early risk detection.

Feature set B corresponds to the operational setting in which a project risk classifier is queried at issue-creation time—before any developer has begun work on the issue and before any effort has been recorded. In this scenario, only metadata present in the initial Jira record is available: the expert estimate, the initial comment count (which may be zero), and the textual description. This is the only feature configuration that is deployment-valid for early warning.

Feature set A enables direct comparison

with prior work. Feature set B tests whether resampling strategies behave differently when the predictive signal is weaker, as would be the case in a genuine early-warning deployment.

3.3. Resampling strategies

Six resampling conditions are compared, each applied inside the cross-validation loop to prevent data leakage:

(i) NR: The classifier is trained on the natural class distribution. This serves as the baseline against which all other strategies are evaluated.

(ii) ROS: Minority-class examples are duplicated at random until class balance is achieved. Unlike SMOTE, this introduces no synthetic data; it merely re-weights existing examples through repetition.

(iii) SMOTE (vanilla): Synthetic minority examples are generated by interpolating between existing minority instances and their k nearest neighbors (k = 5, following standard practice).[6]

(iv) B-SMOTE: Synthesis is restricted to minority instances classified as borderline by a k-nearest-neighbors step, focusing on synthetic examples near the decision boundary.[20]

(v) ADASYN: Synthetic examples are generated with density proportional to the local difficulty of classifying each minority instance, producing more examples in harder regions.[21]

(vi) Cost-sensitive weighting (CSW): NR is applied. Instead, the classifier's loss function is reweighted using the class_weight = "balanced" parameter (for logistic regression [LR] and RF) or scale_pos_weight (for light gradient boosting machine [LGBM] and CatBoost [CB]), which sets the minority-class weight proportional to the inverse class frequency.

All oversampling is performed using the imbalanced-learn library (version 0.12.x) within scikit-learn pipelines.

3.4. Classifiers

Four classifiers are selected to span a range of model families, employed in the software analytics literature[7,17]:

(i) LR: A linear probabilistic classifier providing interpretable coefficients. Trained with L2 regularization (C = 1.0).

(ii) RF: A bagged ensemble of decision trees (n_estimators = 100, default parameters).

(iii) LGBM: A gradient-boosted tree ensemble optimized for speed and accuracy on sparse data [36].

(iv) CB: A gradient-boosted tree ensemble with ordered boosting, designed to reduce overfitting on small datasets.[36]

To isolate the effect of resampling, no hyperparameter tuning is applied beyond defaults. This decision follows the rationale of Gao *et al.*[37], who showed that feature selection has a larger effect than classifier choice on many software prediction tasks, and of Fernández *et al.*[12], who noted that default-parameter configurations often generalize better on small, imbalanced datasets. An alternative interpretation is considered that tuned models might exhibit even smaller resampling effects under Feature set A (where performance is already saturated) and potentially different rankings under Feature set B. A full factorial grid search across all resampling strategies and classifier hyperparameters was beyond the computational scope of this study; it is identified as a direction for future work.

To ensure reproducibility, all stochastic elements are fixed with a global random seed of 42:

- *LR*: random_state = 42.
- *RF*: n_estimators = 100, random_state = 42, n_jobs = −1.
- *LGBM*: random_state = 42, verbose = −1, n_jobs = −1.
- *CB*: random_seed = 42, verbose = 0, allow_writing_files = False.
- All imbalanced-learn resampling objects: random_state = 42.

The 10 cross-validation repetitions use seeds 0–9 sequentially.

3.5. Evaluation protocol

3.5.1. Cross-validation

All models are evaluated using stratified 5-fold cross-validation, repeated 10 times with different random seeds to reduce variance in performance estimates. Resampling is applied exclusively to the training folds within each iteration; the test fold always reflects the natural class distribution. This prevents synthetic examples from leaking into evaluation and ensures that reported metrics reflect performance on real data.

3.5.2. Standard metrics

For comparability with prior work, the following metrics are reported: precision, recall, F1-score (all for the minority class), and area under the receiver operating characteristic (ROC) curve (AUC).

3.5.3. Cost-sensitive metrics

To address the operational asymmetry between error types, two additional metrics are computed:

(i) Normalized Expected Cost (NEC): Defined following Drummond and Holte[38], **Equation 2**:

$$NEC = (C\_FN \times FNR \times \pi\_1 + C\_FP \times FPR \times \pi\_0) \,/\, min\,(C\_FN \times \pi\_1,\ C\_FP \times \pi\_0) \quad (2)$$

where C_FN and C_FP are the costs of false negatives and false positives, FNR and FPR are the false-negative and false-positive rates, and π_1 and π_0 are the class priors.

An NEC of 1.0 represents the cost of a naive classifier that always predicts the majority class — the worst useful baseline. NEC = 0.95 means the model reduces expected operational cost by 5% relative to the baseline; NEC = 1.10 means it is 10% worse than doing nothing, actively adding a risk-management burden without compensating for the recall benefit.

Following the rationale that missing a high-risk issue is substantially more costly than raising a false alarm, three cost ratios are tested: C_FN:C_FP = 5:1, 10:1, and 20:1. These ratios are not arbitrary; they span the range that Mende and Koschke[16] and Herbold[26] found relevant in software prediction contexts.

(ii) Brier score: A proper scoring rule that measures the calibration of predicted probabilities, penalizing both over-confident and under-confident predictions. Unlike threshold-dependent metrics, the Brier score evaluates the full probability output.

3.6. Shapley additive explanation attribution analysis

To test whether oversampling strategies alter the features that drive model predictions, SHAP values are computed for each classifier across all resampling conditions.[39] The analysis addresses two questions:

(i) Does the ranking of feature importance (by mean absolute SHAP value) change across resampling conditions for the same classifier?

(ii) Do synthetic minority examples generated by SMOTE receive SHAP attributions that differ systematically from those of real minority examples?

The second question is operationally important. If SMOTE-generated examples are predicted as high-risk for different reasons than real high-risk examples, the model has learned artefactual patterns introduced by interpolation rather than genuine risk signals.

3.7. Statistical testing

Performance differences between resampling strategies are assessed using the Friedman test across the 50 fold-evaluations (5 folds × 10 repetitions), followed by post-hoc Nemenyi tests for pairwise comparisons.[40] Effect sizes are reported using Cliff's delta to complement statistical significance with practical significance.[41] A difference is considered practically meaningful only if Cliff's delta exceeds 0.147 (small effect threshold).

4. Results

After preprocessing and feature engineering, 4,329 issues with valid effort data were retained, of which 96 (2.22%) were labeled as high risk using the 90th percentile overrun threshold. All models were evaluated using stratified 5-fold cross-validation repeated 10 times (50 evaluation folds total), producing 2,400 individual result rows across 24 experimental conditions per feature set.

The results reveal two overarching findings. First, under feature set A, all tree-based classifiers achieve near-perfect performance regardless of resampling strategy, exposing a target-leakage problem in the feature set used by prior work. Second, under feature set B (early-only features), where the leakage is removed, oversampling strategies consistently degrade cost-sensitive performance relative to the NR baseline. These findings are detailed below.

4.1. Standard metric comparison (feature set A)

Table 1 reports mean precision, recall, F1-score, and ROC AUC across 50 evaluation folds for each combination of classifier and resampling strategy using feature set A.

