
Volume 1 Issue 1

Article Number: JASAT20260004

Explainable Prediction of Open-Source Repository Maintenance Risk from Activity Signals

Suresh Muthusamy¹ and Jitendra Kumar Jaiswal*²

¹Department of Electrical and Electronics Engineering, Kongu Engineering College (Autonomous),
Perundurai, Erode, Tamil Nadu, India

²Department of Computer Science & Engineering (Artificial Intelligence & Machine Learning),
Dayananda Sagar University, Bengaluru, Karnataka 560078, India

Abstract

Open-source software sustainability depends on repositories continuing to accept changes, resolve maintenance demand, preserve contributor knowledge, deliver releases, and update technical dependencies. Distinguishing genuine maintenance risk from an intentional pause, a mature low-change state, or completed software is nevertheless difficult. A single observable, such as time since the last commit, cannot represent this distinction because maintenance is socio-technical, temporally uneven, and conditioned by project context. This paper proposes a conceptual framework for forward-looking repository maintenance-risk prediction using longitudinal signals from commits, contributors, issues, pull requests, releases, dependencies, and community interactions. Maintenance risk is defined as a latent probability that a repository will become unable to perform expected maintenance functions during a specified future horizon, rather than as an unconditional abandonment label. The framework constructs windowed levels, trends, volatility, latency, concentration, and cross-signal interactions; adjusts interpretation for repository context; compares interpretable, nonlinear, survival, and temporal models; and places global, repository-specific, temporal, counterfactual, and uncertainty explanations between prediction and stakeholder action. Ten testable propositions and a time-aware validation protocol are developed without claiming empirical performance. The intended contribution is a falsifiable architecture that can support maintainers, adopters, ecosystem managers, and funders while limiting stigmatizing or overconfident use. Longitudinal, cross-ecosystem validation and human evaluation of explanation usefulness remain necessary.

Keywords: Explainable Artificial Intelligence; Maintenance-Risk Prediction; Mining Software Repositories; Open-Source Software; Repository Maintenance; Software Sustainability

1. Introduction

Modern software is assembled within ecosystems of interdependent projects, contributors, packages, and organizations rather than produced by isolated repositories. Reviews of software ecosystems emphasize both technical relationships and actor networks [1], [2]; health frameworks consequently treat longevity, productivity, robustness, and value creation as related but distinct properties [3], [4]. At repository level, interrupted maintenance can delay fixes, accumulate unanswered demand, weaken knowledge continuity, lengthen release cycles, and transmit risk to dependents. Yet the

*Corresponding Author: Jitendra Kumar Jaiswal (jitendrajaiswal-aiml@dsu.edu.in)

Received: 11 June 2026; Revised: 21 July 2026; Accepted: 24 August 2026; Published: 30 September 2026

© 2026 The Author(s).

This is an open access article licensed under a [Creative Commons Attribution-NonCommercial 4.0 International Licence](https://creativecommons.org/licenses/by-nc/4.0/) (CC BY-NC 4.0).

observable history is ambiguous: a quiet repository may be complete and stable, temporarily paused, maintained elsewhere, or genuinely losing the capability to respond.

Empirical studies reinforce this ambiguity. Maintainer accounts of deprecated projects identify multiple social, technical, and organizational reasons for failure [5]. A maintenance-activity model based on observable repository characteristics shows that maintenance is measurable but should not be reduced to one event type [6]. Ecosystem position is associated with sustained activity [7], while longitudinal survival work finds alternating active and inactive phases and explicitly warns that projects can return after a period classified as “dead” [8]. Consequently, stars, forks, commit totals, or recency considered alone cannot separate diminished maintenance capacity from intentional low activity.

This paper treats *repository maintenance risk* as multidimensional, forward-looking, and time varying. The objective is to develop an explainable conceptual framework that predicts repository maintenance risk from observable longitudinal activity signals. Four questions guide the design: **RQ1**: Which activity signals theoretically indicate emerging maintenance risk? **RQ2**: How can heterogeneous signals be integrated into a temporally aware prediction framework? **RQ3**: How can explanations communicate repository- and ecosystem-level risk factors? **RQ4**: How should the framework be evaluated without treating temporary inactivity as abandonment?

The contributions are: (1) a multidimensional, horizon-specific definition of repository maintenance risk; (2) a taxonomy connecting activity signals to temporal interpretations and confounders; (3) an explainable prediction architecture with calibrated uncertainty and stakeholder-facing explanation levels; and (4) a longitudinal empirical validation protocol. These are design propositions, not results from a completed experiment or deployment.

2. Background and Related Work

2.1. Sustainability, People, and Responsiveness

Maintenance capacity depends on continued knowledge and participation. Truck-factor estimation operationalizes concentration of file knowledge and shows why headcount alone is insufficient [9]. Developer turnover has been associated with software-quality effects [10], while long-time-contributor prediction uses early activity traces to study retention [11]. Social capital is linked with sustained participation [12], and newcomer research documents technical, documentation, orientation, and interaction barriers [13]. Together these findings motivate measures of active contributors, arrivals and departures, repeat participation, core retention, and concentration rather than a single contributor count.

