iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85546

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

RiskPilot: A Web-Based Rule-Based Decision-Support System for Predicting Software Project Success or Failure with Interpretable Risk Indicators

Kaviya R[1], Latha S[2], Lavanya S[3], Mrs. K. Yazhini[4]
[1, 2, 3]*Department of Computer Science and Engineering (Final-year B.E. students),* [4]*Assistant Professor, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution), Dr. M.G.R. Nagar, Hosur – 635 130, Tamil Nadu, India*
(Affiliated to Anna University, Chennai – 600 025)

Abstract: ***Software projects frequently exceed their budgets, miss their schedules or fall short of quality expectations, and manual tracking through spreadsheets rarely exposes these risks early, consistently and with an explanation. This paper presents RiskPilot, a web-based decision-support system that classifies a software project as Success or Failure from twelve project parameters: budget, duration, team size, team experience, project complexity, requirement changes, sprint delays, bug count, testing coverage, resource availability, client involvement and customer satisfaction. The core of the system is a transparent rule-based classifier built from IF–ELSE conditions, which also reports risk indicators and improvement recommendations linked to the rules that were triggered. The prediction service is additionally structured to accept a trained gradient-boosting model with SHAP-based attributions and to fall back to the rule-based classifier when no model is available. The platform provides authenticated access, project management, a risk-distribution dashboard, dataset upload and validation, GitHub repository identification and reporting. The architecture, the prediction-service logic and the implemented modules are described, and the working application is illustrated through screenshots. Quantitative predictive performance on labelled project data has not yet been established; an evaluation protocol is therefore specified. The system is intended to support, not replace, managerial judgement.***

Keywords: Software project management, success/failure prediction, rule-based classification, project risk analysis, explainable AI, decision support system, web application.

I. INTRODUCTION

Delivering software on time, within budget and at the expected quality remains difficult. Kawamura et al. note that several studies indicate roughly 70% of software development projects end in failure [1], and Cooper reports estimates of failure rates as high as 80% for AI projects [3]. Typical contributing factors include budget overruns, schedule delays, growing complexity, frequent requirement changes, insufficient testing and limited resources. When these conditions are monitored separately in spreadsheets or basic dashboards, their combined effect is hard to judge, and warning signs may surface only after the project has already been affected. Predictive approaches can help, but they differ in what they require. Machine-learning classifiers need suitable labelled history of past projects and careful validation, whereas rule-based classifiers evaluate inputs against explicit conditions and are straightforward to inspect. For teams that do not yet hold a labelled project corpus, a transparent rule-based classifier that also states which conditions raised concern is a practical starting point, provided it is built so that a data-driven model can later be introduced and compared.

This paper makes the following contributions:

- A web application, RiskPilot, that gathers twelve project parameters and returns a Success or Failure classification together with rule-linked risk indicators and recommendations.
- A prediction-service design that always produces a rule-based baseline and, when a trained model is available, can additionally produce a model prediction with SHAP explanations, falling back to the baseline on any failure (Section VII-C).
- Supporting modules for authentication, project management, a risk-distribution dashboard, dataset upload with validation of required training features, GitHub repository identification and reporting.

©IJRASET: All Rights are Reserved

- An explicit evaluation protocol for measuring the predictive quality of the classifier on labelled project data (Section V-G).

The remainder of the paper is organised as follows. Section II reviews related work, Sections III and IV state the problem and objectives, Section V describes the methodology, Sections VI and VII present the architecture and implementation, Section VIII discusses results, and Sections IX and X conclude and outline future enhancements.

II. LITERATURE REVIEW

A. *Outcome prediction for software projects*

Kawamura, Toma and Takano collected data on 332 projects from Japanese IT vendors through an Internet survey and built a success/failure prediction algorithm with a Bayes classifier, obtaining a prediction capability of 77.3% [1]. Their aim was to help an organisation identify the projects that most need its support. The work establishes that project outcomes can be predicted from organisational and project attributes, but its contribution is the algorithm and its survey data rather than an interactive tool that explains to a project manager which conditions drive a given result.

B. *Failure analysis*

Nizam examined software project failure as a process, using grounded theory on critical incidents and repeating team behaviours drawn from software-engineering literature and case studies, and reported that common behavioural patterns appeared in approximately 89% of the case studies [2]. The study offers a phase-based description of how failure unfolds that managers can use to recognise warning signs; it is a descriptive process model and does not classify an individual project from numerical inputs. Cooper analysed the available evidence on why AI projects fail and found the causes strikingly similar to those established in new product development, arguing that product-development practices can be adapted to AI projects [3]. This is a managerial analysis rather than a prediction tool.

C. *Machine-learning prediction of failure*

Taye and Feleke trained five classifiers (SVM, decision tree, k-nearest neighbours, logistic regression and naive Bayes) to predict failure in the ten project-management knowledge areas for software companies, using 443 records with nine attributes collected by questionnaire from software companies in Ethiopia; SVM performed best with 92.13% accuracy on a 20% hold-out set [4]. The study shows the potential of data-driven models, and also their dependence on a purpose-collected labelled dataset and on model validation.