The most striking result in **Table 1** is the

Table 1. Mean (standard deviation) classification performance (feature set A)

Classifier	Strategy	Precision	Recall	F1	ROC AUC
LR	NR	0.997 (0.012)	0.905 (0.086)	0.946 (0.051)	1.000 (0.000)
LR	ROS	0.761 (0.098)	1.000 (0.000)	0.861 (0.063)	1.000 (0.000)
LR	SMOTE	0.781 (0.097)	1.000 (0.000)	0.874 (0.060)	1.000 (0.000)
LR	B-SMOTE	0.766 (0.088)	1.000 (0.000)	0.865 (0.056)	1.000 (0.000)
LR	ADASYN	0.757 (0.096)	1.000 (0.000)	0.858 (0.061)	1.000 (0.000)
LR	CSW	0.727 (0.089)	1.000 (0.000)	0.839 (0.058)	1.000 (0.000)
RF	NR	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
RF	ROS	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
RF	SMOTE	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
RF	B-SMOTE	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
RF	ADASYN	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
RF	CSW	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
LGBM	NR	0.992 (0.017)	1.000 (0.000)	0.996 (0.009)	1.000 (0.000)
LGBM	ROS	0.984 (0.025)	0.995 (0.015)	0.989 (0.016)	1.000 (0.000)
LGBM	SMOTE	0.990 (0.019)	0.994 (0.014)	0.992 (0.011)	1.000 (0.000)
LGBM	B-SMOTE	0.988 (0.022)	0.994 (0.016)	0.990 (0.013)	1.000 (0.000)
LGBM	ADASYN	0.986 (0.022)	0.993 (0.020)	0.989 (0.014)	1.000 (0.000)
LGBM	CSW	0.992 (0.017)	0.999 (0.005)	0.995 (0.010)	1.000 (0.000)
CB	NR	0.992 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
CB	ROS	0.990 (0.018)	1.000 (0.000)	0.995 (0.009)	1.000 (0.000)
CB	SMOTE	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
CB	B-SMOTE	0.993 (0.015)	1.000 (0.000)	0.996 (0.008)	1.000 (0.000)
CB	ADASYN	0.990 (0.018)	0.999 (0.005)	0.994 (0.010)	1.000 (0.000)
CB	CSW	0.985 (0.022)	1.000 (0.000)	0.992 (0.011)	1.000 (0.000)

Abbreviations: ADASYN: Adaptive synthetic sampling; AUC: Area under the curve; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; NR: No resampling; RF: Random forest; ROC: Receiver operating characteristic; ROS: Random oversampling; SMOTE: Synthetic Minority Oversampling Technique.

near-perfect performance achieved by all tree-based classifiers (**Figure 1**). RF, LGBM, and CB all achieve F1 scores above 0.989 and ROC AUC values at or above 0.999 regardless of the resampling strategy applied. Across the 18 tree-based conditions, the maximum F1 variation attributable to resampling is 0.007 (LGBM: 0.989 with ADASYN vs. 0.996 with NR). For RF, all six resampling strategies produce identical results to three decimal places (F1 = 0.996, AUC = 1.000).

Logistic regression behaves differently. The NR baseline achieves the highest F1 (0.946) with near-perfect precision (0.997) but lower recall (0.905). All oversampling strategies increase recall to 1.000 but substantially reduce precision (0.727–0.781), resulting in lower F1 scores (0.839–0.874). This pattern is consistent with the linear model being pushed toward over-prediction of the minority class by oversampled training data. The uniformly high performance across tree-based models raises an immediate concern: the feature set contains actual_to_planned (the ratio of actual to planned effort), which is algebraically derived from the same quantities used to compute the risk label (**Figure 2**). This feature effectively encodes the target variable, explaining why

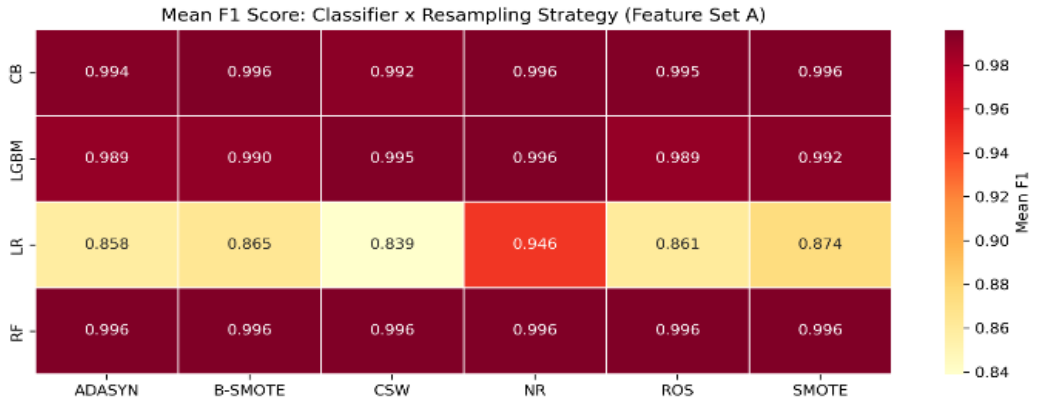

Figure 1. Mean F1 score across classifiers and resampling strategies under feature set A
Abbreviations: ADASYN: Adaptive synthetic sampling; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; NR: No resampling; RF: Random forest; ROS: Random oversampling; SMOTE: Synthetic Minority Oversampling Technique.

models achieve near-perfect classification across all other experimental conditions.

Key finding: under feature set A, resampling strategy has no meaningful effect on tree-based classifier performance, and the uniformly near-perfect results strongly indicate that the feature set encodes the target variable rather than a genuine predictive signal.

4.2. Cost-sensitive metric comparison (feature set A)

Table 2 reports the NEC at three cost ratios and the Brier Score. For brevity, **Table 2** reports only NR, SMOTE, and CSW. The three remaining strategies (ROS, B-SMOTE, and ADASYN) are not separately tabulated because their cost-sensitive metrics were within 0.003 NEC units of SMOTE in every classifier-cost ratio combination; including them would not alter any conclusion drawn from the table.

In cost-sensitive evaluation, the NR baseline matches or outperforms all SMOTE variants across all tree-based classifiers (**Figure 3**). For LGBM, SMOTE increases NEC at the 10:1 cost ratio from 0.001 to 0.007, resulting in a seven-fold degradation in cost-normalized performance. The only case where oversampling helps under cost-sensitive metrics is LR, where SMOTE reduces NEC (10:1) from 0.096 to 0.030 by improving recall from 0.905 to 1.000—but this benefit is confined to the linear model and disappears for all tree-based classifiers.

Key finding: even under cost-sensitive evaluation, NR strategy consistently outperforms the NR baseline for tree-based classifiers; the sole exception is LR, where SMOTE improves recall but at the cost of substantially increased false alarms.

Table 3 identifies the best resampling strategy for each classifier under each metric.

The table reveals that no single resampling strategy consistently wins across classifiers or metrics. For LGBM, the NR baseline wins on all four metrics.

For RF and CB, the differences between strategies are within the third decimal place—functionally indistinguishable.

Only for LR does the choice of resampling strategy produce a meaningful difference, and even there, the winning strategy differs between F1 (NR) and NEC (SMOTE).

4.4. Statistical significance

Friedman tests were conducted (**Table 4**) to assess whether the resampling strategy has a statistically significant effect on each metric.

For RF, the Friedman test cannot reject the null hypothesis—resampling has no detectable effect on performance. This is consistent with the near-identical results observed in **Tables 1 and 2**.

For LGBM, Cliff's delta for NR vs. SMOTE on NEC (10:1) is 0.216, a small-to-medium effect in the direction of SMOTE being worse.