Responsiveness is another maintenance function. Pull-based development research characterizes distributed contribution and integration workflows [14]; pull-request decisions reflect both technical and social information [15]; and evaluation latency is explained by multiple covariates rather than volume alone [16]. Hence issue and pull-request backlogs, response and resolution latency, unanswered fractions, review participation, and external-contributor acceptance should be represented as trajectories conditioned on incoming demand.

2.2. Technical Upkeep, Context, and Data Quality

Dependency maintenance is not equivalent to code activity. Studies of library migration report that updates are often delayed even when dependencies evolve [17]; package-network research shows that ecosystem growth, transitive dependency structure, and concentrated update activity create heterogeneous exposure [18]; and vulnerability propagation depends on discovery, constraints, and repair timing [19]. Upkeep signals should therefore include dependency-update activity, unresolved automated update requests, documentation or build maintenance, and security-fix responsiveness where reliable data exist.

Popularity must remain contextual. Stars represent several user practices and do not provide a direct maintenance label [20]. Sampling itself can also distort inference: curation research separates engineered projects from tutorials and other non-comparable repositories [21], while GitHub-mining studies document missing or misleading fields and platform-specific assumptions [22]. Automated accounts further inflate or reshape commits and conversations, requiring bot detection or sensitivity analysis [23].

2.3. Explainable Prediction

Explainable artificial intelligence (XAI) distinguishes transparent models from post-hoc methods and global from local explanations [24]. Local Interpretable Model-agnostic Explanations (LIME) approximate behavior near an instance [25]; SHapley Additive exPlanations (SHAP) provide additive feature attributions under stated assumptions [26]; and counterfactual explanations describe changes associated with an alternative model outcome [27]. Effect plots require

care when predictors are correlated, motivating accumulated local effects as an alternative to extrapolating partial dependence [28]. In software analytics, correlated metrics can materially alter interpretation rankings [29]. Time-series benchmarks likewise show that attribution across feature and time dimensions is not automatically reliable [30]. Thus explanations must be audited for stability, dependence sensitivity, and temporal fidelity, and must never be presented as causal evidence.

Table 1 synthesizes these streams. The gap is not an absence of useful measures, but the lack of an integrated framework that joins multiple maintenance functions, temporal change, contextual adjustment, calibrated prediction, and decision-oriented explanations.

Table 1: Summary of Related Research Streams

Research stream	Typical signals	Predicted or assessed outcome	Methodological limitation	Relevance here
Project sustainability and survival [5–8]	Activity recency, event histories, project and ecosystem characteristics	Deprecation, maintenance activity, sustained activity, or survival	Outcome definitions and observation windows differ; quiet periods can be reversible	Motivates a horizon-specific, tiered outcome and time-aware validation
Contributor continuity [9–13]	Ownership, core membership, social ties, newcomer and repeat activity	Knowledge concentration, turnover effects, or sustained participation	Contribution volume can conceal concentration and non-code work	Motivates retention, arrival/departure, diversity, and concentration features
Issue and pull-request work [14–16]	Demand volume, review status, latency, participant and technical attributes	Acceptance, integration practice, or response time	Often models a transaction rather than repository-wide capacity	Converts individual-workflow measures into longitudinal capacity indicators
Dependencies and upkeep [17–19]	Dependency age, update events, network position, advisory and fix timing	Migration, network fragility, or vulnerability exposure	Ecosystem metadata and update norms vary	Adds dependency upkeep and ecosystem position while retaining stratification
Repository mining and popularity [20–23]	Stars, repository metadata, engineered-project cues, bot patterns	Popularity, project curation, or data validity	Platform fields are incomplete and automated activity can bias counts	Treats popularity as context and mandates curation, bot handling, and missingness audits
Explainable prediction [24–30]	Coefficients, local surrogates, attributions, effects, counterfactuals, temporal saliency	Global, local, contrastive, or temporal understanding	Correlation, reference populations, and instability can mislead	Requires multi-level explanations, uncertainty, and explanation-quality evaluation

3. Maintenance Risk Conceptualization

Repository maintenance risk is defined as a latent, forward-looking construct representing the probability that repository r will become unable to perform expected maintenance functions during a defined future horizon H . “Expected” is conditional on project type and expressed demand. The construct is not permanent abandonment, temporary inactivity, project completion, low popularity, low development intensity, defect proneness, or vulnerability risk. Those states may coexist with, precede, or be independent of maintenance risk.

Seven dimensions define maintenance capability: development continuity; contributor continuity; issue and pull-request responsiveness; release and delivery continuity; dependency and technical upkeep; community engagement; and governance resilience. A repository may therefore be high risk despite recent commits if almost all work comes from one departing maintainer and unresolved external demand is increasing. Conversely, a mature library with stable interfaces, little demand, and prompt security responses may remain low risk despite sparse feature commits.

The target should be observed over a future window and can be operationalized in tiers: *low observed risk*, *emerging risk*, *sustained high risk*, and *confirmed inactive*, *archived*, or *abandoned*. Candidate evidence includes sustained absence

of maintenance events; maintainer-confirmed archival or deprecation; prolonged non-response to issues and pull requests; absence of releases combined with unresolved maintenance demand; and ecosystem-specific unmaintained markers. No single condition is universally sufficient. Archived status may lag cessation, some projects use external trackers, and completed projects may be stable. Labels should therefore combine indicators, allow censoring, and undergo maintainer or expert review.

