Machine Learning-Based Effort Prediction and Early Risk Detection in Software Development Projects: A Case Study

Andreea-Elena Catana, Adriana Florescu
Department of Industrial Engineering and Management, Transilvania University of Brasov, Brasov, Romania

***Abstract*—Accurate effort estimation and early risk detection are critical for the success of software projects, as inaccurate forecasts can lead to schedule overruns, inefficient resource allocation, and unmet requirements. This study investigates the use of machine learning techniques to support task-level effort prediction and proactive risk identification in software project management. An applied case study was conducted on a simulated dataset of 500 software development tasks, described by planning, technical, and team-related features. Two ensemble-based regression models, Gradient Boosting and Random Forest, are evaluated for predicting actual task duration. Model performance is assessed using standard metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the coefficient of determination (R^2). To enable early risk detection, prediction errors are transformed into deviation-based indicators, and threshold-based classifiers are employed to identify tasks with moderate (>20%) and severe (>30%) schedule overruns. Confusion matrices and classification metrics are used to evaluate the effectiveness of the proposed alerting mechanism, and the distribution of high-risk tasks across sprint quantiles is analyzed to support managerial decision-making.**

***Keywords*—Software effort estimation; risk detection; machine learning; python; open data; gradient boosting; effort estimation; risk thresholding**

I. INTRODUCTION

Software development is a complex, time- and resource-consuming process of designing, implementing and maintaining software systems, which involves different stages. A software project manager's main duty is to ensure project completion on time and within budget, while guaranteeing high quality of deliverables and customer satisfaction, as projects may incur consequences of inefficiency. Therefore, managers generally need to answer important questions about software projects, including how long and how much effort it takes to complete a software project [1].

Project success greatly relies on the reliability of answering these questions, because budget overruns and overdue completion of software projects are universally reported. Poor estimation practices also cause declining client satisfaction and lead to a bad reputation, directly influencing the results of customers and vendors.

Effort estimation is also vital to software task scheduling and is a mandatory task before any software undertaking. Rescheduling and budget reallocation induced by missed deadlines may severely upset the quality of deliverables, and ultimatums of trade-offs between quality and cost factors will arise.

Therefore, automatically estimating or assessing the completion effort of software tasks given syntactically composed particle descriptions and histories is an emergent challenge. In this context, static analysis denotes a static process of examining software text code without actually executing programs. Specifically focusing on static analysis in tractable cycles has proven helpful in quality evaluation and runtime performance prediction of software systems.

However, existing static analysis methods are not able to quantify the completion effort of software tasks, and a demand for an accurate estimator that assesses the effort needed to complete software tasks prompt the emergence of automatic effort estimation techniques.

This study presents an integrated approach to project management using Machine Learning, focusing on the synergy between precise effort prediction and proactive risk identification, validated through a comprehensive case study.

The article is structured in seven sections, starting with an analysis of the current context in software development (Section I) and a review of recent literature on risk prediction and effort estimation (Section II). Subsequently, the research methodology and conceptual framework are defined (Section III), elements that underpin the development of the Machine Learning model for the analyzed case study (Section IV). The results and validity of the models are demonstrated in Section V, followed by a discussion of the practical implications for software development project management (Section VI). The study concludes by summarizing the conclusions and outlining future research directions (Section VII).

This study addresses the following research questions:

RQ1: Can machine learning models improve task-level effort prediction compared to planned estimates alone in a controlled software project environment?

RQ2: Can prediction deviations be operationalized into actionable early-warning indicators for schedule risk detection?

RQ3: How effectively can ensemble learning methods identify high-risk tasks based on development and team-related features?

II. Literature review

Artificial Intelligence (AI) is one of the most important areas of development, which brings new and innovative solutions to improve many tasks in the fields of software engineering, such as software reliability, maintenance, troubleshooting, effort estimation, resource management, and risk detection [2], [3], [4].

In recent years, the demand for software development needs has risen sharply in line with the booming growth of the Internet and online applications. To adapt to this boom and avoid software with less or poor quality, forecasters of software project work effort started to look for means or models to enhance their predictions, like AI-based predictive models, effort estimation frameworks, and AI-ready datasets for software endeavour estimation (SEE) [1].

Software Effort Estimation (SEE) is the process of estimating the effort required to develop a software system. AI-based predictive models [5], [6], including but not limited to Artificial Neural Network [7], Support Vector Regression, Gradient Boosted Tree, Random Forest, Extra Trees, XGBoost and hybrid modelling approaches were introduced [8], [9]. Effort estimation frameworks were proposed and implemented [10], [11], [12].

AI-ready datasets for SEE were made publicly available. Making use of the advantages of advanced machine learning analysis pipelines and the possible findings of other researchers' models is to deal with potential estimation cases with unseen projects [13]. Machine Learning (ML) is one of the most successful AI fields, which focuses on using data to train machines to learn and predict, classify, or make decisions without explicit programming [14], [15].

Table I summarizes a selection of recent literature from the last three years, highlighting the specific research areas and the particular challenges addressed in each study.

TABLE I. Summary of Specific Reviewed Papers 2023-2025

No.	Reviewed papers		
	Research Areas	***Problems Addressed***	***References***
1.	AI in software development projects	AI challenges and software tools in digital project management	[5], [6], [16], [17], [18], [19]
2.	Risk evaluation in software development	Integrated ML decisional models Risk and challenges	[9], [20], [21], [22]
3.	AI and ML models for risk predictions in software development projects	ML in risk predictions, risk detection ML in early estimation of the software functional dimension	[16], [23], [24], [25]
4.	AI and ML effort estimation models in software projects	ML models analysis and identification of stable models	[11], [26], [27], [28]

Recent literature review reveals a strong interest in integrating artificial intelligence (AI) into software project management. However, structural deficiencies emerge regarding the absence of a universally accepted methodology for implementing AI in risk management and the fragmentation of approaches according to specific domains. Furthermore, risk prediction and effort estimation face technical limitations related to the use of datasets.

In this context, the present scientific work aligns with current research trends, contributing to the development of the software project management domain through methodological and applied contributions regarding the accurate prediction of development effort and the early identification of high-risk software issues, leveraging AI techniques.

Despite the extensive research on machine-learning-based effort estimation, several limitations remain. First, many studies focus exclusively on effort prediction accuracy without integrating risk detection mechanisms derived from prediction deviations. Second, prior work often evaluates models independently of their operational value in project management decision support. Third, relatively limited attention has been given to the transformation of regression outputs into interpretable early-warning indicators. [1], [5], [11]

This study addresses this gap by integrating effort prediction and threshold-based risk detection within a unified experimental framework, providing both predictive and actionable insights.

III. Methodology

Software development has become one of the key objectives for countries and companies in order to sustainably develop an economy and improve productivity. Software projects, however, are faced with many risks, leading to many largely failed projects. Risk assessment has been a hot issue in both theory and practice. Moreover, effort estimation is one of the most critical and challenging tasks in software project management.