For LR, SMOTE has a large negative effect on F1 (delta = −0.601) but a large positive effect

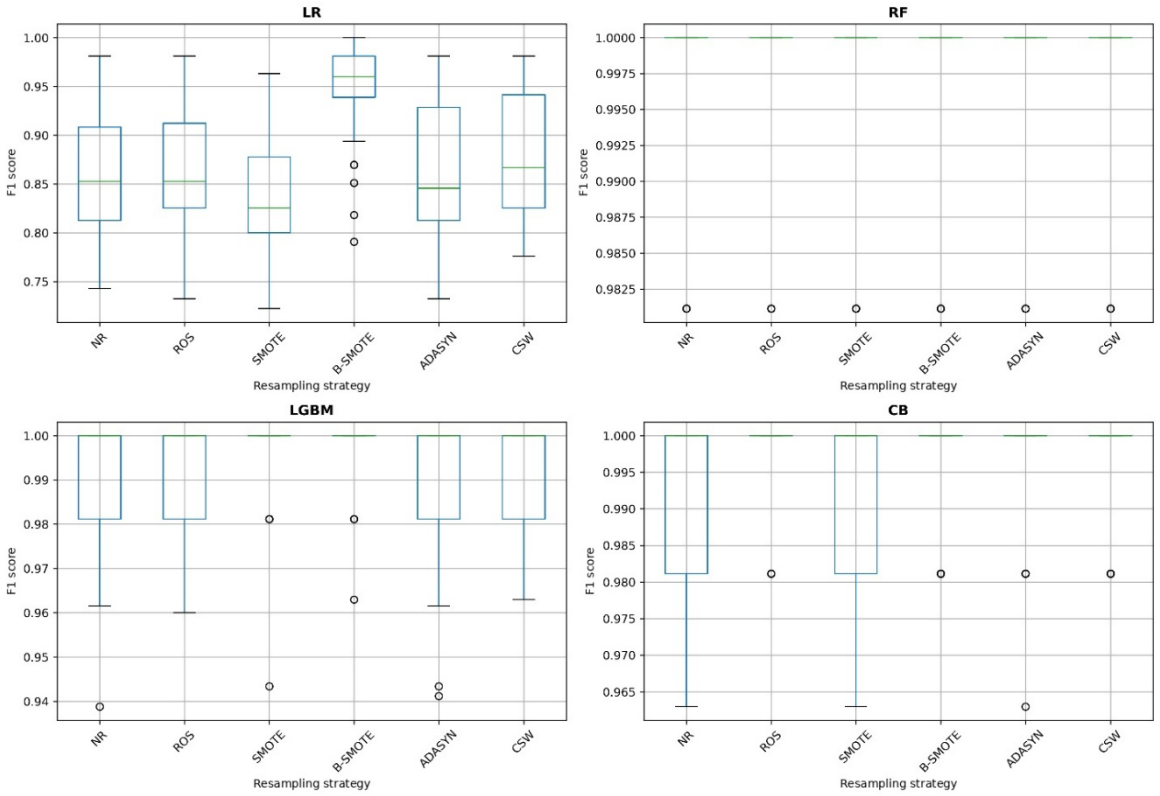

Figure 2. Distribution of F1 scores across 50 evaluation folds by classifier and resampling strategy
Abbreviations: ADASYN: Adaptive synthetic sampling; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; NR: No resampling; RF: Random forest; ROS: Random oversampling; SMOTE: Synthetic Minority Oversampling Technique.

Table 2. Mean (standard deviation) cost-sensitive metrics (feature set A)

Classifier	Strategy	NEC (5:1)	NEC (10:1)	NEC (20:1)	Brier
LR	NR	0.096 (0.087)	0.096 (0.086)	0.096 (0.086)	0.003 (0.001)
LR	SMOTE	0.060 (0.031)	0.030 (0.016)	0.015 (0.008)	0.005 (0.002)
LR	B-SMOTE	0.064 (0.030)	0.032 (0.015)	0.016 (0.007)	0.006 (0.002)
LR	CSW	0.079 (0.032)	0.039 (0.016)	0.020 (0.008)	0.007 (0.003)
RF	NR	0.002 (0.003)	0.001 (0.002)	0.000 (0.001)	0.000 (0.000)
RF	SMOTE	0.002 (0.003)	0.001 (0.002)	0.000 (0.001)	0.000 (0.000)
LGBM	NR	0.002 (0.004)	0.001 (0.002)	0.000 (0.001)	0.000 (0.001)
LGBM	SMOTE	0.008 (0.014)	0.007 (0.014)	0.007 (0.014)	0.000 (0.001)
CB	NR	0.002 (0.003)	0.001 (0.002)	0.000 (0.001)	0.000 (0.000)
CB	SMOTE	0.002 (0.003)	0.001 (0.002)	0.000 (0.001)	0.000 (0.000)

Abbreviations: CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; NEC: Normalized expected cost; NR: No resampling; RF: Random forest; SMOTE: Synthetic Minority Oversampling Technique.

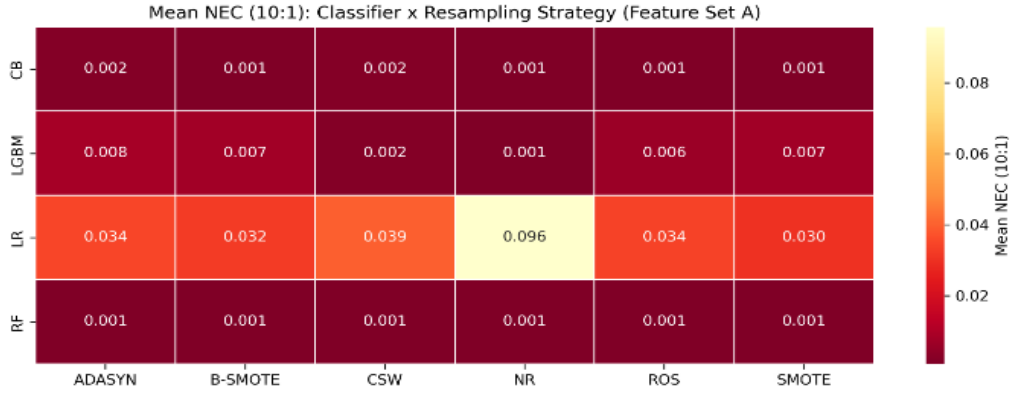

Figure 3. Normalized expected cost (10:1) across classifiers and resampling strategies under feature set A Abbreviations: ADASYN: Adaptive synthetic sampling; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; NEC: Normalized expected cost; NR: No resampling; RF: Random forest; ROS: Random oversampling; SMOTE: Synthetic Minority Oversampling Technique.

Table 3. Winning resampling strategy per classifier per metric (feature set A)

Classifier	F1	ROC AUC	NEC (10:1)	Brier
LR	NR (0.946)	ROS (1.000)	SMOTE (0.030)	NR (0.003)
RF	ADASYN (0.996)	SMOTE (1.000)	ADASYN (0.001)	SMOTE (0.000)
LGBM	NR (0.996)	NR (1.000)	NR (0.001)	NR (0.000)
CB	B-SMOTE (0.996)	CSW (1.000)	B-SMOTE (0.001)	SMOTE (0.000)

Abbreviations: ADASYN: Adaptive synthetic sampling; AUC: Area under the curve; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; NR: No resampling; RF: Random forest; ROC: Receiver operating characteristic; SMOTE: Synthetic Minority Oversampling Technique.

