Intelligent Human Systems Integration (IHSI), Vol. 200, 2026, 102–110
https://doi.org/10.54941/ahfe1007069

Sinusoidal Time-Based Features and Human Error Metrics: Advancing Software Defect Prediction in Safety-Critical Systems

Carlos Andrés Ramírez Cataño[1] and Makoto Itoh[2]

[1]Doctoral Program in Risk and Resilience Engineering, Graduate School of Science and Technology, University of Tsukuba, Tsukuba, 305-8573, Japan
[2]Institute of Systems and Information Engineering, University of Tsukuba, Tsukuba, 305-8573, Japan

ABSTRACT

Defect detection in safety-critical software remains difficult despite advanced tools and mature quality assurance, largely due to the human origins of many errors. Building on prior work introducing human error–driven metrics that outperform traditional code measures, this study enhances predictive accuracy by prioritizing higher recall to strengthen defect triage in environments where missing defects carries severe risk and moderate false positives are acceptable. We integrate temporal cyclicality into defect prediction by transforming code commit timestamps into sine features via a parameterized sinusoidal model, optimized with a genetic algorithm to capture daily and periodic developer activity patterns. These features preserve non-linear, cyclical relationships linked to defect introduction, allowing machine learning models to exploit latent human-behavioural signals. Evaluation across three open-source safety-critical systems shows average recall gains of 48.68% over code metrics baselines and 9.27% over previously defined human error metrics. Embedding periodic human activity patterns alongside human-error features significantly improves defect prediction. The approach is interpretable, and generalizable, offering a pathway for broader application and future integration with adaptive, human-centric software quality models.

Keywords: Computer bugs, Human factors, Software defect prediction, Software development, Software quality, Software testing

INTRODUCTION

Software defect prediction (SDP) models have been extensively researched over the past five decades due to their potential to improve the quality and security posture of software systems (Akiyama, 1971; Bieman, 1997; Menzies et al., 2007). Despite decades of progress in automated techniques for identifying defects and vulnerabilities, faults in critical systems remain common. A significant contributor is human error (Reason, 1990), often resulting in context-dependent defects—instances where a technically correct implementation fails in its operational setting, such as mismatched measurement units. Although SDP aims to forecast defects before their occurrence, its practical effectiveness is limited by the low explainability

Received November 10, 2025; Revised December 2, 2025; Accepted December 17, 2025; Available online February 1, 2026

of many models (Gunning, 2017), which constrains actionable feedback to developers, and by the enduring challenge of detecting errors driven by human factors (Li et al., 2024; Lewis et al., 2013).

Our earlier work introduced a human-error–driven framework with method-level metrics designed for defect prediction. These metrics yielded superior performance over state-of-the-art code and history measures, while enhancing model explainability and practical applicability—shifting prediction tools from diagnostic aids to actionable decision-support systems (Ramirez et al., 2025). In this study, we extend that framework to evaluate its real-world applicability within safety-critical software systems.

This research builds on our previous work to investigate automated feature generation aimed at increasing recall (true positive rate) in defect prediction models. This initial exploration seeks to enhance predictive accuracy by prioritizing higher recall to support defect triage in safety-critical systems, where overlooking defects carries severe risks and tolerating moderate false positives is acceptable.

To achieve our research objectives, we conducted empirical experiments guided by the following research question:

RQ: To what extent does a custom-engineered, human-factors-based feature enhance the recall of software defect prediction models compared with state-of-the-art code- and history-metrics models, and how does its performance compare to our prior method-level ML defect prediction model?

This question examines whether integrating a custom-engineered human factors–based feature (e.g., developer behavioral or temporal patterns) can measurably improve the model's ability to identify defective methods, achieving higher recall than the baseline and thereby strengthening defect triage in safety-critical software systems.

The following findings have been acquired through empirical evaluation across three critical open-source projects: (1) The sinusoidal equation is an effective basis for prediction metrics that capture temporal cyclicality and reflect human behavioural patterns; (2) It achieves a significantly higher recall than code and history metrics (+48.68%), providing greater defect triage coverage in safety-critical domains where a small number of false positives is acceptable but missing true positives is not.

STUDY DESIGN

This section describes the methodology for developing, training, testing, and evaluating the software SDP models, and outlines the process used for engineering the proposed metric.

Human Error-Based Framework

Our previous work (Ramirez et al., 2025) demonstrated the effectiveness of applying human factors theory to software defect prediction, with metrics derived from this theory consistently outperforming state-of-the-art alternatives. Beyond predictive accuracy, these metrics offer enhanced actionability, practicality, and model explainability.

Building on this foundation, the present study extends the framework to evaluate its capability to integrate custom-engineered metrics aimed at increasing true positive prediction rates. The focus is on identifying practical processes to enhance software quality in safety-critical open-source systems. Core cognitive features from prior work—Alertness and Memory Decay—are retained to guide model predictions, ensuring methodological continuity while addressing current defect prediction tasks.