Although software effort estimation has received extensive research attention in the last few decades, effort estimation is still a challenging issue [29], [30]. Many researchers have focused on various traditional and machine learning techniques for estimating software effort and project quality. A guide to good practices in risk management is presented in the study [31], which shows current theoretical models selected from the specialist literature, highlighting existing gaps, while developing practical applications for case studies in the automotive industry.

This research work combines the two important tasks, risk detection and effort estimation, in an integrated framework. The objective of the research was to investigate the idea of combining risk detection and effort estimation in the same model, using an end-to-end AI model.

Solving these tasks separately can bring some important information to project stakeholders. However, software project risks and efforts are closely related and can affect each other. This combination can allow the detection of a hidden class of both tasks, automatically detect the duration of each task, and traverse it recursively across the project history.

A. Implementation Framework

The proposed conceptual framework consumes the project repository as input and generates project risks and gathered exhibit information as output. Several key design components, such as data representation, risk models, predictive models, and evidence gathering, will be detailed to deliver how the framework accomplishes its objectives (Fig. 1).

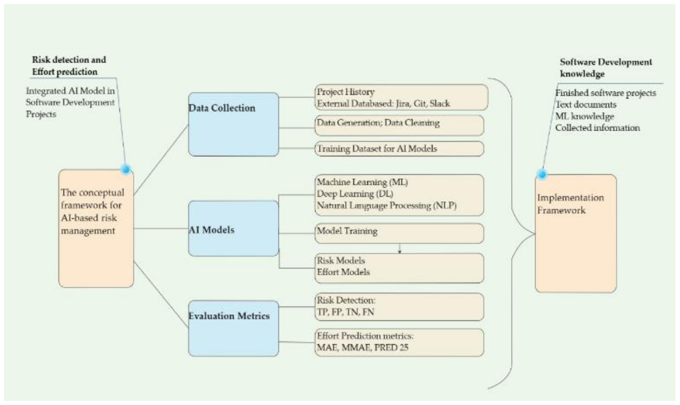
Risk detection and Effort prediction
Integrated AI Model in Software Development Projects
The conceptual framework for AI-based risk management
Data Collection
Project History
External Databased: Jira, Git, Slack
Data Generation; Data Cleaning
Training Dataset for AI Models
AI Models
Machine Learning (ML)
Deep Learning (DL)
Natural Language Processing (NLP)
Model Training
Risk Models
Effort Models
Evaluation Metrics
Risk Detection:
TP, FP, TN, FN
Effort Prediction metrics:
MAE, MMAE, PRED 25
Software Development knowledge
Finished software projects
Text documents
ML knowledge
Collected information
Implementation Framework

Fig. 1. Implementation framework.

In this framework, all these design components cooperate to maximize the detection ability of the model. Projects are represented using project repository data and converted into a joint representation framework, including a bipartite graph and a temporal representation.

B. Data Collection

The research project concerns the detection of risk factors that are likely to endanger the delivery of software systems on time or with the required quality, as well as the prediction of software effort adjustment for associated risk factors. The project mainly involves the innovation of two prototypes, one which performs risk detection and the other which performs risk prediction and visualization.

The data used has been generated using increments for planned hours, complexity of the task, team experience in years and requirements clarity, with a total of 500 rows used, where the first 20 have been exemplified in Table II. The simulated dataset was designed to reflect relationships observed in software project environments. Planned effort was modelled as the primary baseline predictor, while actual effort was generated as a function of planned effort, task complexity, team experience, and requirement clarity, combined with stochastic noise. The simulation assumes that higher complexity and lower team experience increase actual effort, while clearer requirements reduce uncertainty.

This controlled simulation approach allows the evaluation of model behaviour and risk detection mechanisms under known ground-truth conditions, which is consistent with experimental practices in predictive modelling research.

TABLE II. SAMPLE OF 20 ROWS OF TOTALLY USED DATA FROM THE 500 ROWS

Task ID	**Data used**				
	Sprint ID	***Planned hours***	***Complexity***	***Team experience***	***Requirement clarity***
1	4	62.37	5	5.62	1
2	3	36.23	4	2.21	1
3	5	68.97	3	2.98	3
4	4	56.39	5	7.48	4
5	1	9.35	1	0.86	3
6	5	78.10	5	3.96	1
7	4	61.37	5	5.84	1
8	5	63.31	3	1.63	5
9	1	11.99	4	0.86	5
10	3	37.13	3	1.54	4
11	2	30.92	2	7.39	1
12	5	74.29	4	0.57	5
13	4	52.22	2	1.91	4
14	5	66.18	2	0.73	3
15	3	36.59	3	1.33	1
16	2	29.85	4	5.15	5
17	1	8.41	2	2.44	2
18	2	27.66	3	6.12	4
19	3	34.98	1	3.87	2
20	1	7.92	2	1.05	4

C. Machine Learning and Models used

This study investigates the application of machine learning (ML) techniques to support effort prediction and early risk detection in software project management. The objective is to accurately estimate task-level development effort and to identify tasks with a high probability of schedule overrun, enabling proactive decision making before and during project execution.

Recent research has demonstrated the effectiveness of ML-based regression and classification models for software effort estimation and project risk analysis, particularly when complex, non-linear relationships exist between task characteristics, team factors, and execution outcomes [8], [11], [16]. Ensemble learning methods, such as Random Forest and Gradient Boosting, have been shown to provide robust performance and good generalization in such settings due to their ability to combine multiple weak learners and reduce variance and bias [26].

In this research study, two tree-based ensemble models are employed: Gradient Boosting and Random Forest. These models are trained on simulated task-level data described by planning, technical, and team-related features. Their performance is evaluated using standard regression metrics and compared in order to identify the most accurate and stable predictor for actual task duration.

D. Evaluation and Risk Detection Metrics

To assess both effort prediction accuracy and risk detection capability, standard metrics from regression and classification literature were employed, as commonly used in software engineering analytics and project management studies [1], [32].

For the evaluation of threshold-based risk classification, the following measures derived from the confusion matrix are used [33]:

- True Positive (TP): Number of truly high-risk tasks correctly identified as high-risk.
- False Positive (FP): Number of low-risk tasks incorrectly classified as high-risk.
- True Negative (TN): Number of low-risk tasks correctly identified as low-risk.
- False Negative (FN): Number of high-risk tasks incorrectly classified as low-risk.

Based on these values, Precision, Recall, F1-score, and Accuracy are computed to evaluate the effectiveness of the early-warning mechanism.

E. *Effort Prediction Metrics*

The predictive accuracy of the regression models is evaluated using the following metrics, which are standard in software effort estimation research [1], [16]:

- Mean Absolute Error (MAE): Measures the average absolute deviation between predicted and actual effort.
- Root Mean Squared Error (RMSE): Penalizes large errors and reflects overall prediction dispersion.
- Coefficient of Determination (R^2): Indicates the proportion of variance in actual effort explained by the model.

These metrics provide complementary views of model performance in terms of average error magnitude, sensitivity to outliers, and explanatory power.