D. *Summary and research gap*

TABLE I
SUMMARY OF THE REVIEWED STUDIES

Ref.	Approach	Evidence base	Relation to this work
[1]	Bayes classifier for project success/failure	332 projects, Japanese IT vendors (survey); 77.3% prediction capability	Algorithm-focused; no rule-linked explanations or application workflow in its stated contribution
[2]	Grounded-theory failure-process definition	Case studies of failed software projects; behavioural patterns in about 89%	Descriptive process model; no per-project classifier
[3]	Analysis of AI project failure using new-product-development lessons	Available evidence reviewed by the author	Managerial analysis; no prediction tool
[4]	SVM, DT, KNN, LR, NB for knowledge-area failure	443 records, 9 attributes, questionnaire data; SVM 92.13%	Needs a labelled dataset; multi-class knowledge-area target

Within the reviewed set, no study combines in one deployed web application (i) structured intake of the project parameters used by project managers, (ii) a transparent Success/Failure classification whose triggering conditions are shown as risk indicators and recommendations, (iii) an optional model path with feature attributions, and (iv) dataset validation, dashboarding and reporting. RiskPilot addresses this combination. It does not claim higher predictive accuracy than the data-driven studies above, because its rules have not yet been evaluated against labelled data.

III. PROBLEM STATEMENT

Let a project be described by a parameter vector $x = (x_1, \ldots, x_{12})$ containing the parameters listed in Table II. The problem is to determine a mapping from x to a label in {Success, Failure} and, simultaneously, to a set of risk indicators that identify which conditions of x raise concern, subject to three constraints: the decision must be inspectable by a non-specialist user; it must be computable without an a-priori labelled project corpus; and the design must allow a data-driven model to be introduced and compared later.

IV. OBJECTIVES

1) To classify software projects as Success or Failure from project parameters using predefined IF–ELSE conditions.
2) To identify project risks early by evaluating budget, duration, complexity, requirement changes, sprint delays, bug count and related conditions.
3) To present explainable results in which risk indicators and recommendations are tied to the conditions that were evaluated.
4) To provide a centralised platform for registration, login, project creation, parameter entry, history and dashboard visualisation.
5) To generate project analysis reports for later reference, and to support dataset upload so that data-driven models can be trained and compared in future work.

V. PROPOSED METHODOLOGY

A. Overview

The methodology has four stages: (1) intake and validation of the twelve project parameters; (2) evaluation of predefined rules to obtain the set of triggered risk indicators; (3) classification of the project as Success or Failure from the rule results; and (4) presentation of the label, risk indicators and recommendations, with storage for dashboard and history. A model-based path, described in Section V-E, can run alongside stage (3) when a trained model exists.

B. Input parameters

Table II lists the parameters collected by the prediction form (Fig. 6). Ranges shown in parentheses are those printed on the form; the remaining parameters are numeric entries.

TABLE II
PROJECT PARAMETERS USED BY THE PREDICTION FORM

No.	Parameter	Form entry	Risk dimension in Table III
1	Budget	Numeric (Indian rupees, ₹)	—
2	Duration	Numeric, days	Schedule
3	Team size	Numeric	—
4	Team experience	Numeric, years	Team capability
5	Project complexity	Scale 1–5	—
6	Requirement changes	Count	Scope
7	Sprint delays	Count	Schedule
8	Bug count	Count	Quality
9	Testing coverage	Percentage	Testing
10	Client involvement	Scale 1–5	—
11	Resource availability	Percentage	Resource
12	Customer satisfaction	Scale 1–5	—

C. Rule-based classification

Each rule j is a triple (p_j, ρ_j, τ_j) consisting of a parameter index p_j, a comparison operator ρ_j chosen from $<$, $\leq$, $>$ and $\geq$ and a threshold τ_j. The violation indicator and the set of triggered rules are

$$v_j(x) = 1 \text{ if the condition } x[p_j]\ \rho_j\ \tau_j \text{ is violated, otherwise } 0 \quad (1)$$

$$I(x) = \text{the list of all } j \text{ for which } v_j(x) = 1 \quad (2)$$

The label is determined by the configured success predicate S, a Boolean combination of the rule results:

$$\hat{y}(x) = \text{Success if } S(x) \text{ is true; Failure otherwise} \quad (3)$$

The implemented rule set has seven rules (R1–R7), listed with their thresholds in Table III. A rule is violated when its condition holds, for example R1 is violated when the duration exceeds 180 days. The success predicate is a count threshold: $S(x)$ is true when fewer than three of the seven rules are violated, that is, $|I(x)| \leq 2$, and the project is classified as Failure when three or more rules are violated. A request is validated before evaluation: if any of the twelve parameters is missing or outside its permitted range (scales 1–5, percentages 0–100, counts and durations non-negative), the request is rejected and the user is returned to the form; missing values are therefore never imputed. Budget, team size, project complexity, client involvement and customer satisfaction are collected, validated and stored with every prediction, but in this version they do not trigger a rule; they are retained as inputs for the optional model path (Section V-E). Algorithm 1 summarises the procedure.