The proposed tiers are ordered observations, not moral judgments about maintainers. An *emerging* label should require deterioration relative to a repository’s prior behavior or a defensible peer group, whereas *sustained high risk* should require persistence across several observations or maintenance functions. Confirmation should rely on explicit project evidence whenever available. A multi-state formulation can model transitions in both directions, thereby allowing recovery after dormancy instead of forcing an absorbing failure state. The observation unit, prediction horizon, minimum demand condition, and transition rules must be fixed before model training and reported with the dataset.

3.1. Activity-Signal Taxonomy

Table 2: Proposed Repository Maintenance-Risk Signals

Signal family	Example measures	Temporal interpretation	Expected risk relationship	Potential confounders
Development	Commit frequency and recency; active days; change in frequency; code churn; files modified; burstiness	Persistent decline separates contraction from a single quiet window	Declining continuity, after adjustment, is expected to raise risk	Mature/completed software, seasonal work, mirrored or squashed history
Contributor structure	Active and new contributors; arrivals/departures; core retention; ownership and Herfindahl concentration; dominant share; truck-factor proxies	Loss, failed replenishment, or rising concentration indicates reduced redundancy	Core loss and concentration are expected to raise risk beyond volume alone	Bots, aliases, paid team rotations, pair work, non-code maintainers
Issue management	First-response and resolution latency; unanswered fraction; unresolved growth; reopened issues; maintainer discussion	Demand-adjusted backlog growth indicates weakening response capacity	Persistent latency and unanswered demand are expected to raise risk	External trackers, support boundaries, duplicate or invalid reports
Pull requests	Review and merge latency; unreviewed requests; closure patterns; external acceptance; maintainer review	Worsening review flow indicates constrained integration capacity	Delayed or absent review is expected to raise risk when demand exists	Draft PRs, CI delays, large changes, migration to another workflow
Releases	Release frequency and recency; cadence change; interval irregularity; commit-release gap	Lengthening intervals distinguish delivery interruption from coding activity	Deterioration is expected to raise risk conditional on release model	Continuous delivery, libraries released elsewhere, stable applications
Community engagement	New and repeat contributors; discussion participation; retention; participation diversity; interaction intensity	Declining recurrence or diversity indicates a thinning community	Repeat and diverse participation are expected to buffer decline	Private channels, corporate teams, event-driven communities
Technical upkeep	Dependency updates; unresolved automated update requests; build/CI status; documentation and maintenance commits; security-fix response	Maintenance-specific events reveal upkeep even without features	Persistent unattended upkeep demand is expected to raise risk	Vendoring, pinned dependencies, unsupported CI, generated documentation
Repository context	Age; domain; language; organization/individual owner; size; archived status; ecosystem and dependency position	Defines comparable strata, demand, and expected cadence	Primarily control/moderation, not direct failure evidence	Platform conventions, transfers, forks, and ecosystem-specific metadata

Table 2 specifies proposed signals. Levels describe current load or supply; slopes identify deterioration; volatility and burstiness distinguish episodic from consistently declining activity; and cross-family disagreement can expose hidden risk. Context variables such as age, language, domain, ownership, size, archived status, and ecosystem position should primarily control, moderate, or stratify relationships. Treating them as direct evidence of failure would risk encoding ecosystem conventions or popularity into the outcome.

4. Proposed Conceptual Framework

As introduced in Fig. 1, repository event streams are transformed into comparable temporal features, interpreted within context, modeled as future risk, explained at several levels, and presented as decision support. Context adjustment may use stratification, interactions, hierarchical effects, matched comparison groups, or normalization by incoming demand. Missingness indicators are retained where absence is informative, but platform unavailability is distinguished from genuine zero activity.

Governance resilience is partly visible through review participation, ownership distribution, documented contribution processes, release authority, and continuity of maintenance roles, but repository traces do not reveal every decision channel. The framework therefore represents governance through conservative proxies and stakeholder-supplied context rather than inferring leadership quality. Feature construction should retain both supply (maintenance work) and demand (issues, pull requests, advisories) so that inactivity under no demand is not equated with failure to respond. Cross-family interactions are especially important when stable commit volume masks worsening review latency or when automated dependency updates create activity without human follow-through.

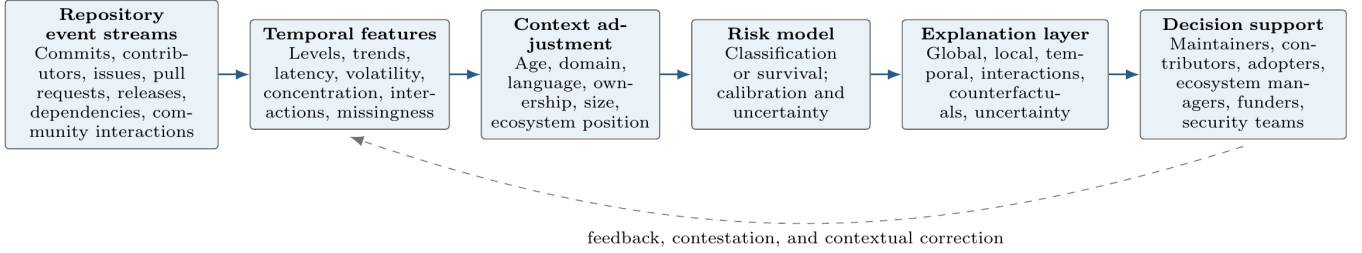


Figure 1: Proposed conceptual framework for explainable repository maintenance-risk prediction. The feedback path allows stakeholders to correct context and challenge interpretations; it is not an automated retraining command.