Planned effort represents the initial human estimate defined during project planning, before task execution. Actual effort reflects the true execution time observed after task completion. The machine learning model uses planned effort together with additional contextual features, such as task complexity, team experience, and requirements clarity, to predict actual effort more accurately. The difference between planned and actual effort represents estimation error, while the difference between predicted and planned effort serves as an early indicator of potential schedule risk.

The proposed framework integrates effort prediction and risk detection into a unified, data-driven decision-support workflow for software project management. Task-level features are extracted from project planning and execution data and used as inputs to the trained regression and classification models. Predicted effort values are compared with planned estimates to derive deviation-based risk indicators, and threshold-based rules are applied to flag moderate and severe overruns.

The framework provides numerical predictions, confusion-matrix-based evaluation, and visual analytics, including learning curves, validation curves, feature importance plots, and risk distribution across sprint quantiles. Such integration of predictive modeling with interpretable outputs supports project managers in understanding both the expected effort and the underlying drivers of schedule risk.

This approach is consistent with recent trends in data-driven project analytics, which emphasize the combination of machine learning models, transparent evaluation metrics, and visualization to enhance trust, usability, and practical adoption in real-world software project environments [4], [34].

IV. Model Development

This section describes the development of the machine-learning-based framework for task-level effort prediction and early risk detection. The proposed approach follows a structured workflow that includes data preparation, feature definition, model training, and hyperparameter tuning, in line with established practices in software analytics and predictive modeling [1], [16], [35].

A. *Data Preprocessing*

Data quality and representativeness are critical for the reliability of any data-driven model. In the present study, a simulated dataset of 500 software development tasks is used to ensure controlled experimentation and the availability of ground-truth values for both actual effort and deviation-based risk labels. The dataset includes planning, technical, and team-related attributes, enabling the analysis of heterogeneous factors that influence task duration and schedule risk.

Before model training, exploratory analysis is performed to examine feature distributions, correlations, and potential outliers. The dataset is then partitioned into training and test subsets using an 80/20 split, ensuring that the proportion of overrun and high-risk tasks is preserved in both sets. This leakage-free evaluation protocol follows recommendations from previous work on effort and defect prediction to ensure credible and unbiased performance assessment [1], [36].

B. *Feature Selection*

Feature engineering and selection play a crucial role in software effort estimation. Prior studies have shown that task size, complexity, developer experience, and requirements-related factors are among the most influential drivers of development effort [8], [16], [37]. In this work, the feature set includes planned effort, task complexity, team experience, requirements clarity, number of dependencies, story points, and structural complexity indicators.

Rather than applying an explicit wrapper-based feature subset search, the importance of individual predictors is analyzed using model-intrinsic feature importance measures derived from tree-based ensembles. This approach provides an interpretable ranking of features and avoids the computational overhead and potential overfitting associated with exhaustive subset selection methods [37], [38]. The resulting importance analysis supports both model understanding and managerial interpretation.

C. *Training the Models*

Two ensemble learning algorithms are employed for effort prediction: Gradient Boosting and Random Forest. These methods are well-suited for tabular software engineering data and are known for their ability to capture non-linear relationships and feature interactions while maintaining good generalization properties [26], [39].

Models are trained on the training subset and evaluated on the held-out test set using standard regression metrics. Predicted effort values are subsequently transformed into deviation-based indicators to support threshold-based risk classification. This two-stage procedure enables the integration of continuous effort forecasting with discrete early-warning signals for schedule overruns.

D. Hyperparameter Tuning

The performance of ensemble models depends on a set of hyperparameters that control model complexity and learning dynamics, such as the number of trees, maximum tree depth, and learning rate. Hyperparameter optimization is therefore conducted using cross-validation-based search strategies, following established guidelines in machine learning research [40], [41].

For each model, a range of candidate values is defined for the most influential hyperparameters, and validation curves are analysed to identify configurations that provide a favourable trade-off between bias and variance. Learning curves are additionally employed to assess how predictive performance evolves with increasing training set size and to verify the absence of severe overfitting.

This systematic tuning and validation process ensures that the selected Gradient Boosting and Random Forest models achieve robust generalization performance and provides empirical justification for the final parameter settings used in the experimental evaluation.

V. Results and Discussion

A contribution of this research is the definition of a taxonomy of the identified problems and their classification, based on the previous study of the specialized literature. A set of state-of-the-art Machine Learning algorithms was selected for each problem to benchmark the best-performing one. The categorization scheme can be used for other scholars and practitioners to classify research problems in software engineering in general, not just Machine Learning-based ones. It is required that, for each type of problem to be tackled by Machine Learning, the problem taxonomy be specified.

The strong importance of planned hours does not represent target leakage, as planned effort is defined before task execution and reflects the initial human estimate. In practical software project environments, planned effort serves as a baseline estimate, while actual effort deviates due to technical and organizational factors.

To predict the actual task duration, two ensemble-based regression models were compared: Gradient Boosting and Random Forest. The results indicate that the Gradient Boosting model achieved the best overall performance, with the lowest MAE and RMSE and the highest R² (Fig. 2–4). This demonstrates the model's ability to capture the complex relationships between task features and actual duration, significantly reducing deviations from planned estimates.

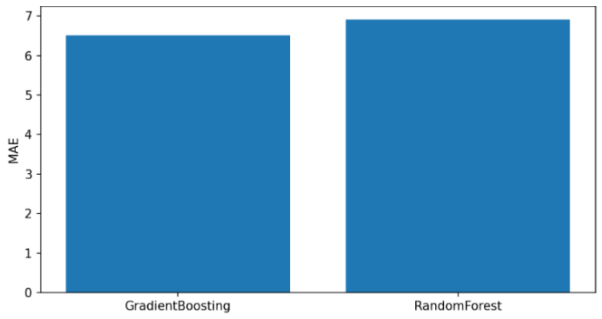

Fig. 2. Comparison of MAE.

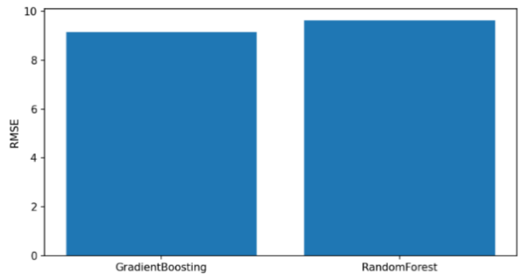

Fig. 3. Comparison of RMSE.

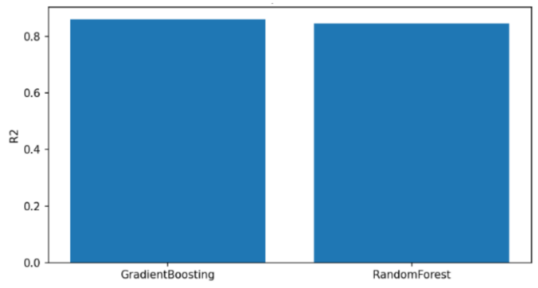

Fig. 4. Comparison of R^2.