Algorithm 1. Rule-based project outcome classification

Input: parameter vector x; rule set $\{(pj, \rho j, \tau j)\}$, $j = 1 \ldots m$ ($m = 7$)
Output: label $\hat{y}$, triggered rules I, recommendations
1. Validate x; reject the request if a parameter is missing or outside its permitted range.
2. Set I to the empty list.
3. for $j = 1$ to m: if $x[pj]\ \rho j\ \tau j$ is violated then append j to I.
4. Evaluate the success predicate S(x): true if $|I| \leq 2$.
5. if S(x) is true then $\hat{y} \leftarrow$ Success else $\hat{y} \leftarrow$ Failure.
6. Look up the recommendation associated with each j in I.
7. return ($\hat{y}$, I, recommendations)

D. Risk indicators and recommendations

Explainability in the rule-based path is obtained directly: every triggered rule is displayed as a risk indicator with an associated recommendation. Table III gives the mapping together with the threshold of each rule.

TABLE III
RISK INDICATORS, THRESHOLDS AND RECOMMENDATIONS

Rule	Condition (violated when)	Risk dimension	Threshold	Recommendation shown
R1	Duration exceeds the defined limit	Schedule	Duration > 180 days	Review milestones and revise the schedule
R2	Repeated sprint delays	Schedule	Sprint delays > 3	Review milestones and revise the schedule
R3	Frequent requirement changes	Scope	Requirement changes > 5	Establish a requirement approval process
R4	High bug count	Quality	Bug count > 50	Increase testing and prioritise defect correction
R5	Low testing coverage	Testing	Testing coverage < 60%	Expand test cases and improve coverage
R6	Limited resource availability	Resource	Resource availability < 60%	Reassess allocation and workload
R7	Low team experience	Team capability	Team experience < 2 years	Provide guidance, training and technical support

The thresholds are fixed, configurable constants chosen from common project-management practice; they have not been calibrated against labelled outcomes (see Section VIII-C).

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

E. Optional data-driven path

To allow later comparison with learned models, the prediction service can also load a trained gradient-boosting classifier (XGBoost [6]) and a feature scaler, and can produce SHAP attributions [5]. SHAP expresses a prediction as an additive explanation

$$g(z') = \varphi_0 + \Sigma_{i=1}^{M} \varphi_i z'_i \quad (4)$$

where φ_0 is the base value, φ_i is the contribution of feature i and z'_i (0 or 1) indicates whether feature i is present [5]. Training requires an uploaded labelled dataset containing, at minimum, the columns budget, duration_days, team_size, team_experience_years, testing_coverage, bug_count and status (Section VII-D). No trained model was used for any result reported in this paper: every output shown here was produced by the baseline rule classifier of Section V-C. When a model is trained in future work, the intended setting is an XGBoost classifier trained on a stratified 80/20 train/test split with a fixed random seed, with hyper-parameters tuned by five-fold cross-validation on the training portion only.

F. Rationale for the rule-based core

TABLE IV
COMPARISON OF EVALUATION APPROACHES

Factor	Manual evaluation	Rule-based	Machine learning
Decision method	Judgement and experience	Predefined conditions	Patterns learned from data
Data requirement	Reports, observations	Parameters and defined rules	Suitable labelled history
Interpretability	Depends on the manager	High when rules are visible	Depends on the model
Adaptability	Human judgement	Rules must be updated	Retrainable with new data
Main limitation	Substantial manual effort	Cannot learn new patterns	Depends on data quality and validation

Table IV compares the three approaches. The rule-based approach is selected as the core because it needs no labelled corpus and yields explanations directly. Its weakness, that rules do not learn from history, motivates the optional data-driven path and the evaluation protocol below.

G. Evaluation protocol

With Failure taken as the positive class, predictive quality is to be measured on held-out labelled projects using

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN) \quad (5)$$

$$\text{Precision} = TP / (TP + FP), \quad \text{Recall} = TP / (TP + FN) \quad (6)$$

$$F1 = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall}) \quad (7)$$

together with the confusion matrix. The rule-based baseline and, if available, the trained model are to be compared on the same split.

VI. SYSTEM ARCHITECTURE / DESIGN

RiskPilot follows a layered design (Fig. 1). The presentation layer offers registration and login, a dashboard, project management, the prediction form, GitHub analysis and system reports. The application layer exposes authentication, project, prediction, dataset and GitHub-URL functions. The service layer contains the baseline rule classifier, the optional trained-model service and SHAP explainer, and the generator of risk indicators and recommendations. The data layer persists users, projects and dataset metadata. The presentation layer is a React single-page web client, the application layer is a Python FastAPI service, and persistence uses SQLAlchemy over a relational database (SQLite in the development setup).