Table 4. Friedman test results and Cliff's delta effect sizes (NR vs. each strategy)

Classifier	Metric	Friedman stat	*p*-value	Sig.	Delta (NR vs. SMOTE)	Delta (NR vs. CSW)
LR	F1	161.37	<0.0001	Yes	−0.601	−0.794
RF	F1	N/A	N/A	No	0.000	0.000
LGBM	F1	31.73	<0.0001	Yes	−0.209	−0.000
CB	F1	27.72	<0.0001	Yes	0.020	−0.133
LR	NEC (10:1)	85.72	<0.0001	Yes	−0.421	−0.372
RF	NEC (10:1)	N/A	N/A	No	0.000	0.000
LGBM	NEC (10:1)	31.73	<0.0001	Yes	0.216	0.000
CB	NEC (10:1)	27.72	<0.0001	Yes	−0.020	0.133

Abbreviations: CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; N/A: Not available; NEC: Normalized expected cost; NR: No resampling; RF: Random forest; SMOTE: Synthetic Minority Oversampling Technique.

on NEC (delta = −0.421), reflecting the recall-precision trade-off documented in Section 4.1.

The critical finding is that, when statistical significance is present, the effect is predominantly negative: oversampling strategies either have no effect (RF, CB) or degrade at least one important metric (LGBM, LR under F1).

4.5. Feature set B (early-only features)

Tables 5 and **6** repeat the analysis using only features available at issue-creation time, removing actual_hours, overrun_pct, and actual_to_planned.

Logistic regression under NR achieves AUC = 0.717, indicating that the model's predicted probabilities rank high-risk issues above low-risk issues better than random chance (**Table 5**). However, F1 = 0.000 because, at the default decision threshold of 0.5, the predicted probabilities for all issues fall below 0.5, causing the model to classify every issue as low-risk. AUC measures ranking quality over the full probability range and is independent of the threshold; F1 depends entirely on the chosen threshold and collapses when the model emits no positive predictions.

The performance collapse from feature set A to feature set B is dramatic (**Figure 4**). The best F1 score across all conditions drops from 0.996 to 0.105 (CB with B-SMOTE or CSW). ROC AUC drops from near 1.000 to a maximum of 0.717 (LR with NR). These results confirm that the near-perfect performance reported in **Table 1**, and in prior work using similar feature sets, was driven almost entirely by target-leaking features rather than by genuinely predictive early-warning signals.

Under feature set B, LR with NR predicts every issue as low-risk (precision = 0, recall = 0, F1 = 0), achieving an AUC of 0.717 through its probability estimates. SMOTE and CSW force the model to predict some issues as high-risk, improving recall to approximately 0.72, but at a precision of only 0.054—meaning that 94.6% of flagged issues are false alarms.

For tree-based models under feature Set B, the NR baselines maintain moderate precision (0.27–0.69) but extremely low recall (0.04–0.05), whereas SMOTE increases recall at the expense of precision. This trade-off is quantified by the cost-sensitive metrics in **Table 6**.

Table 6 contains the most operationally consequential finding. An NEC value above 1.0 means the model performs worse than the trivial classifier that always predicts the majority class. Under early-only features, SMOTE pushes NEC

Table 5. Mean (standard deviation) classification performance (feature set B, early-only)

Classifier	Strategy	Precision	Recall	F1	ROC AUC
LR	NR	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.717 (0.043)
LR	SMOTE	0.054 (0.005)	0.717 (0.080)	0.100 (0.010)	0.714 (0.042)
LR	CSW	0.053 (0.005)	0.719 (0.079)	0.100 (0.009)	0.716 (0.041)
RF	NR	0.402 (0.308)	0.046 (0.035)	0.082 (0.060)	0.606 (0.049)
RF	SMOTE	0.052 (0.029)	0.112 (0.063)	0.071 (0.039)	0.605 (0.049)
RF	CSW	0.176 (0.308)	0.014 (0.023)	0.025 (0.041)	0.599 (0.033)
LGBM	NR	0.272 (0.266)	0.042 (0.037)	0.071 (0.062)	0.632 (0.042)
LGBM	SMOTE	0.057 (0.030)	0.105 (0.056)	0.073 (0.038)	0.597 (0.046)
LGBM	CSW	0.063 (0.029)	0.105 (0.053)	0.078 (0.037)	0.614 (0.050)
CB	NR	0.687 (0.448)	0.044 (0.034)	0.082 (0.062)	0.704 (0.041)
CB	SMOTE	0.060 (0.018)	0.211 (0.070)	0.093 (0.028)	0.627 (0.042)
CB	B-SMOTE	0.072 (0.021)	0.197 (0.066)	0.105 (0.031)	0.649 (0.047)
CB	CSW	0.060 (0.012)	0.392 (0.082)	0.105 (0.021)	0.660 (0.037)

Abbreviations: AUC: Area under the curve; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CB: CatBoost; CSW: Cost-sensitive weighting; LGBM: Light gradient boosting machine; LR: Logistic regression; NR: No resampling; RF: Random forest; ROC: Receiver operating characteristic; SMOTE: Synthetic Minority Oversampling Technique.

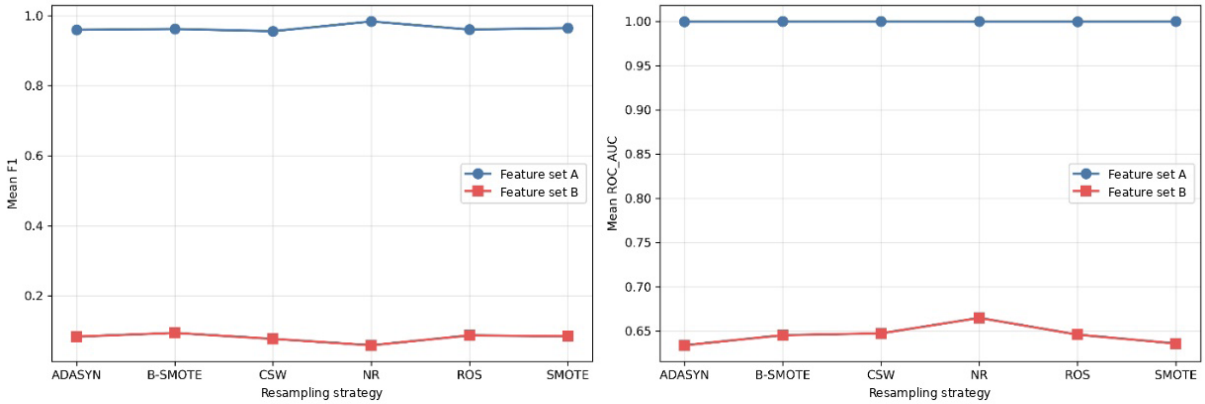

Figure 4. Performance collapse between feature set A and feature set B averaged across all classifiers
Abbreviations: ADASYN: Adaptive synthetic sampling; AUC: Area under the curve; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CSW: Cost-sensitive weighting; NR: No resampling; ROC: Receiver operating characteristic; ROS: Random oversampling; SMOTE: Synthetic Minority Oversampling Technique.