A detailed analysis of the optimized Gradient Boosting model shows a strong correlation between predicted and actual values (Fig. 5). The residuals distribution is centered around zero with relatively low dispersion, confirming that the model does not introduce systematic errors and achieves robust accuracy across most tasks.

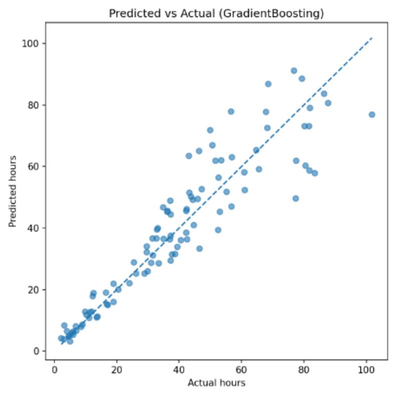

Fig. 5. Comparison between predicted and actual values.

The distribution of prediction errors was analyzed through the residual plot of the best-performing model (Fig. 6), which shows that most errors are centered around zero, indicating good calibration, while a limited number of tasks exhibit larger deviations, reflecting the inherent uncertainty of software development processes.

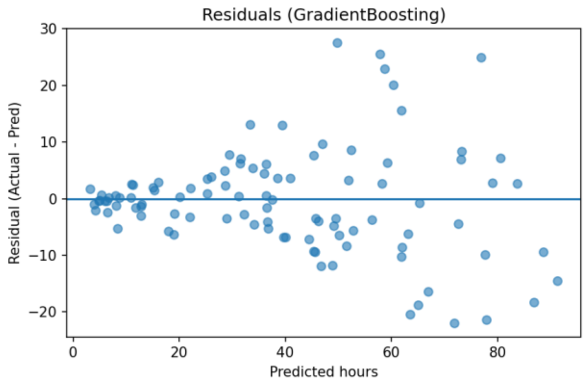

Fig. 6. Residuals of the gradient boosting model.

A. Risk Detection Outcomes

Early identification of schedule risk is essential for effective project control, as it enables timely corrective actions before deviations propagate at the project level [42]. In this research, risk is defined in operational terms, based on the deviation between predicted and planned effort, and is assessed using threshold-based classification supported by machine learning.

Risk labels are not directly learned from regression outputs but are derived from deviation thresholds between predicted effort and planned effort, reflecting operational definitions of schedule overrun risk.

This approach mirrors real-world project monitoring practices, where deviations from planned estimates serve as early-warning indicators. The classification stage, therefore operationalizes deviation thresholds rather than introducing an independent predictive target.

Two risk levels are considered. First, an overrun event is defined when the predicted effort exceeds the planned hours by more than 20%. Based on this criterion, a confusion matrix is computed on the test set of 100 tasks (Fig. 7), allowing the identification of true positives, false positives, true negatives, and false negatives. The results show that the majority of non-overrun tasks are correctly identified, while a subset of true overruns is successfully detected, providing a practical early-warning mechanism with a controlled false-alarm rate.

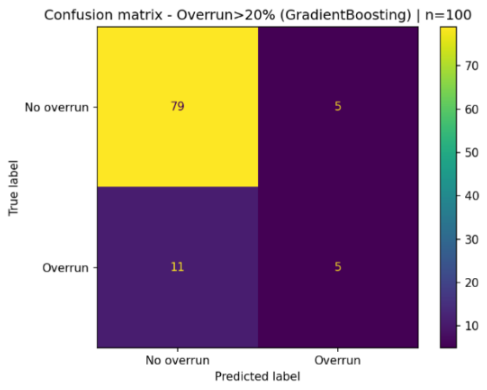

Fig. 7. Confusion matrix for schedule overrun detection based on a 20% deviation threshold between predicted and planned effort.

Second, a high-risk category is defined for severe deviations exceeding 30% of the planned effort. A Random Forest classifier is trained using task features and model-derived error indicators to discriminate between low-risk and high-risk tasks.

The corresponding confusion matrix and classification report (Fig. 8) indicate high overall accuracy and good recall for the critical class, demonstrating the model's ability to highlight tasks that are most likely to cause significant schedule disruption.

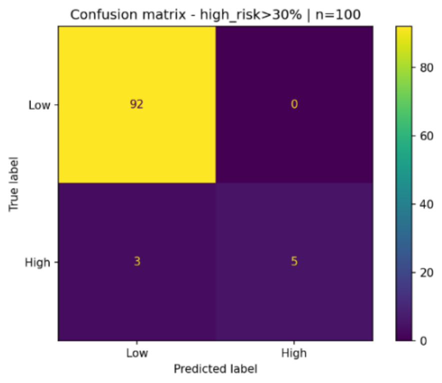

Fig. 8. Confusion matrix for high-risk task classification using a 30% deviation threshold between predicted and planned effort.

In addition, the distribution of predicted and actual high-risk tasks across sprint quantiles (Fig. 9) reveals that higher workload segments tend to concentrate a larger proportion of risky tasks. This analysis supports the use of risk-aware planning and monitoring strategies, enabling project managers to allocate attention and mitigation resources to the most critical parts of the schedule.

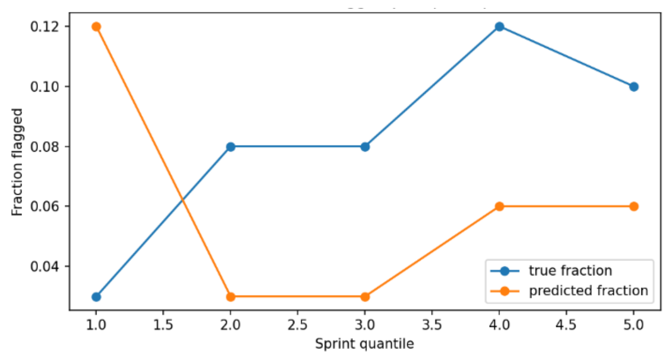

Fig. 9. Distribution of true and predicted high-risk tasks across sprint quantiles.

Overall, the proposed risk detection approach transforms effort prediction errors into interpretable and actionable indicators, linking regression performance directly to operational decision support through threshold-based alerts and confusion-matrix-driven evaluation.

B. Effort Prediction Accuracy

The accuracy of the proposed effort prediction models was evaluated using standard regression metrics, namely Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the coefficient of determination (R^2). Two ensemble-based regressors, Gradient Boosting and Random Forest, were trained and tested on the simulated dataset using an 80/20 train–test split.

The comparative results, reported in Fig. 1–3, show that the Gradient Boosting model achieved the best overall performance, with lower MAE and RMSE values and a higher R^2 compared to Random Forest. These results indicate a superior capability of the model to capture the non-linear relationships between task characteristics and actual development effort.

A detailed analysis of the predicted versus actual values (Fig. 4) reveals a strong alignment around the diagonal, confirming good calibration across the full range of task durations. The residual plot (Fig. 6) further shows that most errors are centered around zero, with no evident systematic bias, demonstrating robust predictive behaviour.