Feature Engineering for Enhanced True Positive Rate

In this study, we use the sinusoidal equation, $y = A \sin [B(x–C)] + D$, as the basis for generating a custom-engineered metric designed to capture temporal cyclicality in developer activity. The equation models periodic phenomena through four parameters: amplitude (A), frequency (B), phase shift (C), and vertical shift (D), with x representing the commit time of a software change. This allows us to preserve cyclical patterns in human behavior that may correlate with defect introduction, enriching the predictive feature space. To adapt the equation to observed commit-time distributions, we apply a simple genetic algorithm guided by a fitness function that prioritizes improvements in recall, evolving the parameters A,B,C, and D. This optimization ensures the transformation accurately represents temporal periodicity while maintaining continuity in the cyclic domain. Unlike treating time as a straightforward numeric or categorical variable, the sinusoidal representation enables machine learning models to learn non-linear, periodic correlations that would otherwise be obscured in linear feature mappings.

Mutation

The mutation process was implemented using a simple genetic algorithm incorporating both mutation and crossover operations. Each individual was represented by four parameters—amplitude, frequency, phase shift, and vertical shift—corresponding to the sinusoidal feature definition. Given that the fitness function required executing a Random Forest prediction and calculating recall for every mutation, the optimization process was computationally expensive. To ensure feasibility, the search space was constrained to a small number of mutations, running for a maximum of 10 generations and capped at 800 total fitness evaluations. Following the constrained evolutionary optimization, the genetic algorithm produced the optimal sinusoidal metric parameters for each target project, as shown in Table 1.

Table 1: Optimal sinusoidal metric parameters evolved through the genetic algorithm for each target project.

Project	Amplitude	Frequency	Phase Shift	Vertical Shift
Astrobee	120.04	0.13	308.98	241.40
MMGIS	172.05	0.05	216.62	69.16
Openpilot	159.77	0.02	–335.25	31.47

Limitations of the Sinusoidal Model

A primary limitation of the basic sinusoidal model is its assumption of perfectly symmetric, wave-like behaviour. In practice, many temporal patterns, including developer activity in software commits, are asymmetric. For example, the peak in commit activity during the workday may be followed by a gradual decline into the evening, while the rise from low activity in the early morning to the midday peak may occur more rapidly.

Prediction Model

For defect prediction at the function level, we adopt the Random Forest (RF) algorithm, aligning with its established effectiveness in prior SDP research (Mo et al., 2022). RF is a robust ensemble method that constructs multiple decision trees during training and, for classification, derives the final output through a majority vote across trees. Its ability to handle large, high-dimensional datasets and its resilience to overfitting make it well-suited for our study.

The implementation employs the sklearn.ensemble module from the scikit-learn Python library. We configure the model with 100 decision trees (n_estimators=100), unrestricted tree depth (max_depth=None), and the 'balanced' class weight option to mitigate issues related to dataset imbalance.

Model Validation and Performance Metrics

Although the human factors metrics used in this study were assessed in prior work, we re-validate our models to account for enhancements introduced through the custom feature engineering strategy. Model performance is measured using ten-fold cross-validation, a widely adopted approach in SDP research (Stone, 1974), which offers a robust and unbiased estimate of predictive capability.

To quantify effectiveness, we employ the F1 score (Van, 1979) and Matthew's Correlation Coefficient (MCC) (Matthews, 1975). These metrics provide a balanced assessment, particularly for imbalanced datasets, by considering false positives and false negatives alongside true positives. This comprehensive evaluation framework not only verifies overall predictive quality but also supports our core objective of maximizing recall—ensuring a higher rate of correctly identified defects, which is critical in safety-sensitive environments where missing a defect carries significant risk.

EVALUATION

Subjects

This study examines three open-source software systems—NASA Astrobee Robot Software, the Multi-Mission Geographical Information System (MMGIS), and openpilot—which are deployed in real-world safety-critical environments. Their operational domains, ranging from autonomous robotics in space to planetary science mission planning and functional-safety-compliant automotive driver assistance, provide relevant and diverse contexts for

evaluating our defect prediction approach. Studying these systems allows us to assess the effectiveness of our proposed metrics and feature engineering methods under conditions where software reliability is paramount.

NASA Astrobee Robot Software. The NASA Astrobee system comprises three free-flying robots that have been operating aboard the International Space Station (ISS) since 2019. The corresponding Astrobee repository contains the source code for the Astrobee Robot Software, including the flight software executed onboard the robots, a high-fidelity software simulator, and various supporting tools. The codebase is primarily implemented in C++.