Table 6. Mean (standard deviation) cost-sensitive metrics (feature set B, early-only)

Classifier	Strategy	NEC (5:1)	NEC (10:1)	NEC (20:1)	Brier
LR	NR	1.002 (0.004)	1.001 (0.002)	1.001 (0.001)	0.029 (0.001)
LR	SMOTE	2.800 (0.113)	1.541 (0.078)	0.912 (0.073)	0.217 (0.006)
RF	NR	0.966 (0.036)	0.960 (0.035)	0.957 (0.035)	0.031 (0.001)
RF	SMOTE	1.298 (0.091)	1.093 (0.070)	0.990 (0.064)	0.070 (0.006)
LGBM	NR	0.980 (0.039)	0.969 (0.038)	0.963 (0.037)	0.031 (0.002)
LGBM	SMOTE	1.243 (0.077)	1.069 (0.059)	0.982 (0.055)	0.064 (0.004)
CB	NR	0.957 (0.034)	0.956 (0.034)	0.956 (0.034)	0.028 (0.001)
CB	SMOTE	1.443 (0.108)	1.116 (0.067)	0.953 (0.061)	0.095 (0.006)

Abbreviations: CB: CatBoost; LGBM: Light gradient boosting machine; LR: Logistic regression; NEC: Normalized expected cost; NR: No resampling; RF: Random forest; SMOTE: Synthetic Minority Oversampling Technique.

above 1.0 for every classifier at the 5:1 cost ratio and for three of four classifiers at the 10:1 ratio. In practice, applying SMOTE to these models would yield a risk-detection system worse than doing nothing.

The NR baselines, in contrast, achieve NEC below 1.0 across all tree-based classifiers and cost ratios, indicating marginal but genuine improvement over the trivial baseline. CB with NR achieves the best NEC (10:1) at 0.956, and the lowest Brier score at 0.028. SMOTE's degradation under cost-sensitive metrics is most severe for CB:NEC (10:1), which rises from 0.956 to 1.116, turning a model that marginally beats the trivial classifier into one that is 11.6% worse than doing nothing.

4.6. Shapley additive explanation attribution shifts

The following SHAP analysis is conducted on feature set A. As demonstrated in Sections 4.1 and 4.2, feature set A contains post-resolution variables that introduce target leakage and are unsuitable for deployment. The purpose of this analysis is therefore diagnostic rather than prescriptive: it illustrates how oversampling reorganizes model attribution structure under conditions where a strong predictive signal exists, not which features drive risk in a genuine early-warning setting. Interpretability claims in this

section should be read in that context.

To test whether oversampling alters the features driving model predictions, SHAP values were computed for LGBM under each resampling strategy. **Table 7** reports the Spearman rank correlation between the NR feature importance ordering and that of each other's strategy.

Figure 5 displays SHAP feature importance rankings for LGBM under five resampling conditions. Each horizontal bar represents the mean absolute SHAP value for a single feature, averaged across all test instances across the 50 evaluation folds. Mean |SHAP| values are expressed in log-odds units: a value of 1.0 means that, on average, the feature shifts the model's log-odds prediction by one unit in either direction. Longer bars indicate features that exert greater influence on individual predictions. The x-axis scale differs across panels—note in particular the order-of-magnitude difference between the NR and CSW panels (maximum ≈

Table 7. Spearman's rank correlation between each strategy's mean |SHAP| feature ordering and the NR baseline ordering

Comparison	Spearman's rho	*p*-value	Interpretation
NR vs. SMOTE	0.152	0.676	Major shift
NR vs. B-SMOTE	0.103	0.777	Major shift
NR vs. ADASYN	−0.006	0.987	Major shift
NR vs. CSW	0.952	< 0.001	Stable

Notes: Values near 1.0 indicate preserved importance structure; values near 0 indicate fundamental reorganization.
Abbreviations: ADASYN: Adaptive synthetic sampling; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CSW: Cost-sensitive weighting; NR: No resampling; SHAP: Shapley additive explanation; SMOTE: Synthetic Minority Oversampling Technique.

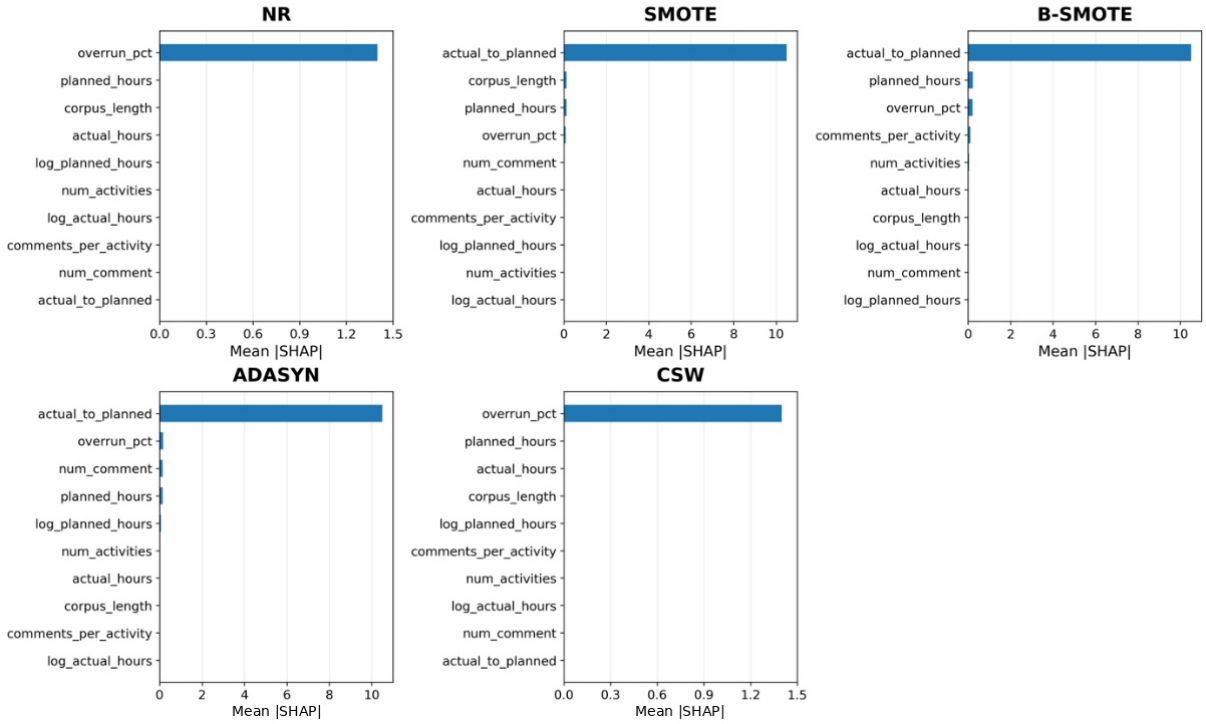

Figure 5. Shapley additive explanation (SHAP) feature importance rankings under five resampling strategies for the light gradient boosting machine
Abbreviations: ADASYN: Adaptive synthetic sampling; B-SMOTE: Borderline Synthetic Minority Oversampling Technique; CSW: Cost-sensitive weighting; NR: No resampling; SMOTE: Synthetic Minority Oversampling Technique.

1.4) and the SMOTE-family panels (maximum ≈ 20–25)—reflecting a fundamental change in model behavior rather than a difference in axis formatting.

The SHAP analysis reveals a stark divergence. All three SMOTE-family strategies produce feature importance orderings that are essentially uncorrelated with the NR baseline (rho = 0.152, 0.103, and −0.006). Under NR, the top feature is overrun_pct (mean |SHAP| = 1.36), followed by planned_hours (0.27) and corpus_length (0.18). Under SMOTE, the dominant feature shifts to actual_to_planned (mean |SHAP| = 10.3), with all other features contributing marginally.

Cost-sensitive weighting, by contrast, preserves the original feature importance structure almost exactly (rho = 0.952, $p < 0.001$). The same features that matter under NR continue to matter under CSW, with nearly identical relative magnitudes.