Feature importance analysis for the best-performing Gradient Boosting model (Fig. 10) highlights planned_hours as the dominant predictor of actual effort, followed by task complexity, team experience, and requirements clarity. This ranking is consistent with software project management practice, where initial estimates provide the baseline, while technical difficulty and human factors significantly influence execution outcomes.

The feature importance analysis (Fig. 10) highlights that planned_hours is the dominant predictor, as expected in a project management context. However, other factors such as task complexity, team experience, and requirement clarity also contributed to prediction accuracy, emphasizing their relevance in anticipating actual task duration.

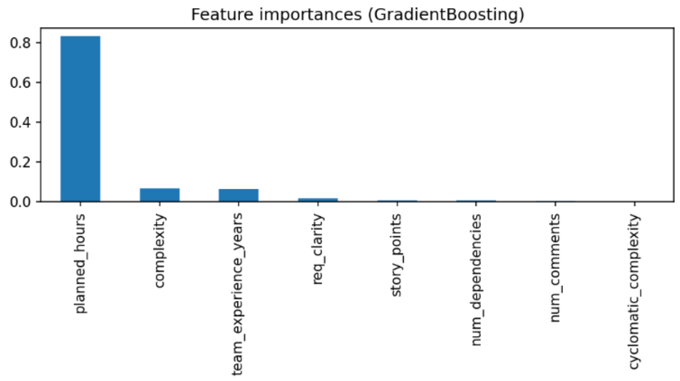

Fig. 10. Importance of features used.

In addition to task duration prediction, a classifier was developed to identify high-risk tasks likely to exceed their planned duration. The risk classifier achieved an overall accuracy of approximately 95% and a recall of 0.80 for the high-risk class, indicating a strong capability to detect critical situations (Table III).

The fraction of flagged tasks per sprint quantile (Fig. 9) further illustrates differences between the true and predicted distributions of high-risk tasks, highlighting where early risk detection can be improved.

TABLE III. PERFORMANCE OF THE HIGH-RISK TASK CLASSIFIER (DEVIATION > 30%)

Class	**Performance of the High-Risk Task**			
	Precision	***Recall***	***F1-score***	***Support***
0 (not high-risk)	0.97	1.00	0.98	92
1 (high-risk)	1.00	0.62	0.77	8
Accuracy			0.97	100
Macro avg	0.98	0.81	0.88	100
Weighted avg	0.97	0.97	0.97	100

To analyze generalization and model stability, learning and validation curves were computed using cross-validation. Fig. 11 presents the learning curve of the Gradient Boosting model, showing the evolution of training and validation error as the size of the training set increases, and indicating good convergence without severe overfitting.

Fig. 12 reports the validation curve for the Random Forest model with respect to the number of trees, highlighting the trade-off between model complexity and generalization performance. These analyses confirm the robustness of the selected ensemble models and support the choice of hyperparameters used in the experimental evaluation.

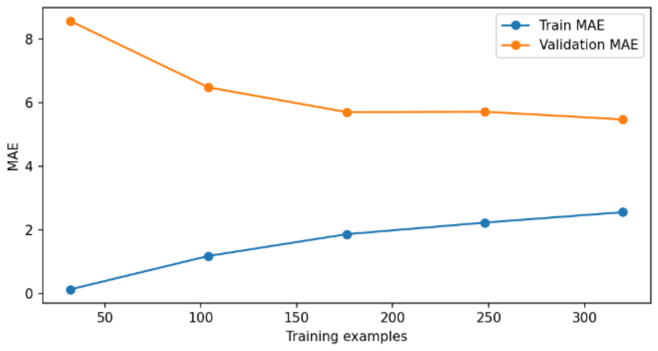

Fig. 11. Learning curve of the gradient boosting regressor showing the evolution of training and validation MAE as a function of the number of training samples.

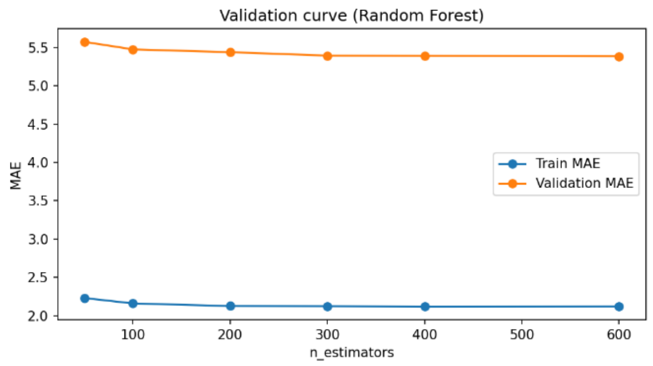

Fig. 12. Validation curve of the random forest regressor illustrating the impact of the number of trees (n_estimators) on training and validation MAE.

The analysis of feature importance (Fig. 10) revealed that planned hours is the dominant predictor, which is intuitive since initial estimates usually provide the baseline for task planning. However, secondary factors such as task complexity, requirements clarity, and team experience also contribute to prediction accuracy, highlighting the relevance of both technical and human-related aspects in effort estimation. This indicates that integrating quantitative planning variables with qualitative project characteristics improves the robustness of predictive models.

The residual analysis of the best-performing Gradient Boosting model (Fig. 6) confirms that most prediction errors are centered around zero, with no evident systematic bias, demonstrating stable and well-calibrated predictions. Nevertheless, a limited number of tasks exhibit larger deviations, suggesting the presence of latent factors (e.g., unexpected blockers, shifting priorities, or external dependencies) that are not explicitly captured by the available features.

In terms of risk classification, the proposed approach achieved encouraging results, with an overall accuracy of approximately 97% and a recall of 0.62 for the high-risk class (Table III and Fig. 10). This indicates an effective capability to detect most critical tasks, which is particularly valuable for project managers aiming to mitigate delays at an early stage. For the high-risk class, the model achieved a precision of 1.00 and a recall of 0.62. This indicates that while the model is precise in identifying high-risk tasks, it fails to detect a portion of them, as reflected by the lower recall. From a practical perspective, this suggests that when the model predicts a task as high-risk, the prediction is highly reliable, although some high-risk tasks may remain undetected.

Taken together, these findings demonstrate that machine-learning-based effort prediction combined with threshold-based risk classification can provide valuable decision-support for project managers, enabling more accurate time planning and proactive risk mitigation.

Future work should focus on improving generalization by incorporating additional contextual information, such as team workload, communication patterns, and organizational constraints, as well as on integrating explainable AI techniques to enhance transparency and trust in model-driven recommendations.

C. Comparison with Traditional Methods

AI-based techniques bringing enormous effectiveness are into considerations and analysis. For the same purpose, traditional techniques such as analogy-based ones can be analyzed which have been proved effective in many fields, including risk management and cost/effort estimation in software projects [43]. Performance-wise, techniques like random forest and ensemble models outperform the rest when used without optimization.