- Repository: https://github.com/nasa/astrobee

Multi-Mission Geographical Information System (MMGIS). The Multi-Mission Geographical Information System (MMGIS) is a web-based mapping and spatial data infrastructure designed to support planetary science operations. It forms part of NASA's Advanced Multi-Mission Operations System.

- Repository: https://github.com/NASA-AMMOS/MMGIS

Openpilot. Openpilot is an open-source operating system for robotics, designed primarily to enhance driver assistance capabilities in over 300 supported vehicle models. The system adheres to the ISO 26262 standard for functional safety in road vehicles, ensuring compliance with international safety guidelines defined by the International Organization for Standardization (ISO) in 2011 and revised in 2018.

- Repository: https://github.com/commaai/openpilot

Table 2 summarizes the key characteristics of each project. It details the number of files processed (1,011 to 5,518), the number of methods handled (9,292 to 49,315), the count of identified defective methods (343 to 725) and the total amount of lines of code (LOC) contained in those methods (236,933 to 528,272) for which Git data was studied. Given the significant contribution volumes, commit data for Openpilot was collected from January 5, 2025, through October 30, 2025. For all other projects, metadata extraction encompassed a wider range, from the specified date in the table through October 30, 2025.

Table 2: Key code characteristics for each target project.

Project	Start Date	No. Files	No. Methods	No. Defective Methods	LOC
Astrobee	2017-08-31	5,518	9,292	725	236,933
MMGIS	2019-10-02	1,011	18,528	421	503,168
Openpilot	2025-01-05	2,832	49,315	343	528,272

Analysis of the target projects' characteristics yielded several notable findings. A key observation is that only 21.12% of methods were labeled by developers as having received fixes, indicating the presence of a defect. This small proportion of positive instances results in a highly imbalanced dataset, presenting a significant challenge for effective machine learning

model training. Consequently, and consistent with our previous work, model performance was assessed using PR-AUC (Saito et al., 2015), F1, and MCC scores, which provide more reliable evaluation under class imbalance conditions. The scope of this study encompassed a cumulative codebase exceeding 1.2 million lines of source code. Addressing this imbalance is integral to our aim of maximizing recall, ensuring that a greater proportion of actual defects are correctly identified in safety-critical software systems.

Results

RQ: To what extent does a custom-engineered, human-factors-based feature enhance the recall of software defect prediction models compared with state-of-the-art code- and history-metrics models, and how does its performance compare to our prior method-level ML defect prediction model?

To address this question, we applied the RF algorithm to develop three bug prediction model types: (a) Type1 – based on code metrics and history measures, (b) Type2 – incorporating only the engineered sinusoidal time-based metric, and (c) Type3 – based on our previously defined human-error (HE) metrics. Table 3 reports each model's recall value, with the fourth column showing the recall improvement of Type2 over Type1, and the sixth column showing the improvement of Type2 over Type3. The percentage increase in recall was computed as relative improvement, using the formula ((Type2 – Type1) / Type1) × 100, which measures the gain relative to the baseline (Type1) performance.

Table 3: Recall value achieved for each target project and model type.

Project	Type1	Type2	↑ (%)	Type3	↑ (%)
Astrobee	0.52	0.83	59.61%	0.80	3.75%
MMGIS	0.46	0.61	32.60%	0.54	12.96%
Openpilot	0.39	0.60	53.84%	0.54	11.11%
Overall Average	0.45	0.68	48.68%	0.62	9.27%

Average recall values across all projects reveal notable trends. The Type2 model, incorporating the sinusoidal time-based metric, consistently outperformed the Type1 model in every case, achieving an average relative improvement of 48.68%. Furthermore, Type2 also surpassed the Type3 model, which employs our previously defined human-error-based metrics (Alertness and Memory Decay), with an average performance gain of 9.27%.

Evaluating predictive performance with emphasis on correct positive predictions is essential in this study. Table 4 presents the F1 scores and MCC values for the three model types: Type1 (code and history metrics), Type2 (sinusoidal time-based metric), and Type 3 (human-error metrics). On average, the Type2 models achieved an F1 score of 0.62 = AVG(0.66 + 0.62 + 0.58), representing a 72.22% improvement over the 0.36 average for Type 1 models. Similarly, the average MCC for Type2 models was 0.59, a

78.78% gain over the 0.33 average for Type1 models. These consistent gains in both F1 and MCC demonstrate that the sinusoidal time-based approach delivers substantially higher predictive capability and robustness compared to traditional code and history metrics. In contrast, the average F1 score for Type3 models was 0.59, leaving Type2 with only a 5.08% improvement over our previous human-error-based metrics.

Table 4: Recall value achieved for each target project and model type.