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

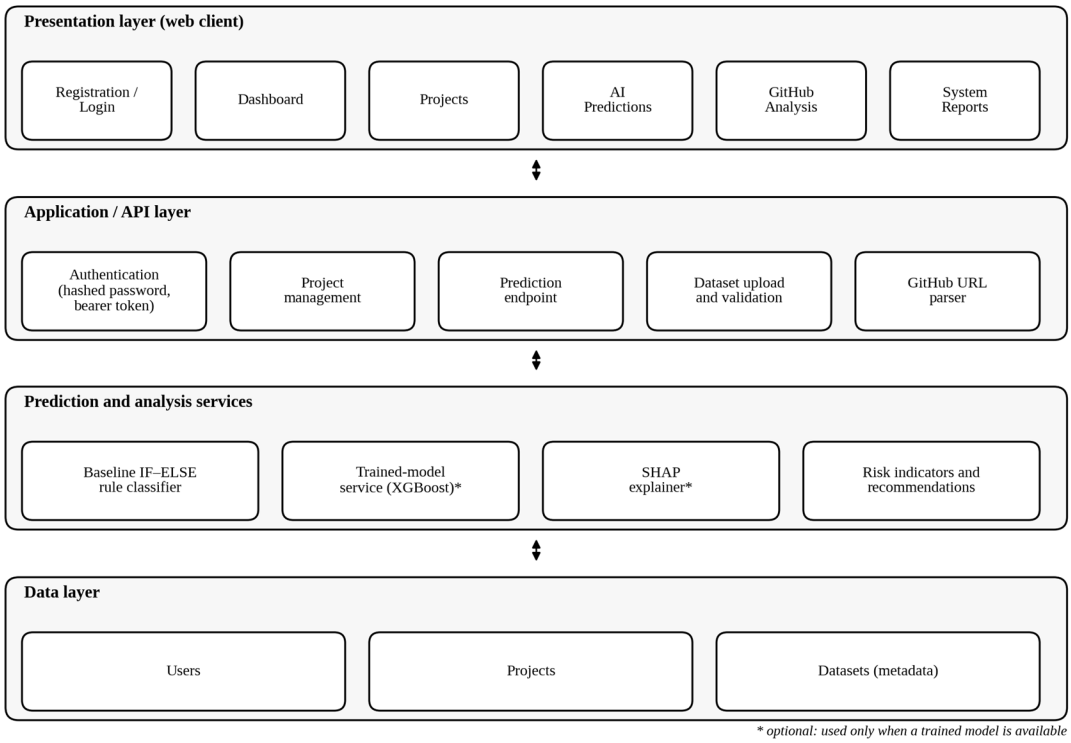


Fig. 1. Layered architecture of RiskPilot. Components marked * are used only when a trained model is available.

Fig. 2 shows the end-to-end workflow. After authentication the user opens the dashboard, creates or selects a project and enters the twelve parameters. Invalid input returns the user to the form; valid input is passed to the prediction service, and the result, risk indicators and recommendations are displayed, stored for the dashboard and history, and made available for report generation where enabled.

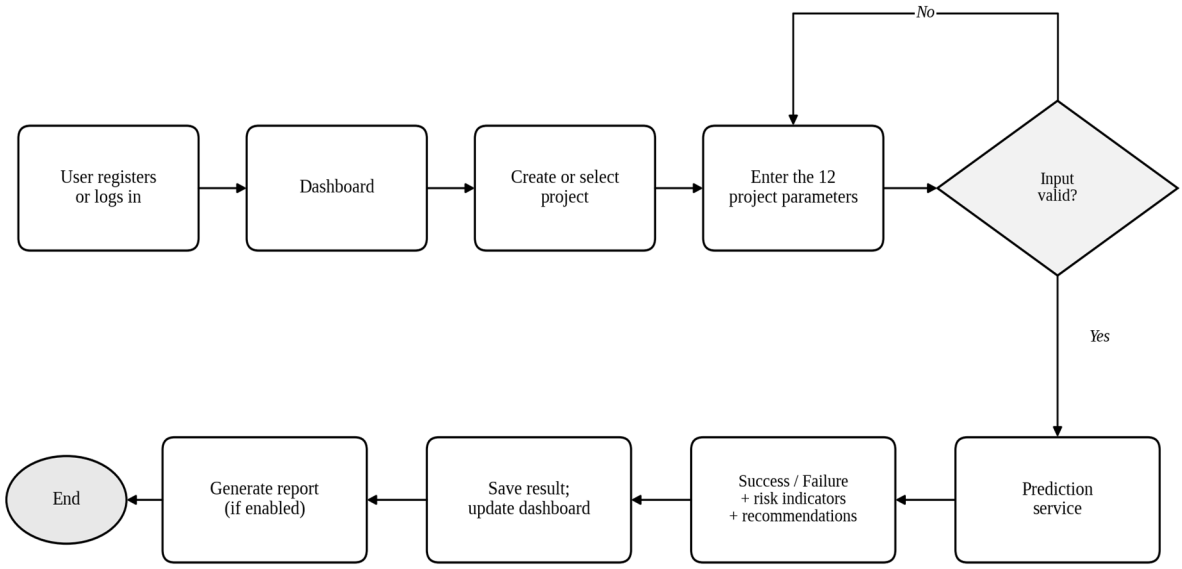


Fig. 2. End-to-end workflow from login to result display and reporting.