This finding has direct implications for interpretability. If a practitioner examines the SHAP explanation of an SMOTE-augmented model, the features highlighted as risk drivers will be substantially different from those the model would identify without oversampling. The "explanation" would reflect artifacts of the interpolation process rather than genuine risk patterns in the data.

4.7. Synthetic vs. real minority attribution

Table 8 compares the mean SHAP values for real high-risk issues versus SMOTE-generated synthetic high-risk issues within the SMOTE-augmented model.

The model classifies both real and synthetic minority examples using the same dominant feature (actual_to_planned, SHAP ≈ 10.7), which is expected given this feature's near-deterministic relationship with the label. However, for every other feature, the SHAP attributions for synthetic examples are approximately 40–54% of those for real examples. This means the SMOTE-generated examples are classified based on a narrower set of reasons—they are recognized as high-risk almost exclusively because of the actual-to-planned ratio, whereas real high-risk issues trigger a broader constellation of features.

In an operational setting without the leaked feature, this discrepancy would be more consequential: the model would have learned a simplified, interpolation-driven pattern from synthetic examples rather than the richer, multi-feature pattern present in genuinely risky issues.

Table 8. Mean |SHAP| values for real vs. synthetic minority examples

Feature	Real minority	Synthetic minority	Ratio
actual_to_planned	10.697	10.711	1.00
corpus_length	0.083	0.043	0.52
overrun_pct	0.049	0.021	0.43
planned_hours	0.039	0.019	0.48
num_comment	0.018	0.010	0.54
actual_hours	0.007	0.004	0.54
comments_per_activity	0.003	0.001	0.43
num_activities	0.000	0.000	0.80
log_planned_hours	0.000	0.000	0.40
log_actual_hours	0.000	0.000	0.58

Abbreviation: SHAP: Shapley additive explanation.

5. Discussion

5.1. The target leakage problem

The most consequential finding of this study is not about SMOTE but about the feature set. Feature Set A, which mirrors the features used in prior work, includes actual_hours, overrun_pct, and actual_to_planned — all of which are known only after the issue is resolved and all of which are algebraically related to the risk label. The near-perfect results under feature set A reflect arithmetic relationships between features and the label, not genuine predictive capability.

It is important to note that including post-resolution features is appropriate for retrospective analysis—for example, characterizing the distributional properties of historically risky issues or validating effort-recording practices. Our critique is limited to studies that present such models as early-warning tools intended for deployment at issue-creation time, a framing that presupposes the availability of features that are, by construction, unavailable before the work begins. The claim is not supported by a deployment-valid feature set: a model cannot provide early warning using information that becomes available only after the work is complete.

5.2. Synthetic Minority Oversampling Technique under realistic conditions

When the leaking features are removed (feature set B), the experimental conditions correspond to genuine early warning—predicting risk using only information available at issue-creation time. Under these conditions, the results are unambiguous: SMOTE degrades cost-sensitive performance across all classifiers tested. Under every tree-based classifier tested, SMOTE converts models that marginally outperform the trivial baseline into models that are actively worse than doing nothing—a result that holds consistently across all three cost ratios examined.

This finding extends Song *et al.*'s[7] results to a new domain (effort risk prediction) and a more extreme imbalance regime (2.2% versus the 5–35% range in their study). The convergence is notable: in both defect prediction and effort risk prediction, SMOTE fails to deliver consistent improvement, and its failure is most pronounced when the minority class is rare and the predictive signal is weak.

The mechanism of failure is visible in the precision-recall trade-off. Under feature set B, SMOTE boosts recall (from near zero to 0.10–0.72, depending on the classifier) by generating synthetic high-risk examples that shift the decision boundary toward the majority class. The precision-recall trade-off driving this result was detailed in Section 4.5 (**Table 5**).

5.3. Cost-sensitive weighting as an alternative

Cost-sensitive weighting emerges as a more reliable alternative to SMOTE on two dimensions. First, under feature set A, CSW produces F1 and NEC scores that are comparable to or better than those of SMOTE variants across all tree-based classifiers. Second, and more importantly, the SHAP analysis shows that CSW preserves the original feature importance structure (Spearman rho = 0.952), while all SMOTE-family strategies fundamentally reorganize it (rho $\leq$ 0.152). This means that a CSW-trained model can be interpreted using standard SHAP analysis with confidence that the feature attributions reflect genuine data patterns, whereas a SMOTE-trained model's explanations are unreliable.

Cost-sensitive weighting achieves its effect by reweighting the loss function rather than altering the training data. It does not generate synthetic examples, does not risk interpolating into majority-class regions, and does not require additional library dependencies. For practitioners considering deploying interpretable risk-detection models, CSW is the safer default.

5.4. Practical recommendations

Based on the experimental findings, the following recommendations are offered:

First, any effort risk prediction pipeline must strictly separate features available at prediction time from features known only after resolution.

Second, SMOTE should not be applied by default to effort risk datasets with extreme imbalance (minority rate below 5%). Under realistic feature sets, it degrades cost-sensitive performance and distorts model interpretability. If oversampling is used, it should be treated as an experimental variable rather than a fixed pipeline component and evaluated against the NR baseline using cost-sensitive metrics.

Third, CSW is preferable to oversampling for interpretable risk models. It preserves feature attribution structure, requires no synthetic data generation, and produces comparable or better performance on the metrics that matter operationally.

Fourth, evaluation of effort risk models should always include cost-sensitive metrics (NEC or equivalent) alongside standard F1 and AUC. The present study demonstrates that a model can appear to benefit from SMOTE under standard metrics while simultaneously degrading under cost-sensitive metrics—a discrepancy that F1 alone cannot reveal.

6. Threats to validity

6.1. Internal validity

The primary internal threat is the use of default hyperparameters across all classifiers. While this decision is justified by the goal of isolating resampling effects, it means that performance differences between classifiers may reflect default-parameter quality rather than algorithmic capability. A secondary threat is the choice of k = 5 for SMOTE's nearest-neighbor parameter; different values of k could produce different results. We mitigate this by noting that k = 5 is the standard used in most prior work [6,12], making our results comparable to the existing literature.

The use of default hyperparameters is an acknowledged limitation. While justified by the goal of isolating resampling effects, default

settings may disadvantage certain classifier-resampling combinations relative to their tuned counterparts. Future work should examine whether the resampling rankings observed here persist under systematic hyperparameter optimization.

6.2. External validity

The JOSSE dataset reflects the practices of Apache Foundation projects—large-scale, open-source, predominantly Java-based, with a volunteer-heavy contributor base. Results may not generalize to proprietary software development, small teams, agile-only environments, or domains with substantially different effort distributions, a challenge documented more broadly by Hosseini *et al.*[42] in the context of cross-project defect prediction. The 90th percentile threshold for defining high-risk issues is a modeling choice; different thresholds would yield different imbalance ratios and potentially different resampling effects.

The methodological concern—that class-imbalance handling strategies should be evaluated as experimental variables across realistic feature sets and cost-sensitive metrics, rather than adopted as pipeline defaults—applies beyond software effort prediction. Adjacent domains with comparable extreme imbalance include security alert triage (where alert rates can be below 1%), clinical adverse-event prediction, and equipment failure monitoring in industrial Internet-of-Things settings. Whether the specific rankings observed here—CSW outperforming SMOTE, and SMOTE degrading cost-sensitive performance under weak signal—generalize to those domains is an empirical question warranting separate investigation.