Project	Metric	Recall	PR-AUC	F1	MCC
Astrobee	Type1	0.52	0.46	0.46	0.39
Astrobee	Type2	0.83	0.78	0.66	0.62
Astrobee	Type3	0.80	0.80	0.68	0.65
MMGIS	Type1	0.46	0.43	0.45	0.42
MMGIS	Type2	0.61	0.59	0.62	0.60
MMGIS	Type3	0.54	0.66	0.58	0.56
Openpilot	Type1	0.39	0.15	0.19	0.20
Openpilot	Type2	0.60	0.50	0.58	0.57
Openpilot	Type3	0.54	0.49	0.52	0.51

Answer to RQ: Average recall values were 0.45 for code and history metrics, 0.68 for sinusoidal time-based metrics, and 0.62 for our previously defined HE-based metrics. The sinusoidal approach achieved a 48.68% recall improvement over state-of-the-art code and history metrics, and a 9.27% gain over our previously defined Alertness and Memory Decay metrics.

THREATS TO VALIDITY

A potential threat to validity stems from the precision of bug data. Our method relies on correlating bugs with commit messages to identify bug-fixing commits and label affected methods. However, two limitations remain: (1) methods modified in a bug-fixing commit may not have been changed specifically to address the bug, and (2) the lack of a direct, explicit link between bug-fix commits and their location in the revision history hinders accurate identification. For the NASA repositories in particular, internal code changes may occur without being reflected in the public repository.

DISCUSSION AND CONCLUSION

Consistent with our previous findings on the Alertness and Memory Decay human-error metrics, the performance of the sinusoidal time-based feature is comparable, with a modest increase in recall. In safety-critical domains, higher recall expands the defect triage surface, increasing the likelihood of detecting defects before production deployment. While the sinusoidal metric offers marginal gains over the HE-based metrics, it is project-specific: its parameters must be evolved individually for each target project, making it a time- and resource-intensive process. In contrast, the previously defined HE

metrics are standardized and, as demonstrated in earlier work, generalize well across diverse projects.

Conclusion

The sinusoidal equation proved to be an effective foundation for generating prediction metrics that capture temporal cyclicality, reflecting patterns in human behavior. By evolving its parameters to fit the characteristics of each target project, the resulting metric adapts to project-specific activity patterns, delivering superior performance compared to state-of-the-art code and history metrics. Its most notable advantage is a substantially higher recall compared to traditional metrics (+48.68%), which is particularly valuable in safety-critical domains where defect triage benefits from a broader set of potential defect candidates. When compared with our previously defined human-error metrics of Alertness and Memory Decay, the sinusoidal metric achieved a 9.27% improvement in recall while maintaining similar F1 and MCC scores—a meaningful gain that enhances defect identification workflows in safety-critical software systems.

DECLARATIONS

Declaration of Funding

The author(s) received no specific funding for this work.

Competing Interests

There are no competing interests to declare.

Data Availability Statement

The data and machine learning model training algorithms that support the findings of this study are openly available in the figshare repository at the following DOI: 10.6084/m9.figshare.30736385. This repository includes the algorithms used to calculate F1 scores, MCC scores, and statistical confidence intervals, as well as the pre-calculated features (software metrics and HE metrics) from all target repositories analyzed.

REFERENCES

Akiyama, F., 1971. An example of software system debugging. In IFIP Congress (1), pp. 353–359.

Bieman, J.M., 1997. Software metrics: A rigorous & practical approach. IBM Systems Journal, 36(4), p. 594.

Gunning, D., 2017. Explainable artificial intelligence (XAI). Defense Advanced Research Projects Agency (DARPA), 2(2), p. 1.

Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R. and Whitehead, E.J., 2013. Does bug prediction support human developers? Findings from a Google case study. 2013 35th International Conference on Software Engineering (ICSE), pp. 372–381.

Li, Z., Niu, J. and Jing, X.Y., 2024. Software defect prediction: future directions and challenges. Automated Software Engineering, 31(1), p. 19.
Matthews, B.W., 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA) – Protein Structure, 405(2), pp. 442–451.
Menzies, T., Greenwald, J. and Frank, A., 2007. Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering, 33(1), pp. 2–13.
Mo, R., Wei, S., Feng, Q. and Li, Z., 2022. An exploratory study of bug prediction at the method level. Information and Software Technology, 144, p. 106794.
Ramirez, C.A. & Itoh, M., 2025. Actionable Insights from Developer Behavior: A Practical Approach to Software Defect Prediction. Preprint. Available at: https://doi.org/10.21203/rs.3.rs-7415622/v1
Reason, J., 1990. Human Error. Cambridge: Cambridge University Press.
Saito, T. and Rehmsmeier, M., 2015. The precision–recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLoS ONE, 10(3), e0118432.
Stone, M., 1974. Cross-validatory choice and assessment of statistical predictions. Journal of the Royal Statistical Society: Series B (Methodological), 36(2), pp. 111–133.
Van Rijsbergen, C.J., 1979. Information Retrieval. London: Butterworths.