



Cofinanțat de
Uniunea Europeană

HUB Român de Inteligență Artificială
Cod SMIS: 351416



ET-SDP: Enhancing code embeddings with effort-related and test coverage metrics for improved software defect prediction

Ioana-Gabriela Chelaru, Gabriela Czibula, Zuzsanna Oneț-Marian,
Istvan-Gergely Czibula

Babeș-Bolyai University

June 5, 2026

What is Software Defect Prediction (SDP)?

Problem Statement

Proposed Methodology

Results and Discussion

Comparison to Related Work

- SDP: Detecting software modules likely to contain faults in new releases.
- Essential for software maintenance, reliability prediction, and guiding testing activities.
- Traditional approaches rely on **static code metrics** (complexity, coupling, cohesion) [1].
- **Gap:** Process-oriented metrics (development effort, test coverage) are underexplored for SDP [2, 3].

What is Software Defect Prediction (SDP)?

Problem Statement

Proposed Methodology

Results and Discussion

Comparison to Related Work

- SDP: Detecting software modules likely to contain faults in new releases.
- Essential for software maintenance, reliability prediction, and guiding testing activities.
- Traditional approaches rely on **static code metrics** (complexity, coupling, cohesion) [1].
- **Gap:** Process-oriented metrics (development effort, test coverage) are underexplored for SDP [2, 3].

What is Software Defect Prediction (SDP)?

Problem Statement

Proposed Methodology

Results and Discussion

Comparison to Related Work

- SDP: Detecting software modules likely to contain faults in new releases.
- Essential for software maintenance, reliability prediction, and guiding testing activities.
- Traditional approaches rely on **static code metrics** (complexity, coupling, cohesion) [1].
- **Gap:** Process-oriented metrics (development effort, test coverage) are underexplored for SDP [2, 3].

What is Software Defect Prediction (SDP)?

Problem Statement

Proposed Methodology

Results and Discussion

Comparison to Related Work

- SDP: Detecting software modules likely to contain faults in new releases.
- Essential for software maintenance, reliability prediction, and guiding testing activities.
- Traditional approaches rely on **static code metrics** (complexity, coupling, cohesion) [1].
- **Gap:** Process-oriented metrics (development effort, test coverage) are underexplored for SDP [2, 3].

- Software effort estimation (SEE) and SDP share relevant features (e.g., code complexity).
- **Effort-related metrics** capture how code evolves: change frequency [4], bug-fix history [5], static analysis warnings [6].
- **Test coverage metrics** directly measure how well code is validated [7].
- **Hypothesis:** Combining effort-related and test coverage metrics creates more informative code embeddings for defect prediction [8].

- Software effort estimation (SEE) and SDP share relevant features (e.g., code complexity).
- **Effort-related metrics** capture how code evolves: change frequency [4], bug-fix history [5], static analysis warnings [6].
- **Test coverage metrics** directly measure how well code is validated [7].
- **Hypothesis:** Combining effort-related and test coverage metrics creates more informative code embeddings for defect prediction [8].

- Software effort estimation (SEE) and SDP share relevant features (e.g., code complexity).
- **Effort-related metrics** capture how code evolves: change frequency [4], bug-fix history [5], static analysis warnings [6].
- **Test coverage metrics** directly measure how well code is validated [7].
- **Hypothesis:** Combining effort-related and test coverage metrics creates more informative code embeddings for defect prediction [8].

- Software effort estimation (SEE) and SDP share relevant features (e.g., code complexity).
- **Effort-related metrics** capture how code evolves: change frequency [4], bug-fix history [5], static analysis warnings [6].
- **Test coverage metrics** directly measure how well code is validated [7].
- **Hypothesis:** Combining effort-related and test coverage metrics creates more informative code embeddings for defect prediction [8].

Research Questions

ET-SDP

Chelaru, Czubala,
Oneț-Marian,
Czubala

Problem
Statement

Proposed
Methodology

Results and
Discussion

Comparison to
Related Work

- **RQ1:** How to extract meaningful effort-related and test coverage metrics to enrich the feature-based representation of application classes for SDP?
- **RQ2:** How qualitative are the proposed embeddings in uncovering patterns correlated with defect-proneness?
- **RQ3:** Are the effort-related and test coverage metrics beneficial for SDP compared to software metrics previously proposed in the literature?

Case Studies

Two Apache projects from the Herbold dataset [1] (4189 software metrics per class):

- **Apache Calcite** [9]: 15 releases (1.1.0–1.15.0), data management framework
- **Apache Ant-Ivy** [10]: 6 releases (1.4.1–2.4.0), transitive package manager

Defective rates: 3.3%–16.6% (Calcite), 2.9%–14.2% (Ant-Ivy) → severe class imbalance.