6.3. Construct validity

Effort overrun, as computed from recorded estimates and actuals in Jira, may not faithfully represent project risk. Expert estimates in issue trackers are subject to anchoring, political padding, and default-value usage, which could introduce noise into the risk label. Additionally, actual effort depends on accurate time logging, which is known to be unreliable in open-source projects. These measurement concerns are inherent to the dataset and shared with all prior studies that use JOSSE data.

6.4. Conclusion validity

The evaluation design reduces variance and avoids parametric assumptions, but it cannot fully eliminate uncertainty introduced by the small absolute size of the minority class. However, with only 96 minority-class examples, the effective test set for the high-risk class in each fold is approximately 19, limiting the precision of recall estimates. This is not a limitation of the study design but of the dataset itself, and it reflects the genuine challenge of evaluating classifiers under extreme imbalance, a concern echoed by Shepperd *et al.*[43], who documented systematic researcher bias in the design and reporting of ML-based defect prediction studies.

A practical limitation of the evaluation design is computational cost. The full factorial experiment (6 strategies × 4 classifiers × 2 feature sets × 50 folds) required approximately 4.5 h on a standard CPU without parallelization, with CB training time dominating. Scaling to larger datasets or additional resampling conditions would require GPU acceleration or parallel fold execution. For practitioners wishing to replicate the design on larger repositories, 5 × 5 cross-validation would substantially reduce runtime while retaining adequate variance control.

7. Conclusion

This study presents a controlled investigation into whether SMOTE and alternative resampling strategies improve the detection of effort overrun risk in software projects or introduce noise that distorts classifier behavior. By treating oversampling as an independent experimental variable rather than a fixed pipeline component, the study provides the first direct evidence on a question that the software analytics community has assumed rather than tested.

The experimental design—six resampling conditions, four classifiers, two feature sets, both standard and cost-sensitive metrics, with SHAP-based attribution analysis—is deliberately comprehensive, reflecting the lesson from Song *et al.*[7] that narrow evaluations with one or two datasets can produce misleading conclusions.

Regardless of the specific numerical outcomes, the methodological contribution is clear: future effort risk studies should not adopt SMOTE by default. Instead, the resampling strategy should be selected through controlled comparison on

the target data, evaluated under cost-sensitive metrics that reflect the operational context, and validated through feature-attribution analysis to confirm that the model has not learned artefactual patterns. This principle applies equally to defect prediction, reliability modeling, and other software engineering tasks where class imbalance is the norm rather than the exception.

Acknowledgments

The authors would like to thank Transilvania University of Brașov for the institutional support provided for the study.

Funding

This study received no external funding. The publication fee for the article was borne by Transilvania University of Brașov.

Conflict of interest

The authors declare no conflict of interest.

Author contributions

Conceptualization: Andreea-Elena Catana
Methodology: All authors
Formal analysis: All authors
Writing – original draft: Andreea-Elena Catana
Writing – review & editing: All authors

Availability of data

Data are available upon reasonable request to the corresponding author.

AI tools statement

All authors confirm that no AI tools were used in the preparation of this manuscript.

References

1. Tran TN, Tran HT, Nguyen QN. Leveraging AI for Enhanced Software Effort Estimation: A Comprehensive Study and Framework Proposal. In: Proceedings of the 2023 International Conference on Cognitive Computing and Complex Data (ICCD), Huaian, China, 2023:284–289.

   https://doi.org/10.1109/ICCD59681.2023.10420603

2. Alhamed M, Storer T. Evaluation of Context-Aware Language Models and Experts for Effort Estimation of Software Maintenance Issues. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), Limassol, Cyprus, 2022:129–138.

   https://doi.org/10.1109/ICSME55016.2022.00020

3. Kotsiantis S, Kanellopoulos D, Pintelas P. Handling imbalanced datasets: A review. *GESTS Int Trans Comput Sci Eng.* 2006;30:25–36. April 6, 2026. https://www.researchgate.net/publication/228084509_Handling_imbalanced_datasets_A_review

4. He H, Garcia EA. Learning from Imbalanced Data. *IEEE Trans Knowl Data Eng.* 2009;21(9):1263–1284.

   https://doi.org/10.1109/TKDE.2008.239

5. Fernández A, García S, Galar M, Prati RC, Krawczyk B, Herrera F. *Learning from Imbalanced Data Sets*. Cham: Springer; 2018:19-46.

   https://doi.org/10.1007/978-3-319-98074-4

6. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic Minority Over-sampling Technique. *J Artif Intell Res.* 2002;16:321–357.

   https://doi.org/10.1613/jair.953

7. Song Q, Guo Y, Shepperd M. A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction. *IEEE Tran Softw Eng.* 2019;45(12):1253–1269.

   https://doi.org/10.1109/TSE.2018.2836442

8. Jude A, Uddin J. Explainable Software Defects Classification Using SMOTE and Machine Learning. *Ann Emerg Technol Comput.* 2024;8(1):36–49.

   https://doi.org/10.33166/AETiC.2024.01.004

9. Ali SS, Ren J, Zhang K, Wu J, Liu C. Heterogeneous Ensemble Model to Optimize Software Effort Estimation Accuracy. *IEEE Access*. 2023;11:27759–27792.

   https://doi.org/10.1109/ACCESS.2023.3256533

10. ForouzeshNejad AA, Arabikhan F, Aheleroff S. Optimizing Project Time and Cost Prediction Using a Hybrid XGBoost and Simulated Annealing Algorithm. *Machines*. 2024;12(12):867.

    https://doi.org/10.3390/machines12120867

11. Ferhati K, Burlea-Schiopoiu A, Nascu AG. A Text-Based Project Risk Classification System Using Multi-Model AI: Comparing SVM, Logistic Regression, Random Forests, Naive Bayes, and XGBoost. *Systems*. 2025;13(12):1078.

    https://doi.org/10.3390/systems13121078

12. Fernández A, García S, Herrera F, Chawla NV. SMOTE for learning from imbalanced data:

progress and challenges, marking the 15-year anniversary. *J Artif Intell Res.* 2018;61:863–905.

https://doi.org/10.1613/jair.1.11192

13. Aflaha R, Herteno R, Faisal MR, Abadi F, Saputro S. Effect of SMOTE Variants on Software Defect Prediction Classification Based on Boosting Algorithm. *J Ilm Tek Elektro Komput Inform.* 2024;10(2):201–216.

https://doi.org/10.26555/jiteki.v10i2.28521

14. Imani M, Beikmohammadi A, Arabnia HR. Comprehensive Analysis of Random Forest and XGBoost Performance with SMOTE, ADASYN, and GNUS Under Varying Imbalance Levels. *Technologies.* 2025;13(3):88.

https://doi.org/10.3390/technologies13030088

15. Alhamed M, Storer T. JOSSE: A Software Development Effort Dataset Annotated with Expert Estimates, Version 1. 2025.

https://doi.org/10.5281/zenodo.7022735

16. Mende T, Koschke R. Revisiting the evaluation of defect prediction models. In: *Promise'09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, New York, NY, USA:ACM; 2009; 7, 1–10. Accessed April 21, 2026. https://promisedata.org/pdf/2009/06_Mende.pdf

17. Hall T, Beecham S, Bowes D, Gray D, Counsell S. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Trans Softw Eng.* 2012;38(6):1276–1304.

https://doi.org/10.1109/TSE.2011.103

18. Blagus R, Lusa L. SMOTE for high-dimensional class-imbalanced data. *BMC Bioinform.* 2013;14(1):106.

https://doi.org/10.1186/1471-2105-14-64

19. Kovács, G. SMOTE-variants: A Python implementation of 85 minority oversampling techniques. *Neurocomputing.* 2019;366:352–360.