However, they become less competitive when hyper-parameter optimization is explored. On the contrary, tree-based methods exhibit different performance with hyper-parameter optimization involved. Such differences might be due to the overfitting issue with tree-based methods.

In addition to predictive accuracy, computational efficiency is an important aspect when considering the practical deployment of machine learning models in project management environments. Although ensemble methods such as Gradient Boosting and Random Forest can deliver accurate predictions, their training and hyperparameter tuning require a higher computational effort compared to simple baseline models. This computational cost depends on dataset characteristics, including the number of tasks, the dimensionality of the feature space, and the complexity of the model search space.

In the present study, the dataset size (500 tasks) and the moderate number of features allow the models to be trained within a reasonable time, while cross-validation and parameter optimization still represent the most computationally demanding steps. This confirms that, in data-driven project analytics, model selection and tuning play a crucial role not only in terms of accuracy but also in terms of computational resources.

Table IV summarizes the regression models evaluated for predicting actual task duration. Among the considered approaches, the Gradient Boosting regressor achieved the best overall performance, with the lowest MAE and RMSE and the highest R^2, while the Random Forest model showed slightly lower but comparable results.

These findings indicate that tree-based ensemble methods provide a favorable balance between predictive accuracy and computational feasibility for task-level effort estimation in software project management.

TABLE IV. MODEL PERFORMANCE COMPARISON AND HYPERPARAMETER OPTIMIZATION

Regression Models	Effort Prediction Metrics		
	MAE	*RMSE*	*R^2*
Gradient Boosting	6.51	6.51	0.86
Random Forest	6.91	6.91	0.84

D. *Limitations of the Study*

Despite the encouraging results obtained with the proposed machine-learning-based framework, several limitations of the present study must be acknowledged.

First, the experimental evaluation is conducted on a simulated dataset. Although the simulation was designed to reflect realistic relationships between planning variables, team characteristics, and actual effort, it cannot fully capture the complexity, uncertainty, and organizational dynamics of real industrial software projects. Consequently, the reported performance metrics should be interpreted as indicative of the potential of the approach rather than as definitive benchmarks.

Second, the analysis is restricted to task-level static features and does not explicitly model temporal dependencies, evolving project context, or interactions between tasks over time. Factors such as changing priorities, resource reallocation, communication patterns, and external constraints may significantly influence effort and risk, but are not represented in the current feature set.

Third, risk detection is based on fixed deviation thresholds (20% for overrun and 30% for high-risk classification). While such thresholds are commonly used in practice and allow the construction of interpretable confusion matrices, optimal threshold values may vary across organizations, project types, and risk tolerance levels. More adaptive or cost-sensitive thresholding strategies could further improve the practical relevance of the alerts.

Finally, only two ensemble models, Gradient Boosting and Random Forest, were considered in the current implementation. Although these models showed strong predictive performance, other regression and classification techniques, as well as hybrid and temporal models, may provide additional insights and should be investigated in future work.

The present study does not include explicit comparisons with simple baseline predictors such as mean estimation or linear regression. Future work will include such baselines to quantify incremental predictive improvements provided by machine learning models.

As another point for future research, it will focus on validating the proposed framework on real-world industrial datasets, extending the feature space with temporal and organizational variables, and integrating explainable AI methods to enhance transparency and trust. Such extensions are expected to strengthen the generalization capability of the models and to further support their adoption as decision-support tools for effort prediction and early risk detection in software project management.

VI. IMPLICATION FOR PRACTICE

The following recommendations are conceptual and derived from the observed predictive patterns and risk detection framework. They are intended to illustrate potential practical applications rather than to represent empirically validated interventions through user studies or industrial deployment.

Based on the issues in the assessment, the following recommendations can be considered, in specific steps [1]:

1) Overcoming the lack of assessment breaks: To overcome this obstacle, such an organization working on a software project management should time-box efforts to incorporate risk assessment alongside code reviews and design discussions. As a tangible strategy, a small postmortem section could be added as a code review template. If efforts are maintained despite tight timelines, a continuously available assessment with adequate success measures is trivial.

2) Communicating effort estimation and improvement properties: This could be a challenge given the differences in individual engineering types, development environments, and product types. However, efforts to make experiences and properties public could still be undertaken by bundling characteristics to common classes and focusing on visualization styles that communicate relative differences, such as spider charts or illustrative statistics. Additionally, risk assessment as a service could be converted to more general terms.

3) Mitigating limited development time: While development time generally presents a significantly higher system importance for high-performing traditional teams, such an organization working on software project management still has many short- to medium-term development priorities. Building on continuous lack-of-time provisions, urgent and easily implementable slam-dunk uses, such as enabling one-click creations and using fully loaded default configurations, should be prioritized.

4) Providing a starting use for initial negative perception: To overcome this difficulty, a customized start with familiarization sessions is essential. As a start, before implementations, several risk assessments could be conducted based on historical data from past development processes. In addition, features matching the working style of existing platforms should be prioritized.

5) Getting out of the necessity loop: While the belief that risk assessment necessarily needs to be overcome, a potential shift in tasks to a more exploratory state should be encouraged. However, to soothe the fear of losing a successful assessment loop, supporting functions for incremental advances, such as early warnings, visualizations of development processes, and use encouragement, could be prioritized.

6) Overcoming a negative perception of AI-based approaches: Before AI incorporation, natural-language-based misunderstanding mitigations could be supported. Humans still need to completely classify issues and distinguish domain-specific uses based on individual understanding. Possible augmentations include focused simulations based on recorded engineering tickets that fall below a certain length, as well as adjustments to existing screening methods for static analysis issues.

A. *Integrating AI in Project Management*

The rise of AI brings new opportunities and challenges to the discipline of project management. AI has the potential to significantly transform project management by assisting project managers and team members in various ways, such as automating repetitive, high-volume tasks of low business value to free resources for more productive work, mining big data repositories to provide project analytics for team formation, effort estimation, risk prediction, and plan development, providing actionable recommendations, and providing decision reports for project managers to understand the implications of the recommendations [44].

Despite these potential transformations, the successful adoption of AI in project management is lower than that of other software engineering activities. Understanding the factors that influence the intention to adopt AI and the various readiness factors is essential for researchers and software management practitioners in facilitating AI adoption and benefiting fully from it.

Software project management is a challenging point in software project management by itself due to software development's dynamic, unique, and complex nature. Moreover, software project managers are often blamed for the failure, exceeding costs, or not being deployed on time in a software project. Consequently, the management area of software projects has faced strong challenges. The estimation of effort required for software development is one of the important and difficult challenges [45].

Project managers have always sought methods, techniques, and tools for estimating project efforts promptly before the start of the implementation to produce budgets and plans, attract potential clients, and solicit collaboration contracts with suitable teams, while misleading manipulation of budget issues has adverse effects on firms and society. The effort estimation issue is the task of predicting the amount of effort required to develop a software project. An effort estimation technique produces a prediction of the required effort for developing the software and is evaluated based on its estimation accuracy.