VII. IMPLEMENTATION

A. Technology and modules

The server-side code is written in Python with router-based API endpoints (FastAPI), an ORM (SQLAlchemy) for persistence and schema-validated request bodies; pandas is used to parse uploaded datasets. The database connection is configured through a single URL setting, so the same code can run on SQLite during development and on a server-based relational database in deployment. The client is a React single-page application. The system has been run as a local development deployment on the authors' machines and has not been deployed to a public hosting environment.

B. Authentication and roles

Registration rejects an e-mail address that already exists, stores only a hash of the password, and assigns the first registered account the role admin; later accounts receive the role manager or member according to the registration choice. Login verifies the password hash, rejects inactive accounts and returns a bearer access token with the user record. The registration form requires a password of at least eight characters that matches its confirmation.

C. Prediction service

The prediction service (Fig. 3) always attempts the baseline IF–ELSE prediction first. If a trained model with the requested name exists, it computes the model prediction and, when requested and when the model and scaler are available, a SHAP explanation; a failure in SHAP generation is recorded without discarding the prediction. The recommended prediction is the model prediction when one exists, and otherwise the baseline, with the method used recorded as “XGBoost ML (trained)”, “Baseline IF-ELSE (No trained model)” or “Baseline IF-ELSE (ML failed)”. The returned object carries a timestamp, the input metrics, both predictions, the SHAP output, the recommended prediction and the method used.

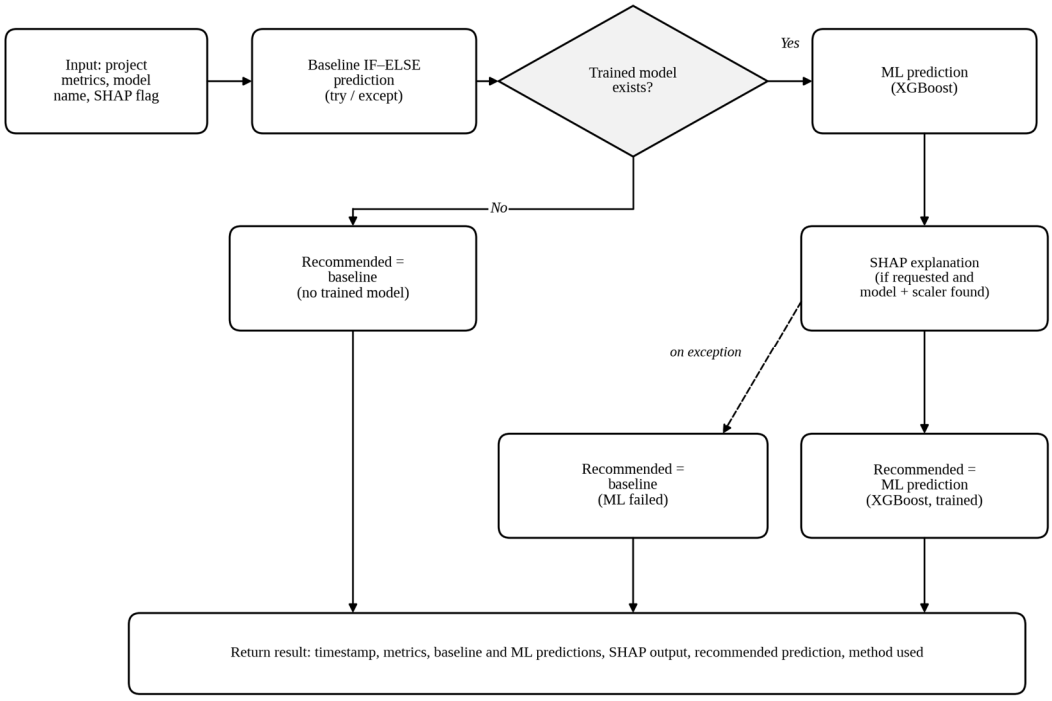


Fig. 3. Selection logic of the prediction service.

D. Dataset upload and validation

The dataset module (Fig. 4) accepts only CSV or Excel files, rejects unreadable or empty files with an error, counts missing values and duplicate rows, and checks for the required columns budget, duration_days, team_size, team_experience_years, testing_coverage, bug_count and status. The resulting feature report, including a flag stating whether the dataset is trainable, is stored with the filename, row count and uploading user.