https://doi.org/10.1016/j.neucom.2019.06.100

20. Han H, Wang WY, Mao BH. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In *Advances in Intelligent Computing (ICIC)*, Lecture Notes in Computer Science, Berlin: Springer; 2005;3644:878–887.

https://doi.org/10.1007/11538059_91

21. He H, Bai Y, Garcia EA, Li S. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. In: *Proceedings of the IEEE International Joint Conference on Neural Networks*, Hong Kong: IEEE; 2008:1322–1328.

https://doi.org/10.1109/IJCNN.2008.4633969

22. Nguyen HM, Cooper EW, Kamei K. Borderline over-sampling for imbalanced data classification. *Int J Knowl Eng Soft Data Paradig.* 2011;3(1):4–21.

https://doi.org/10.1504/IJKESDP.2011.039875

23. Ghinaya H, Hereno R, Faisal MR, Farmadi A, Indriani F. Analysis of Important Features in Software Defect Prediction Using SMOTE, RFE and Random Forest. *J Electron Electromed Eng Med Inform.* 2024;6(3):276-288.

https://doi.org/10.35882/jeeemi.v6i3.453

24. Jørgensen M. A review of studies on expert estimation of software development effort. *J Syst Softw.* 2014;70(1–2):37–60.

https://doi.org/10.1016/S0164-1212(02)00156-5

25. Idri A, Hosni M, Abran A. Systematic mapping study of ensemble effort estimation. *J Inf Softw Technol.*2016;118:151-175.

https://doi.org/10.1016/j.jss.2016.05.016

26. Herbold S. On the costs and profit of software defect prediction. *IEEE Trans Softw Eng.* 2021;47(11):2617–2631.

https://doi.org/10.1109/TSE.2019.2957794

27. Wen J, Li S, Lin Z, Hu Y, Huang C. Systematic literature review of machine learning based software development effort estimation models. *Inf Softw Technol.* 2012;54(1):41–59.

https://doi.org/10.1016/j.infsof.2011.09.002

28. Jadhav A, Shandilya K. Reliable machine learning models for estimating effective software development efforts: A comparative analysis. *J Eng Res.* 2023;11(4):362–376.

https://doi.org/10.1016/j.jer.2023.100150

29. Malhotra R, Jain A. Software effort prediction using statistical and machine learning methods. *Int J Adv Comput Sci Appl.* 2011;2(1).

https://doi.org/10.14569/IJACSA.2011.020122

30. Boehm BW, Abts C, Brown AW, et al. *Software Cost Estimation with COCOMO II*; Upper Saddle River, NJ, USA: Prentice Hall; 2009. https://dl.acm.org/doi/10.5555/1795822

31. Kumar PS, Behera HS. Estimating software effort using neural network: An experimental investigation. In *Computational Intelligence in Pattern Recognition*; Singapore: Springer; 2020:1120:165–180.

https://doi.org/10.1007/978-981-15-2449-3_14

32. Weinan T. Application of support vector machine

system introducing multiple submodels in data mining. *Syst Soft Comput.* 2024;6:200096.

https://doi.org/10.1016/j.sasc.2024.200096

33. Matloob F, Ghazal T, Taleb N. Software Defect Prediction using Ensemble Learning: A Systematic Literature Review. *IEEE Access.* 2021;9:98754–98771.

https://doi.org/10.1109/ACCESS.2021.3095559

34. Kumar H, Saxena V. Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study. *J Softw Eng Appl.* 2024;17(4):155–171.

https://doi.org/10.4236/jsea.2024.174009

35. Ke G, Meng Q, Finley T, et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Red Hook, NY, USA: Curran Associates Inc.; 2017:3149–3157.

https://doi.org/10.52202/068431-1367

36. Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. CatBoost: unbiased boosting with categorical features. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Red Hook, NY, USA: Curran Associates Inc.; 2018:6639–6649. https://dl.acm.org/doi/abs/10.5555/3327757.3327770

37. Gao K, Khoshgoftaar TM, Wang H, Seliya N. Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Softw PractExp.* 2011;41(5):579–606.

https://doi.org/10.1002/spe.1043

38. Drummond C, Holte RC. Cost Curves: An Improved Method for Visualizing Classifier Performance. *Mach Learn.* 2006;65(1):95–130.

https://doi.org/10.1007/s10994-006-8199-5

39. Lundberg SM, Lee SI. A Unified Approach to Interpreting Model Predictions. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Red Hook, NY, USA: Curran Associates Inc.; 2017:4768–4777. Accessed April 18, 2026. https://dl.acm.org/doi/10.5555/3295222.3295230

40. Demšar J. Statistical Comparisons of Classifiers over Multiple Data Sets. *J Mach Learn Res.* 2006;7:1–30. Accessed April 18, 2026. https://jmlr.org/papers/volume7/demsar06a/demsar06a.pdf

41. Romano J, Kromrey JD, Coraggio J, Skowronek J. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and similar surveys? In: *Proceedings of the Annual Meeting of the Florida Association of Institutional Research*, Cocoa Beach, FL, USA. 2006:1-3. Accessed April 21, 2026. https://www.researchgate.net/publication/237544991_Appropriate_Statistics_for_Ordinal_Level_Data_Should_We_Really_Be_Using_t-test_and_Cohen's_d_for_Evaluating_Group_Differences_on_the_NSSE_and_other_Surveys

42. Hosseini S, Turhan B, Gunarathna D. A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction. *IEEE Trans Softw Eng.* 2019;45(2):111–147.

https://doi.org/10.1109/TSE.2017.2770124

43. Shepperd M, Bowes D, Hall T. Researcher Bias: The Use of Machine Learning in Software Defect Prediction. *IEEE Trans Softw Eng.* 2014;40(6):603–616.

https://doi.org/10.1109/TSE.2014.2322358

Andreea-Elena Catana *is currently a PhD student in Engineering and Management, with research focused on the application of artificial intelligence in project management, software development, and risk management. She is also working as a Project Manager in Software development, where she manages IT-related projects for clinical studies and contributes to the implementation of digital solutions in regulated environments. She has several years of professional experience in project coordination, stakeholder management, and software-related processes within the healthcare and clinical research sectors. Her current research interests include artificial intelligence, machine learning, predictive analytics, software project optimization, and early risk identification in project environments. Her doctoral work investigates AI-driven approaches for improving project performance and supporting decision-making in software project management.* http://orcid.org/0009-0008-5458-6920

Adriana Florescu *is a Professor and PhD Supervisor in Engineering and Management at Transilvania University of Brasov, Romania, within the Faculty of Technological Engineering and Industrial Management. His primary areas of expertise include integrated production systems, digital manufacturing, design and optimization, and the modeling and simulation of Flexible Manufacturing Systems (FMS). His research focuses on the real-time monitoring and control of manufacturing processes, the implementation of In-*

dustry 4.0 technologies, digital twins, and the integration of machine learning and AI within intelligent production systems and operations management. As the Director of the "Economic Engineering and Production Systems" Research Center at the Research and Development Institute of Transilvania University of Brasov, he has spearheaded complex, interdisciplinary research initiatives. His work emphasizes the strategic association of competence units—including universities and research organizations—leveraging advanced computing platforms, experimental research facilities, and technology transfer to advance the field of Smart Manufacturing and real-time factory optimization. https://orcid.org/0000-0003-0374-7022

An International Journal of Optimization and Control: Theories & Applications (https://accscience.com/journal/ijocta)