4.1. Mathematical Specification

Let repository r be observed in windows $t = 1, \dots, T$. Its feature vector is

$$\mathbf{x}_{r,t} = [\mathbf{d}_{r,t}, \mathbf{c}_{r,t}, \mathbf{i}_{r,t}, \mathbf{p}_{r,t}, \mathbf{l}_{r,t}, \mathbf{u}_{r,t}, \mathbf{z}_r], \quad (1)$$

where the components respectively represent development, contributors, issues, pull requests, releases, combined upkeep/community signals, and relatively stable context. The desired prediction is

$$\hat{P}_{r,t}^{(H)} = P(Y_{r,t+H} = 1 \mid \mathbf{x}_{r,1:t}), \quad (2)$$

for a prespecified horizon H . A composite representation is

$$R_{r,t} = f_{\theta}(D_{r,t}, C_{r,t}, I_{r,t}, P_{r,t}, L_{r,t}, U_{r,t}, Z_r). \quad (3)$$

Equation (3) is a conceptual specification, not a validated score or weighting rule. A recent-versus-prior change feature is

$$\Delta x_{r,t}^{(k)} = \frac{1}{k} \sum_{j=0}^{k-1} x_{r,t-j} - \frac{1}{k} \sum_{j=k}^{2k-1} x_{r,t-j}, \quad (4)$$

which preserves the direction of change over adjacent k -window blocks. Contributor concentration can be represented as

$$C_{r,t}^{HHI} = \sum_{j=1}^{n_{r,t}} s_{j,r,t}^2, \quad (5)$$

where $s_{j,r,t}$ is contributor j 's activity share. For censored time-to-risk outcomes, a proportional-hazards candidate is

$$h_r(t | \mathbf{x}_{r,t}) = h_0(t) \exp(\beta^\top \mathbf{x}_{r,t}). \quad (6)$$

Survival formulations retain repositories that have not yet experienced an event and distinguish event probability from time until event.

4.2. Conceptual Propositions

Table 3 states falsifiable propositions. Their purpose is to guide preregistered tests, not to imply confirmation.

Table 3: Conceptual Propositions for Future Testing

ID	Proposition	Required qualification or comparison
P1	Persistent decline in development activity is positively associated with future maintenance risk.	Compare trends with level-only and last-event baselines; adjust for maturity and seasonality.
P2	Increasing contributor concentration is positively associated with maintenance risk.	Separate concentration from team size and bot activity.
P3	Loss of core contributors increases risk beyond the effect of declining commit volume.	Test incremental value and contributor-identity uncertainty.
P4	Growth in unresolved issues and response latency indicates increasing risk.	Normalize by incoming demand and tracker coverage.
P5	Increasing pull-request review latency predicts declining integration capacity.	Condition on request complexity, draft status, and CI delay.
P6	Lengthening or irregular release intervals indicate higher risk conditional on project type.	Stratify continuous-delivery, library, application, and stable/complete projects.
P7	Simultaneous deterioration across signal families is stronger evidence than a single-family decline.	Test interactions and out-of-family validation, not an arbitrary sum.
P8	Repeat participation and contributor diversity moderate the effect of declining development activity.	Define diversity transparently and avoid demographic inference from uncertain data.
P9	Age, domain, ecosystem, and development model moderate signal-risk relationships.	Compare pooled, hierarchical, and ecosystem-specific models.
P10	Temporal explanations improve stakeholder usefulness relative to static rankings alone.	Evaluate with maintainers using controlled, task-based studies.

5. Explainable Risk-Prediction Approach

5.1. Staged Model Architecture

The modeling sequence should begin with regularized logistic regression or a generalized additive model (GAM), then evaluate trees and ensembles, followed by survival and explicitly temporal models. The selection rule is not maximum discrimination alone: prefer the least complex model that achieves adequate discrimination, calibration, lead time, and explanation reliability. Missing data should be characterized by source and mechanism; imputation must be fitted only on training periods, accompanied by missingness indicators where justified. Class weights or resampling may be evaluated within training folds, never before temporal partitioning.