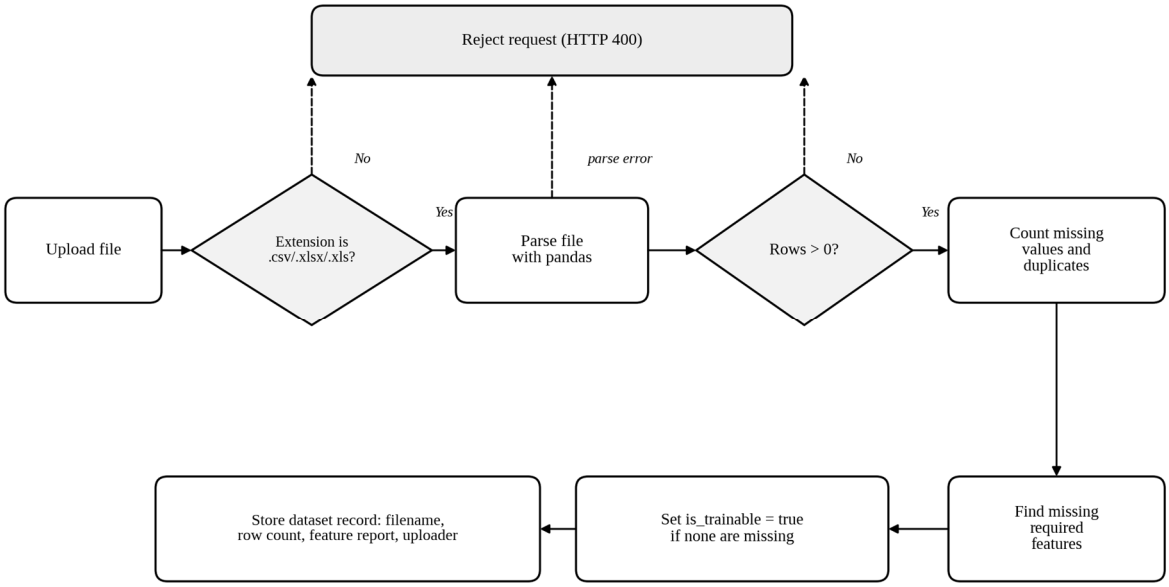


Fig. 4. Dataset upload and validation flow.

E. GitHub repository identification

The GitHub module extracts the repository owner and name from the common URL forms (SSH form git@github.com:owner/repo.git, HTTPS URLs on github.com, and a bare owner/repo string), ignoring a trailing .git suffix, and returns nothing for other hosts or malformed input. This provides the foundation for adding repository activity as an additional project indicator.

F. User-interface modules

The dashboard summarises total, successful, at-risk and failed projects and plots the risk distribution. The Projects page lists name, company, type, budget, test coverage, bugs, sprint delays, success probability, risk level and health, supports search by name or company, filtering by risk level and project type, adding projects and viewing archived projects. The AI Predictions page hosts the parameter form (Fig. 6). The System Reports page organises project information, outcomes and risk indicators in one consolidated on-screen view; export of this report to PDF and Excel is not yet implemented and is listed under future enhancements.

VIII. RESULTS AND DISCUSSION

A. Functional results

Fig. 5 shows the dashboard with a single test project in the portfolio (one active, none archived). The summary cards report one project in total, no successful project (0.0% portfolio success rate), one at-risk project and one failed project, and the risk-distribution chart places the project in the High-risk category. Fig. 6 shows the prediction form populated with sample values (budget 100000, duration 120 days, team size 8, team experience 5 years, project complexity 3, requirement changes 2, sprint delays 1, bug count 15, testing coverage 80%). Figs. 5 and 6 illustrate the interface and are not claimed to be the same entry.

As a worked example of the rule set in Table III, consider the sample values of Fig. 6. Duration (120 days) does not exceed 180 days, sprint delays (1) do not exceed 3, requirement changes (2) do not exceed 5, bug count (15) does not exceed 50, testing coverage (80%) is not below 60% and team experience (5 years) is not below 2 years, so R1–R5 and R7 are not violated. The resource-availability field lies below the visible part of the form in Fig. 6, so R6 depends on the value entered; even if R6 were violated, $|I| \leq 1 < 3$ and the success predicate S is satisfied. The rule-based outcome for these inputs is therefore Success, with at most one risk indicator (limited resource availability, with the recommendation to reassess allocation and workload).

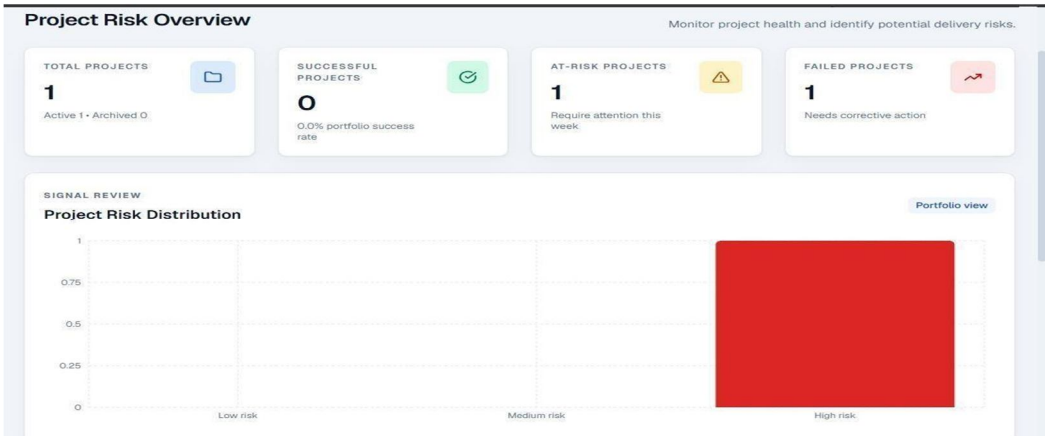


Fig. 5. RiskPilot dashboard showing portfolio summary and project risk distribution (implementation screenshot).