B. *Training and Development for Teams*

Training efforts should focus on developing processes as well as increasing the skills of the coding team. Team members involved in requirements validation, architecture, component design, and component testing can perform their designated activities and review another team member's output.

A similarity in effort between a person's own task effort and a co-worker's task effort indicates risk events for the review task. A similarity in effort between a component's design task effort and its coding task effort indicates the importance of the design document and potential errors in both related processes.

Abnormal task completion effort and ongoing task effort can indicate potential risk for a task or a team member. These detected risks should be reviewed and proceed according to the established process [1]. Available training sessions on each related skill set should be reviewed and classified according to TC, TD, and TE. Ignored training with obvious risks shown when analysing the effort of TC tasks should be scheduled as soon as possible.

When a sidetracked coding or design process is detected, support should be offered to the potentially distracted member in terms of both time and understanding. Such support may involve additional training on the skill needed, correcting the component's design, pair programming, or inspecting coding and design documents. Effort estimation tools should always be installed and used. They can help members understand the estimation process for task effort quantification. Task effort estimation is another area in need of more effort for both the training program and the processes in which members account for the final estimation project.

Implementing a method for comparing the estimated and actual effort needs attention. One possibility is to produce a secondary process to find the differences in the already available reports, which should be useful in refining the training tiering process.

VII. Conclusions

As software project managers are required to control costs and resources in increasingly data-driven environments, accurate estimation of the effort required to implement Software Development Requests (SDRs) becomes a critical decision-support activity.

Through the developed models, the use of machine learning algorithms is demonstrated as an effective solution for predicting software task duration, estimating effort, and early detection of risks in software development projects. These models can capture complex relationships between technical, planning, and human factor variables, demonstrating promising predictive capability within a controlled experimental setting.

The research provides decision support that can be integrated into a management team's workflow, regardless of the users' level of technical AI expertise. The transition from simple numerical estimation to proactive risk detection allows managers to prioritize interventions where the negative impact is greatest. Critical periods in the software project development cycle can be identified, and resources can be allocated dynamically.

The study aligns with current research trends in the field and contributes to the development of machine-learning-based models that effectively support effort estimation and risk assessment by learning complex, non-linear relationships between planning attributes, team characteristics, and actual execution outcomes.

A. *Trends and Future Research Directions*

From a project management perspective, the near-term impact of artificial intelligence is expected to be most significant in the area of data-driven planning and control rather than in fully autonomous decision making [46], [47]. AI techniques will optimize conventional management methodologies by integrating dynamic forecasting, early-warning systems, and scenario simulations, all grounded in the continuous flow of project data.

The integration of predictive analytics with established project management tools, such as backlog management systems, issue trackers, and resource planning platforms, can enable tighter feedback loops between planning and execution. Such integration supports adaptive scheduling, dynamic risk assessment, and evidence-based decision making throughout the project lifecycle.

At the same time, challenges related to data quality, model interpretability, and organizational acceptance remain central research issues. To foster trust and adoption, future studies should investigate explainable AI techniques that clarify why certain tasks are predicted to be risky or time-consuming and how specific features (e.g., complexity, dependencies, or team experience) contribute to these predictions [29], [31].

Furthermore, extending current models with temporal and organizational factors, such as workload dynamics and inter-task dependencies, is expected to enhance their ability to forecast effort and risk in complex, real-world software projects. This perspective positions machine-learning-based effort prediction and risk detection not as isolated analytical components, but as integral elements of next-generation, data-driven project management environments [40].

Consequently, future work will focus on the systematic collection of task-level datasets from real software projects, including both successful and problematic deliveries, in order to improve model generalization and robustness. In addition, integrating predictive models with project management information systems can facilitate automated data acquisition and continuous model updating, as suggested in recent research on data-driven project analytics [4], [34].

References

[1] T. N. Tran, H. T. Tran, Q. N. Nguyen, "Leveraging AI for Enhanced Software Effort Estimation: A Comprehensive Study and Framework Proposal," in Proceedings of the 2023 International Conference on the Cognitive Computing and Complex Data (ICCD), Huaian, China, 21–22 October 2023, pp. 284–289, https://doi.org/10.48550/arXiv.2402.05484.

[2] H. Zhang, M. A. Babar, P. Tell, "Identifying relevant studies in software engineering," Inform. and Soft. Technol., vol. 53, no., pp. 625–637, 2011, https://doi.org/10.1016/j.infsof.2010.12.010.

[3] A. Mohammad, B. Chirchir, "Challenges of Integrating Artificial Intelligence in Software Project Planning: A Systematic Literature Review," Digital, vol. 4, pp. 555–571, 2024, https://doi.org/10.3390/digital4030028.

[4] I. K. Kirpitsas, T. P. Pachidis, "Evolution towards hybrid software development methods and information systems audit challenges," Software, vol. 1(3), pp. 316–363, 2022, https://doi.org/10.3390/software1030015.

[5] M. E. Nenni, F. De Felice, et al., "How artificial intelligence will transform project management in the age of digitization: a systematic literature review," Manag. Rev. Q., vol. 75, pp. 1669–1716, 2025, https://doi.org/10.1007/s11301-024-00418-z.

[6] M. L. Prasetyo, R. A. Peranginangin, et al., "Artificial intelligence in open innovation project management: A systematic literature review on technologies, applications, and integration requirements," J. Open Innov. Technol. Mark. Complex., vol. 11 (1), 100445, 2025, https://doi.org/10.1016/j.joitmc.2024.100445.

[7] T. R. Benala, S. Dehuri, S. C. Satapathy, S. Madhurakshara, "Genetic Algorithm for Optimizing Functional Link Artificial Neural Network Based Software Cost Estimation," in: Satapathy, S.C., Avadhani, P.S., Abraham, A. (eds) Proceedings of the International Conference on Information Systems Design and Intelligent Applications,Visakhapatnam, India, January 2012. Advances in Intelligent and Soft Computing, vol. 132, Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-27443-5_9.

[8] M. Imani, A. Beikmohammadi, H. R. Arabnia, H.R., "Comprehensive Analysis of Random Forest and XGBoost Performance with SMOTE, ADASYN, and GNUS Under Varying Imbalance Levels," Technologies, vol. 13, pp. 88, 2025. https://doi.org/10.3390/technologies13030088.

[9] A. A. ForouzeshNejad, F. Arabikhan, S. Aheleroff, "Optimizing Project Time and Cost Prediction Using a Hybrid XGBoost and Simulated Annealing Algorithm," Machines, vol. 12, pp. 867, 2024, https://doi.org/10.3390/machines12120867.

[10] H. Kumar, V. Saxena, "Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study," J. of Soft. Eng. and Applic., vol. 17, pp. 155-171, 2024, doi: 10.4236/jsea.2024.174009.

[11] L. Lavazza, A. Locoro, G. Liu, R. Meli, "Estimating Software Functional Size via Machine Learning," ACM Trans. Softw. Eng. Methodol., vol. 32 (5), pp. 1–27, 2023. https://doi.org/10.1145/3582575.