ET-SDP: Overview

ET-SDP

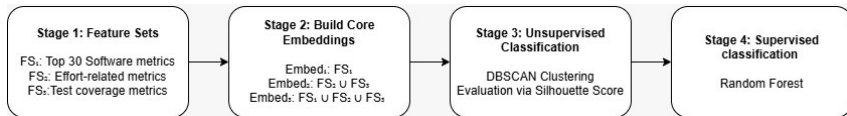
Chelaru, Czubala,
Oneț-Marian,
Czubala

Problem
Statement

Proposed
Methodology

Results and
Discussion

Comparison to
Related Work



Four stages: (1) Feature set construction, (2) Code embeddings, (3) Unsupervised validation, (4) Supervised SDP.

Stage 1: Feature Sets (RQ1)

- $\mathcal{FS}_1$: **Top-30 software metrics** selected via clustering relevance ranking from 2859 non-zero features [1].
- $\mathcal{FS}_2$: **Effort-related metrics** from 5 categories:
 - CHANGE_TYPE, HASSAN, ISSUE, MOSER, PMD
 - 170 features (Calcite), 149 features (Ant-Ivy)
 - $\mathcal{FS}_2_common_26$: 26 features with correlation ≥ 0.1 in both projects
- $\mathcal{FS}_3$: **5 test coverage metrics** from JaCoCo [7] (instruction, branch, line, complexity, method).

Stage 2: Code Embeddings

Three embedding variants for representing application classes:

- 1 Embed₁: $\mathcal{FS}_1$ (top-30 software metrics)
- 2 Embed₂: $\mathcal{FS}_2 \cup \mathcal{FS}_3$ (effort + coverage only)
- 3 Embed₃: $\mathcal{FS}_1 \cup \mathcal{FS}_2 \cup \mathcal{FS}_3$ (all combined)

For $\mathcal{FS}_2$, two variants: project-specific ($\mathcal{FS}_{2_170}$ / $\mathcal{FS}_{2_149}$) and cross-project ($\mathcal{FS}_{2_common_26}$).

Stages 3 & 4: Evaluation Methodology

Unsupervised (Stage 3): DBSCAN clustering [11]

- Internal: Silhouette Score, Davies-Bouldin Index
- External: Homogeneity, Completeness, V-measure

Supervised (Stage 4): Random Forest classifier

- *Chronological expanding-window CV*: train on releases $1..k$, test on $k + 1$
- 14 folds (Calcite), 5 folds (Ant-Ivy)
- Metrics: Precision, Recall (POD), F1, AUC, AUPRC
- Class weights inversely proportional to class frequencies

Clustering Results (RQ2)

ET-SDP

Chelaru, Czubala,
Oneț-Marian,
Czubala

Problem
Statement

Proposed
Methodology

Results and
Discussion

Comparison to
Related Work

Dataset	Embedding	Variant	Silhouette $\uparrow$	DB $\downarrow$	Homog. $\uparrow$	Compl. $\uparrow$	V-meas. $\uparrow$
	Embed ₁	–	0.932	0.177	0.024	0.025	0.024
	Embed ₂	$\mathcal{FS}_2_{-170}$	0.518	0.462	0.067	0.040	0.050
Calcite		$\mathcal{FS}_2_{-c.26}$	0.478	0.769	0.043	0.065	0.052
	Embed ₃	$\mathcal{FS}_2_{-170}$	0.565	0.417	0.065	0.045	0.053
	Embed ₁	–	0.987	0.012	0.001	0.012	0.001
Ant-Ivy	Embed ₂	$\mathcal{FS}_2_{-149}$	0.569	0.485	0.016	0.025	0.019
	Embed ₃	$\mathcal{FS}_2_{-149}$	0.566	0.487	0.016	0.025	0.019

Key finding: Effort-related embeddings achieve better alignment with defect labels (higher V-measure) despite lower cluster separation.

Supervised Classification Results (RQ3)

Dataset	Embedding	Variant for $\mathcal{FS}_2$	Prec	POD	F1	AUC	AUPRC
Calcite	Embed ₁	–	0.386	0.297	0.321	0.632	0.253
	Embed ₂	$\mathcal{FS}_2$ _170	0.847	0.493	0.610	0.932	0.720
	Embed ₃	$\mathcal{FS}_2$ _170	0.859	0.497	0.614	0.933	0.723
Ant-Ivy	Embed ₁	–	0.355	0.338	0.299	0.587	0.205
	Embed ₂	$\mathcal{FS}_2$ _149	0.461	0.263	0.300	0.883	0.450
	Embed ₃	$\mathcal{FS}_2$ _149	0.434	0.250	0.274	0.882	0.433

Effort + coverage metrics **dramatically outperform** software metrics: AUC improves from 0.632 to 0.932 (Calcite) and from 0.587 to 0.883 (Ant-Ivy).