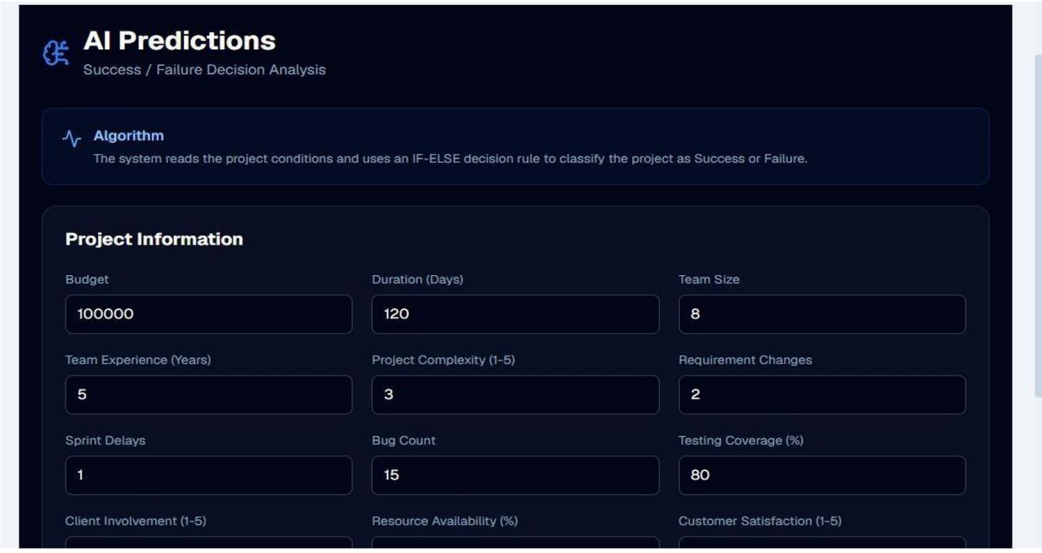


Fig. 6. AI Predictions page: project-information form for the IF–ELSE Success/Failure analysis (implementation screenshot).

B. Quantitative evaluation

No labelled evaluation has been run on the system so far, and no accuracy figure is claimed. Table V defines the results to be reported under the protocol of Section V-G and records their current status.

TABLE V
EVALUATION RESULTS TO BE REPORTED

Item	Rule-based baseline	Trained model (if used)
Dataset source and size	Not yet available: labelled project records have not been collected	Not yet available (no model trained)
Class balance, train/test split	Not yet available; planned stratified 80/20 hold-out	Not yet available; same split as the baseline
Accuracy	Not yet evaluated	Not yet evaluated
Precision / Recall / F1	Not yet evaluated	Not yet evaluated
Confusion matrix	Not yet evaluated	Not yet evaluated

C. Discussion

The principal strength of the design is that every output can be traced to explicit conditions: the same inputs and rules always yield the same label, and the triggered conditions are shown to the user. The fallback structure of the prediction service keeps the application usable when no trained model exists, while leaving room for the comparison that the literature suggests is worthwhile [4]. The limitations are equally clear. The rules and their thresholds have not been validated against labelled outcomes; rules do not learn new patterns; five of the twelve parameters do not currently trigger a rule; the parameter set was chosen from project-management practice and its influence on outcomes has not been independently established; and registration currently marks accounts as verified without a separate e-mail confirmation step. The system should therefore be read as a decision aid, not as a guarantee of any project's outcome.

IX. CONCLUSION

This paper presented RiskPilot, a web-based decision-support system that classifies software projects as Success or Failure from twelve parameters using transparent IF–ELSE rules, reports rule-linked risk indicators and recommendations, and is structured to admit a trained model with SHAP explanations. The architecture, prediction-service logic and implemented modules were described and the working interface was illustrated. Because predictive performance has not yet been measured on labelled project data, the paper does not claim a level of accuracy; the usefulness of the system depends on the quality of its rules, the accuracy of the entered data and the outcome of the evaluation set out in Section V-G.

X. FUTURE ENHANCEMENT

1) Collect labelled project data, then train and compare algorithms such as Random Forest and XGBoost against the rule-based baseline using accuracy, precision, recall, F1-score and confusion matrices.
2) Calibrate the rule thresholds and the success predicate against labelled outcomes, and extend the rules to the parameters that do not yet trigger one.
3) Add real-time monitoring of progress, budget utilisation, task completion and deadlines so that risk indicators update when parameters change.
4) Assign graded risk levels to individual factors and give more specific recommendations.
5) Extend GitHub analysis to commit frequency, pull requests, issue counts and development activity as supplementary indicators.
6) Present SHAP attributions in the interface so that users can see how each parameter influences a model-based prediction.
7) Add PDF and Excel export to the System Reports page and e-mail confirmation to registration.