Table 4: Candidate Prediction-Model Comparison

Family	Strength / limitation	Temporal capability	Role and explanation
Logistic / GAM	Transparent baseline; interactions must be specified	Window summaries, splines, lags	Baseline; coefficients/effect plots
Decision tree	Direct rules; unstable if unconstrained	Lagged or windowed inputs	Auditable comparator; paths
RF / boosted trees	Nonlinear interactions; post-hoc explanation needed	Windowed temporal features	Performance challenger; SHAP/ALE
Survival models	Censoring and event time; assumptions vary	Native time-to-event	Hazard/risk curves; coefficients or survival importance
RNN / temporal CNN	Learns sequences; costly and opaque	Native ordered sequences	Challenger; time-window attribution
Temporal transformer	Long-range attention; data intensive	Native long context	Conditional research role; sequence attribution audit

Table 4 summarizes the roles. Random forests and boosted trees can accommodate nonlinearities and interactions, but post-hoc importance can vary with correlated inputs. Recurrent neural networks (RNNs), temporal convolutional networks (CNNs), or transformers are justified only if ordered histories materially outperform well-designed window features and their explanations pass temporal-fidelity tests. Their probability outputs require an explicit calibration check [31]. Survival forests provide a nonlinear censored-data alternative to Equation (6).

5.2. Explanation Layer

Intrinsic interpretation includes signed coefficients, constrained GAM curves, or compact tree rules. Post-hoc interpretation includes permutation importance, accumulated local effects, LIME, SHAP, feature-interaction displays, and counterfactual scenarios. Global explanations answer which signals the fitted model uses across a reference population; local explanations answer why one repository received its current estimate; temporal explanations identify influential windows and trajectory changes; counterfactuals ask what feasible change in modeled inputs is associated with a lower-risk classification.

For an additive local explanation,

$$g(\mathbf{x}_{r,t}) = \phi_0 + \sum_{j=1}^m \phi_j, \quad (7)$$

where ϕ_j is the contribution assigned to feature j under the explainer’s background distribution. A qualitative example is sufficient: a repository could receive elevated risk because issue-response latency has risen, active contributors have declined, and the release interval has lengthened. The interface should show which windows drove those statements, uncertainty around the prediction, and whether similar repositories exhibit the same pattern. It should not invent a prescriptive threshold or claim that changing one metric will cause recovery.

Explanations face correlated commit, contributor, and backlog features; unstable rankings; leakage from outcome-adjacent events; disagreement across explainers; and sensitivity to reference populations. Partial dependence should be replaced or supplemented by dependence-aware effect analysis when extrapolation is implausible. Stability should be evaluated across bootstrap refits, time periods, plausible feature groupings, and matched reference cohorts. Feature attribution establishes model reliance, not causality; counterfactuals describe model-consistent scenarios, not guaranteed interventions.

Interaction explanations should be grouped by maintenance function before individual metrics are displayed, because several measures may encode the same underlying event stream. Uncertainty should separate at least predictive uncertainty, variability across refits, and missing-data sensitivity. Where explanations disagree, the interface should expose the disagreement rather than selecting the most persuasive story. A repository-level explanation is considered decision-ready only if it is faithful to the fitted model, temporally located, robust to a reasonable reference cohort, and comprehensible to its intended stakeholder.

6. Proposed Evaluation and Validation Methodology

6.1. Data, Outcome, and Temporal Design

Future studies should collect public repository event histories under explicit inclusion criteria and a reproducible data schema. Mirrors, forks, tutorials, automatically generated repositories, and very small experimental repositories may be excluded only with documented rules. Projects should be stratified by language, ecosystem, age, application domain, and organization versus individual ownership. Identity resolution, bot filtering, repository transfers, external trackers, and archived or missing fields require audit logs.

As Fig. 2 shows, each example uses only an observation window ending at prediction time t_0 ; an optional gap prevents near-outcome events from leaking the label; outcomes are assessed during horizon H . Multiple cut points per repository require grouped uncertainty estimates so that repeated histories are not treated as independent. Training, validation, and testing must obey $T_{\text{train}} < T_{\text{validation}} < T_{\text{test}}$. Random record-level splitting can place later windows of a repository in training and earlier windows in testing, overstating generalization. Validation design materially affects software-prediction estimates.

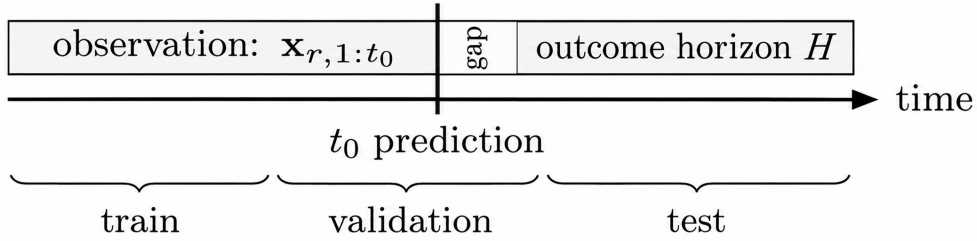


Figure 2: Longitudinal observation, prediction, and outcome windows. Calendar-ordered folds preserve the direction of deployment and may be repeated for several horizons.

Alternative targets should be preregistered: sustained absence of human maintenance events; confirmed archive/deprecation; persistent non-response despite demand; combined release absence and unresolved demand; or a tiered transition. A blinded maintainer/expert sample should review ambiguous cases. Sensitivity analysis should vary the required duration and indicator combination rather than assert a universal cutoff.