Feature Importances & Ablation Study

Top RF feature importances (chronological CV, Calcite):

- 1 COV_BRANCH (0.072)
- 2 COV_COMPLEXITY (0.055)
- 3 COV_INSTRUCTION (0.053)
- 4 COV_LINE (0.052)
- 5 COV_METHOD (0.041)
- 6 MOSER_age (0.035)

All 5 coverage features rank in the top 5 → test coverage is highly predictive.

Ablation: Removing bug-related features has negligible impact ($\leq 0.1\%$ AUC), confirming that the performance is not driven by potential label leakage.

Comparison to Related Work

- **vs. full Herbold SM set [1]** (2859 features):
 - Comparable clustering V-measure (0.053 vs. 0.051)
 - With 93% fewer features (205 vs. 2859)
- **vs. SVC [12]** (4189 SM features, same CV methodology):
 - Precision: +92.1% (0.859 vs. 0.447)
 - AUC: +19.3% (0.933 vs. 0.782)
 - F1: +17.8% (0.614 vs. 0.521)
- Consistent with findings of Al-Shaaby et al. [3] on Apache Lucene, confirming the utility of coverage metrics for SDP.

- ET-SDP: effort-related and test coverage metrics create more informative embeddings for SDP.
- Adding software metrics to effort features brings **no significant improvement**.
- Test coverage metrics are the **strongest predictors** of defect proneness.
- Validated on two Apache projects with chronological cross-validation.
- **Future work:**
 - Combine with semantic features (CodeBERT)
 - Extend to additional projects
 - Use estimated effort as a predictive feature [13]

Bibliography



S. Herbold, A. Trautsch, F. Trautsch, and B. Ledel, "Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection," *Empirical Software Engineering*, vol. 27, Jan 2022.



P. S. Kochhar, D. Lo, J. Lawall, and N. Nagappan, "Code coverage and postrelease defects: A large-scale study on open source projects," *IEEE Transactions on Reliability*, vol. 66, no. 4, pp. 1213–1228, 2017.



B. Al-Ahmad, "Using code coverage metrics for improving software defect prediction.," *J. Softw.*, vol. 13, no. 12, pp. 654–674, 2018.



R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, pp. 181–190, 2008.



A. E. Hassan, "Predicting faults using the complexity of code changes," in *2009 IEEE 31st International Conference on Software Engineering*, pp. 78–88, 2009.



GitHub, "PMD - An extensible cross-language static code analyzer," 2023.
<https://pmd.github.io/>.



“Jacoco - java code coverage,” 2026.
<https://www.jacoco.org/>.



I.-G. Chelaru, G. Czubala, Z. Oneț-Marian, and I.-G. Czubala, “ET-SDP: Enhancing code embeddings with effort-related and test coverage metrics for improved software defect prediction,” *Acta Universitatis Sapientiae, Informatica*, p. submitted, 2026.



E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire, “Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources,” in *Proceedings of SIGMOD ’18*, (New York, NY, USA), pp. 221–230, ACM, 2018.



S. Loughran and E. Hatcher, *Ant in Action*.
Manning Publications, 2007.



K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, “DBSCAN: Past, present and future,” in *Proceedings of ICADIWT 2014*, pp. 232–238, 2014.



G. Ciubotariu, G. Czubala, I. G. Czubala, and I.-G. Chelaru, “Uncovering behavioural patterns of one- and binary-class SVM-based software defect predictors,” in *Proceedings of ICSOFT 2023*, pp. 249–257, SciTePress, 2023.



I.-G. Chelaru, “Enhancing the performance of software effort estimation through boosting ensemble learning,” in *Proceedings of the 25th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2023)*, p. accepted, 2023.

ET-SDP

Chelaru, Czubala,
Oneț-Marian,
Czubala

Problem
Statement

Proposed
Methodology

Results and
Discussion

Comparison to
Related Work

Thank you for your attention!

Financial support from:

“Romanian Hub for Artificial Intelligence - HRIA”, Smart Growth, Digitization, and Financial Instruments Program, 2021–2027, MySMIS no. 351416.