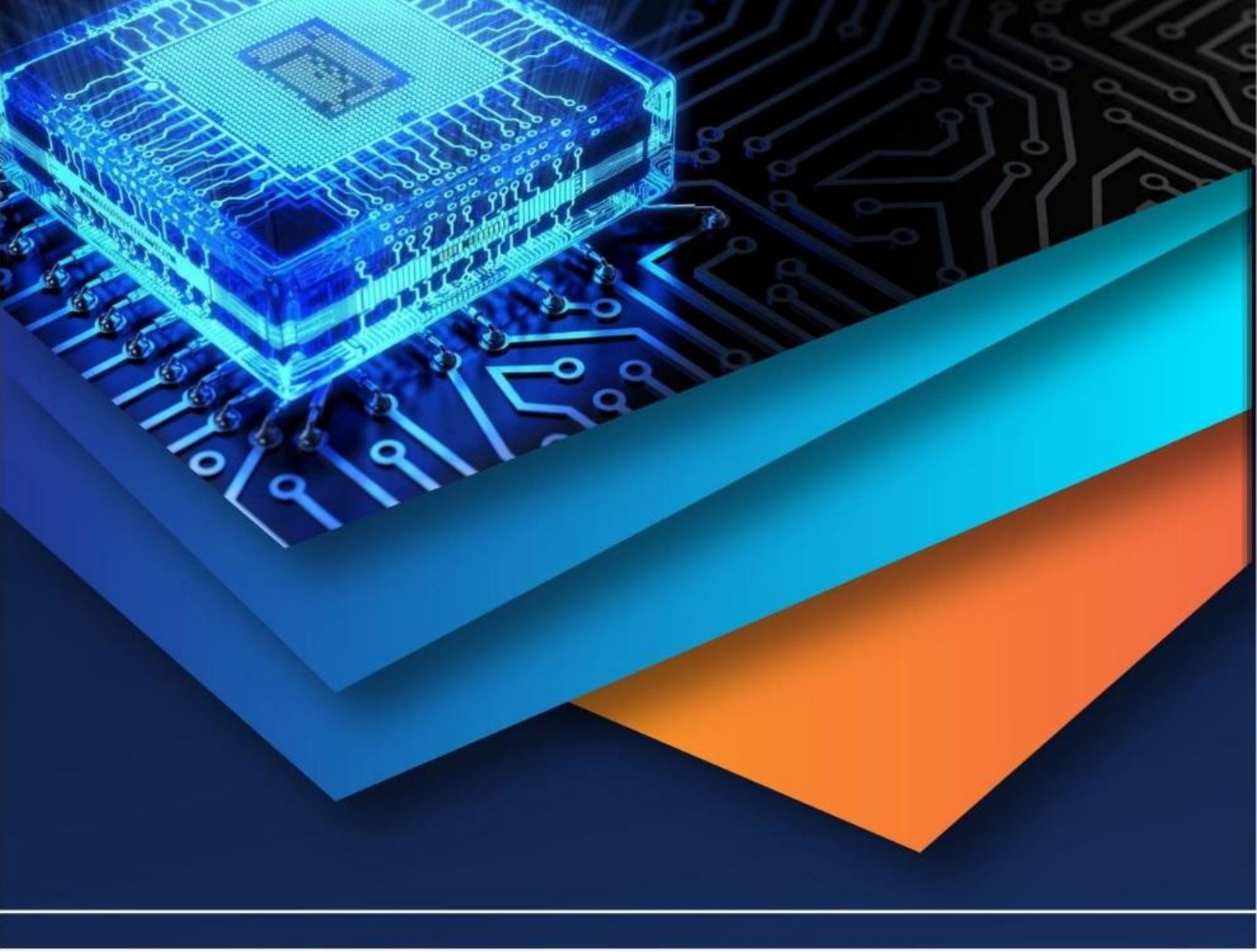
XI. ACKNOWLEDGEMENT

The authors thank Dr. R. Radhakrishnan, Principal, and Dr. G. Fathima, Professor and Head, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur, for providing the facilities and encouragement for this work, and acknowledge the guidance and suggestions of the project supervisor, Mrs. K. Yazhini, Assistant Professor, Department of Computer Science and Engineering.

REFERENCES

[1] T. Kawamura, T. Toma and K. Takano, “Outcome prediction of software projects for information technology vendors,” in Proc. IEEE Int. Conf. Industrial Engineering and Engineering Management (IEEM), Singapore, Dec. 2017, pp. 1733–1737.
[2] A. Nizam, “Software project failure process definition,” IEEE Access, vol. 10, pp. 34428–34441, 2022, doi: 10.1109/ACCESS.2022.3162878.
[3] R. G. Cooper, “Why AI projects fail: Lessons from new product development,” IEEE Eng. Manag. Rev., vol. 52, no. 4, pp. 15–21, 2024, doi: 10.1109/EMR.2024.3419268.
[4] G. D. Taye and Y. A. Feleke, “Prediction of failures in the project management knowledge areas using a machine learning approach for software companies,” SN Appl. Sci. (now Discover Applied Sciences), vol. 4, Art. no. 165, 2022, doi: 10.1007/s42452-022-05051-7.
[5] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017, pp. 4765–4774; arXiv:1705.07874.
[6] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)