Label adjudication should record both the assigned tier and the evidence supporting it. Reviewers should see repository material only up to the outcome-assessment date, use an explicit rubric, and be permitted to mark a case indeterminate. Agreement and disagreement categories should be reported without converting uncertain cases silently into the majority class. For external validation, event concepts rather than platform field names should be aligned: for example, “first maintainer response” may be observed through different APIs or trackers. This design tests whether the construct transports, not merely whether the extraction script can be reused.

6.2. Measures, Baselines, and Robustness

Discrimination should include precision, recall, F1, receiver-operating-characteristic area, and precision–recall area; the latter is particularly informative under imbalance [32]. Probability quality should include Brier score, calibration curves, and calibration error, with calibration learned on validation periods. Temporal validation choices materially affect software-prediction estimates [33]; censoring-aware comparisons should include a random-survival-forest baseline [34]. Survival models also require concordance and time-dependent calibration. Operational measures include lead time, alerts per maintainer, false-alert burden, and performance by ecosystem and project age.

Required baselines are time since last commit, recent commit count, contributor count, issue-response latency, a simple inactivity rule, and an interpretable multivariable model. The full framework must demonstrate incremental value over these simpler signals. Table 5 lists the protocol. Confidence intervals should be clustered by repository and, where appropriate, ecosystem. Model selection, thresholding, imputation, balancing, and feature screening must occur inside the training/validation process.

Table 5: Proposed Evaluation and Robustness Protocol

Component	Required design	Measures or comparison	Failure checked
Temporal prediction	Calendar-ordered train/validation/test; grouped repositories; multiple horizons and cut points	Precision, recall, F1, PR-AUC, ROC-AUC, lead time	Future leakage and repository overlap
Probability / survival	Validation-period calibration; censoring-aware models where appropriate	Brier score, calibration error/plot, concordance, time-dependent calibration	Confident but unreliable probabilities
Baselines	Single metrics, simple inactivity rule, and interpretable multivariable baseline	Incremental discrimination, calibration, lead time, and alert burden	Complexity without decision value
Robustness	Alternative labels, windows, horizons; age strata; ecosystems; bot exclusion; missing tracker sensitivity; drift; external platform when feasible	Performance distribution and subgroup differences; prediction disagreement	Definition dependence and poor transportability
Explanation quality	Global/local/temporal fidelity; refit and cohort stability; sparsity; comprehensibility; actionability; expert agreement	Perturbation tests, rank agreement, task accuracy, interviews or controlled studies	Plausible but unfaithful or unusable explanations
Governance	Uncertainty display, contestation channel, contextual notes, access controls, periodic review	Error reports, override reasons, differential false alerts	Stigma, metric gaming, and unreviewed automation

Robustness checks should cover alternative risk labels, observation windows, prediction horizons, repository-age strata, ecosystem-specific and hierarchical models, exclusion of bots, missing issue/PR data, and concept drift. External validation should cross an ecosystem or hosting platform when equivalent events can be aligned. Explanations must be evaluated for fidelity, stability, consistency, sparsity, comprehensibility, actionability, and maintainer usefulness. A useful explanation study asks participants to identify the deteriorating function, compare repositories, decide whether to investigate, and detect an intentionally misleading explanation.

7. Discussion and Implications

The proposed framework can support early-warning dashboards, maintainer-support prioritization, dependency-risk review, contributor onboarding, ecosystem health monitoring, funding decisions, succession planning, and software adoption. Different stakeholders require different views. A maintainer needs local trajectories and contestable context; an adopter needs uncertainty and evidence of response capacity; an ecosystem manager needs calibrated aggregate patterns and dependency centrality; a funder needs transparent criteria and safeguards against rewarding superficial activity.

Operationalization should therefore separate evidence, inference, and action. The evidence layer reports dated observations and missingness; the inference layer reports horizon-specific probabilities, calibration information, and explanation stability; the action layer applies a stakeholder-owned policy. A low-risk classification may require no action, emerging risk may prompt a private review, sustained high risk may justify direct outreach, and confirmed inactivity may trigger migration planning. These tiers should not be fixed by the prediction model itself. Thresholds should reflect the cost of missed deterioration, the burden of false alerts, the reversibility of the response, and available support capacity. Alerts should also expire or be recomputed as new windows arrive, because a repository can recover. Recording acknowledgement, override, and follow-up outcomes would make the decision process auditable and provide evidence for later evaluation without treating a maintainer’s response as automatic confirmation of the original prediction.

The framework should support, not replace, human judgment. Public scores can stigmatize volunteers, attract unwanted pressure, or be mistaken for definitive judgments about project quality. Visible activity is easier to count than maintenance value, encouraging metric gaming and undervaluing design review, moderation, or quiet stability. A repository owner should be able to explain completion, migration, private support work, seasonal schedules, or a deliberate maintenance mode. Interfaces should use neutral language (“signals warrant review”), display uncertainty and time windows, expose the evidence used, and permit contestation or correction.

Counterfactuals require special restraint. “Reducing response latency” may be model-associated with lower risk yet infeasible without funding; generating cosmetic commits could lower a poorly designed score without improving

capability. Feasible counterfactuals should operate on grouped maintenance functions, respect immutable context, and be labeled scenarios rather than recommendations. Funding or security triage should never depend on an unexplained rank alone.

8. Threats to Validity

Construct validity: maintenance, dormancy, completion, and abandonment are not interchangeable. Composite or expert-reviewed outcomes reduce but do not eliminate ambiguity. External trackers and private communication hide legitimate maintenance. *Internal validity:* associations between activity and later outcomes do not identify causes; contributor workload, organizational decisions, or ecosystem shocks may confound them. Outcome-adjacent archival labels, dependency alerts, or maintainer statements can leak future information.

External validity: languages, release conventions, governance models, and hosting platforms differ. Popular public repositories are not representative of small, scientific, government, or corporate-sponsored projects. Survivorship and selection biases remain possible. *Conclusion validity:* rare events, repeated windows, tuning multiplicity, and unstable explanations can produce overconfident conclusions; clustered uncertainty and held-out temporal tests are required.

Data-quality threats: bots, aliases, squashed commits, migrated repositories, missing events, changed platform features, and incomplete dependency graphs can distort levels and trends. Sensitivity analyses must report how each correction changes predictions. *Ethical validity:* labels may burden volunteers, encode popularity or ecosystem privilege, and incentivize visible activity. Human review, uncertainty, access controls, appeal mechanisms, and documented intended use are part of validity, not optional deployment details.

9. Conclusion

Repository maintenance risk cannot be inferred responsibly from one activity counter. This paper defines it as a horizon-specific latent probability of losing expected maintenance capability and proposes a framework that combines longitudinal development, contributor, responsiveness, release, upkeep, community, and contextual signals. The architecture separates event collection, temporal feature construction, context adjustment, calibrated modeling, multi-level explanation, and stakeholder decision support. It provides equations, ten testable propositions, candidate model roles, a tiered outcome, and a future validation protocol without reporting invented experiments or thresholds.

The central empirical question is whether cross-family deterioration yields earlier, better-calibrated, and more useful warnings than simple inactivity rules while avoiding false alarms for mature or temporarily quiet projects. Answering it requires calendar-ordered, cross-ecosystem validation, censored time-to-event analysis, alternative label definitions, and maintainer-centered explanation evaluation. Until those tests are completed, the framework should be regarded as a transparent research specification rather than a deployed scoring system.

Author Contributions

Suresh Muthusamy (S.M): Conceptualization, Methodology, Investigation, Formal analysis, Data curation, Visualization, Writing – original draft; Jitendra Kumar Jaiswal (J.K.J): Conceptualization, Methodology, Validation, Supervision, Writing – review & editing, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

The data that support the findings of this study are available from the corresponding author, S.M, upon reasonable request.

Declaration of Generative AI and AI-Assisted Technologies in the Writing Process

During the development of this manuscript, the authors employed AI-assisted tools to support selected aspects of the writing and editing process. These tools were used primarily for improving language quality, correcting grammatical errors, enhancing clarity, and refining the overall presentation of the manuscript. AI-assisted technologies were also used selectively during the initial stages of drafting to improve writing efficiency. All AI-generated or AI-assisted material was critically evaluated, revised, and verified by the authors. The authors take full responsibility for the accuracy, originality, interpretation, and integrity of the final manuscript.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

- [1] K. Manikas and K. M. Hansen, “Software ecosystems—A systematic literature review,” *J. Syst. Softw.*, vol. 86, no. 5, pp. 1294–1306, 2013, doi: 10.1016/j.jss.2012.12.026.
- [2] O. Franco-Bedoya, D. Ameller, D. Costal, and X. Franch, “Open source software ecosystems: A systematic mapping,” *Inf. Softw. Technol.*, vol. 91, pp. 160–185, 2017, doi: 10.1016/j.infsof.2017.07.007.
- [3] S. Jansen, “Measuring the health of open source software ecosystems: Beyond the scope of project health,” *Inf. Softw. Technol.*, vol. 56, no. 11, pp. 1508–1519, 2014, doi: 10.1016/j.infsof.2014.04.006.
- [4] M. Oriol, C. Müller, J. Marco, P. Fernandez, X. Franch, and A. Ruiz-Cortés, “Comprehensive assessment of open source software ecosystem health,” *Internet Things*, vol. 22, Art. no. 100808, 2023, doi: 10.1016/j.iot.2023.100808.
- [5] J. Coelho and M. T. Valente, “Why modern open source projects fail,” in *Proc. 11th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2017, pp. 186–196, doi: 10.1145/3106237.3106246.
- [6] J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, “Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects,” *Inf. Softw. Technol.*, vol. 122, Art. no. 106274, 2020, doi: 10.1016/j.infsof.2020.106274.
- [7] M. Valiev, B. Vasilescu, and J. Herbsleb, “Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem,” in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 644–655, doi: 10.1145/3236024.3236062.
- [8] A. Ait, J. L. Cánovas Izquierdo, and J. Cabot, “An empirical study on the survival rate of GitHub projects,” in *Proc. 19th Int. Conf. Mining Softw. Repositories*, 2022, pp. 365–375, doi: 10.1145/3524842.3527941.
- [9] G. Avelino, L. Passos, A. Hora, and M. T. Valente, “A novel approach for estimating truck factors,” in *Proc. IEEE 24th Int. Conf. Program Comprehension*, 2016, pp. 1–10, doi: 10.1109/ICPC.2016.7503718.
- [10] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, “Impact of developer turnover on quality in open-source software,” in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 829–841, doi: 10.1145/2786805.2786870.
- [11] L. Bao, X. Xia, D. Lo, and G. C. Murphy, “A large scale study of long-time contributor prediction for GitHub projects,” *IEEE Trans. Softw. Eng.*, vol. 47, no. 6, pp. 1277–1298, 2021, doi: 10.1109/TSE.2019.2918536.
- [12] H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu, “Going farther together: The impact of social capital on sustained participation in open source,” in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.*, 2019, pp. 688–699, doi: 10.1109/ICSE.2019.00078.
- [13] I. Steinmacher, M. A. Graciotto Silva, M. A. Gerosa, and D. F. Redmiles, “A systematic literature review on the barriers faced by newcomers to open source software projects,” *Inf. Softw. Technol.*, vol. 59, pp. 67–85, 2015, doi: 10.1016/j.infsof.2014.11.001.
- [14] G. Gousios, M. Pinzger, and A. van Deursen, “An exploratory study of the pull-based software development model,” in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 345–355, doi: 10.1145/2568225.2568260.

- [15] J. Tsay, L. Dabbish, and J. Herbsleb, “Influence of social and technical factors for evaluating contribution in GitHub,” in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 356–366, doi: 10.1145/2568225.2568315.
- [16] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, “Wait for it: Determinants of pull request evaluation latency on GitHub,” in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, 2015, pp. 367–371, doi: 10.1109/MSR.2015.42.
- [17] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, “Do developers update their library dependencies?” *Empir. Softw. Eng.*, vol. 23, no. 1, pp. 384–417, 2018, doi: 10.1007/s10664-017-9521-5.
- [18] A. Decan, T. Mens, and P. Grosjean, “An empirical comparison of dependency network evolution in seven software packaging ecosystems,” *Empir. Softw. Eng.*, vol. 24, no. 1, pp. 381–416, 2019, doi: 10.1007/s10664-017-9589-y.
- [19] A. Decan, T. Mens, and E. Constantinou, “On the impact of security vulnerabilities in the npm package dependency network,” in *Proc. 15th Int. Conf. Mining Softw. Repositories*, 2018, pp. 181–191, doi: 10.1145/3196398.3196401.
- [20] H. Borges and M. T. Valente, “What’s in a GitHub star? Understanding repository starring practices in a social coding platform,” *J. Syst. Softw.*, vol. 146, pp. 112–129, 2018, doi: 10.1016/j.jss.2018.09.016.
- [21] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, “Curating GitHub for engineered software projects,” *Empir. Softw. Eng.*, vol. 22, no. 6, pp. 3219–3253, 2017, doi: 10.1007/s10664-017-9512-6.
- [22] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, “The promises and perils of mining GitHub,” in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 92–101, doi: 10.1145/2597073.2597074.
- [23] M. Golzadeh, A. Decan, D. Legay, and T. Mens, “A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments,” *J. Syst. Softw.*, vol. 175, Art. no. 110911, 2021, doi: 10.1016/j.jss.2021.110911.
- [24] A. Barredo Arrieta et al., “Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” *Inf. Fusion*, vol. 58, pp. 82–115, 2020, doi: 10.1016/j.inffus.2019.12.012.
- [25] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?: Explaining the predictions of any classifier,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144, doi: 10.1145/2939672.2939778.
- [26] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 4765–4774, arXiv:1705.07874.
- [27] S. Wachter, B. Mittelstadt, and C. Russell, “Counterfactual explanations without opening the black box: Automated decisions and the GDPR,” *Harvard J. Law Technol.*, vol. 31, no. 2, pp. 841–887, 2018, doi: 10.2139/ssrn.3063289.
- [28] D. W. Apley and J. Zhu, “Visualizing the effects of predictor variables in black box supervised learning models,” *J. Royal Stat. Soc. Series B*, vol. 82, no. 4, pp. 1059–1086, 2020, doi: 10.1111/rssb.12377.
- [29] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, “The impact of correlated metrics on the interpretation of defect models,” *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 320–331, 2021, doi: 10.1109/TSE.2019.2891758.
- [30] A. A. Ismail, M. Gunady, H. Corrada Bravo, and S. Feizi, “Benchmarking deep learning interpretability in time series predictions,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 6441–6452, arXiv:2010.13924.
- [31] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proc. 34th Int. Conf. Mach. Learn.*, ser. PMLR, vol. 70, 2017, pp. 1321–1330, arXiv:1706.04599.
- [32] T. Saito and M. Rehmsmeier, “The precision–recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets,” *PLoS ONE*, vol. 10, no. 3, Art. no. e0118432, 2015, doi: 10.1371/journal.pone.0118432.
- [33] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “An empirical comparison of model validation techniques for defect prediction models,” *IEEE Trans. Softw. Eng.*, vol. 43, no. 1, pp. 1–18, 2017, doi: 10.1109/TSE.2016.2584050.
- [34] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer, “Random survival forests,” *Ann. Appl. Stat.*, vol. 2, no. 3, pp. 841–860, 2008, doi: 10.1214/08-AOAS169.