[12] A. Jadhav, K. Shandilya, "Reliable machine learning models for estimating effective software development efforts: A comparative analysis," J. of Eng. Research, vol. 11 (4) pp. 362-376, 2023, https://doi.org/10.1016/j.jer.2023.100150.

[13] M. Jørgensen, "A review of studies on expert estimation of software development effort," J. of Syst. and Soft. Vol. 70 (1-2), pp. 37–60, 2024, https://doi.org/10.1016/S0164-1212(02)00156-5.

[14] J. Wen, S. Li, Z. Lin, Z. Y. Hu, Y., C. Huang, "Systematic literature review of machine learning based software development effort estimation models", Inform. and Soft. Technol., vol. 54 (1), pp. 41–59, 2022, https://doi.org/10.1016/j.infsof.2011.09.002.

[15] C. Capone, T. Narbaev, T., "Estimation of Risk Contingency Budget in Projects using Machine Learning", IFAC-PapersOnLine, vol. 55, pp. 3238–3243, 2022, doi:10.1016/j.ifacol.2022.10.140.

[16] K. Li, A. Zhu, P. Zhap, J. Song, J., J. Liu, "Utilizing Deep Learning to Optimize Software Development Processes," J. of Comp. Techn. and Appl. Mathem., vol. 1(1), https://doi.org/10.5281/zenodo.11084103

[17] G. D. Aregbesola, I. Asghar, S. Akbar, R. Ullah, "Fuzzy Logic Model for Informed Decision-Making in Risk Assessment During Software Design," Systems, vol. 13, pp. 825, 2025, https://doi.org/10.3390/systems13090825.

[18] K. Ferhati, A. Burlea Schiopoiu, A. G. Nascu, "A Text Based Project Risk Classification System Using Multi-Model AI: Comparing SVM, Logistic Regression, Random Forests, Naive Bayes, and XGBoost,". Systems, vol. 13, pp. 1078, 2025, https://doi.org/10.3390/systems13121078.

[19] L. Hughes, R. M. Mavi, M. Aghajani, M., K., et al., "Impact of artificial intelligence on project management (PM): Multi-expert perspectives on advancing knowledge and driving innovation toward PM2030,", J. of Innov. & Knowledge, vol.10, no. 5, pp. 100772, 2025, https://doi.org/10.1016/j.jik.2025.100772.

[20] M. H. Haghighi, M. Ashrafi, "A novel framework for risk management of software projects by integrating a new COPRAS method under cloud model and machine learning algorithms," Ann. Oper. Res., vol. 338, pp. 675–708, 2024, https://doi.org/10.1007/s10479-023-05653-3.

[21] S. S. Almalki, "AI-Driven Decision Support Systems in Agile Software Project Management: Enhancing Risk Mitigation and Resource Allocation," Systems, vol. 13, pp. 208, 2025, https://doi.org/10.3390/systems13030208.

[22] M. Zhang, M. F. Antwi-Afari, C. Wang, C., W. Sun, S. R. Mohandes, S.F. Abdulai, "Uncertainty in Software Development Projects: A Review of Causes, Types, Challenges, and Future Research Directions," Systems, vol. 13, pp. 650. https://doi.org/10.3390/systems13080650.

[23] M. L. Prasetyo, R. A. Peranginangin, N. Martinovic, M. Ichsan, H. Wicaksono, "Artificial intelligence in open innovation project management: A systematic literature review on technologies, applications, and integration requirements," J. Open Innov. Technol.

Mark. Complex., vol. 11 (1), pp. 100445, 2025, https://doi.org/10.1016/j.joitmc.2024.100445.

[24] R. Malhotra, A. Jain, “Software effort prediction using statistical and machine learning methods,” Int. J. Adv. Comput. Sci. Appl., vol. 2 (1), 2023, doi: 10.14569/IJACSA.2011.020122.

[25] A. Catana, A. Florescu, “Enhancing Risk Identification in Software Project Management through Artificial Intelligence and Machine Learning,” RECENT J., vol. 75, pp. 114-119, 2025, https://doi.org/10.31926/RECENT.2025.75.114.

[26] A. Yasmin, W. Haider, A. Daud, A. Banjar, “Machine learning based software effort estimation using development-centric features for crowdsourcing platform,” Intellig. Data Analysis., vol. 28 (1), pp. 299-329, 2023, doi:10.3233/IDA-237366.

[27] S. S. Ali, J. Ren, K. Zhang, J. Wu, C. Liu, C. “Heterogeneous Ensemble Model to Optimize Software Effort Estimation Accuracy,” IEEE Access, vol. 11, pp. 27759-27792, 2023.

[28] M.R. Mehregan, A. Rezasoltani, A. M. Khani, “A Novel Hybrid Machine Learning Model for Defect Prediction,” Ind. Manuf. Processes, pp. 43-58, 2025, 10.22080/cste.2025.29099.1037.

[29] T. Menzies, J. Greenwald, A. Frank, “Data mining static code attributes to learn defect predictors,” IEEE Trans. on Soft. Eng., vol. 33 (1), pp. 2–13, 2007, https://doi.org/10.1109/TSE.2007.256941.

[30] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” App. Soft Computing, vol. 27, pp. 504–518, 2016, doi:10.1016/j.asoc.2014.11.023.

[31] A. Tereso, C. Santos, J. Faria, “Risk Management Practices in the Purchasing System of an Automotive Company,” Systems, vol. 13, pp. 444, 2025, https://doi.org/10.3390/systems13060444.

[32] P. Pospieszny, “Software estimation: Towards prescriptive analytics,”, in Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, Gothenburg, Sweden, 25–27 October 2017, pp. 221–226.

[33] M. Choraś, R. Kozik, M. Pawlicki, W. Hołubowicz, X. Franch, “Software Development Metrics Prediction Using Time Series Methods,” in Computer Information Systems and Industrial Management, Springer International Publishing, 2019, pp. 311–323.

[34] A. Khalifeh, A. S. Al-Adwan, M. K. Alrousan, H. Yaseen, B. Mathani, F.R. Wahsheh, “Exploring the Nexus of Sustainability and Project Success: A Proposed Framework for the Software Sector,” Sustainability, vol. 15, pp. 15957, 2023, https://doi.org/10.3390/su152215957.

[35] McKinsey, “The state of AI in 2025: Agents, innovation, and transformation,” https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai (accesed on 5 November 2025).

[36] S. K. Sehra, S. Y. Brar, N. Kaur, “Soft Computing Techniques for Software Effort Estimation,” Int. J. of Adv. Computer and Mathematics Sci., vol. 2(3), pp. 160-167, 2013, https://doi.org/10.48550/arXiv.1310.5221.

[37] E. Papatheocharous, H. Papadopoulos, A. S. Andreou, “Feature subset selection for software cost modelling and estimation,” Eng. Intelligent Systems, vol. 18 (3/4), 2010, https://doi.org/10.48550/arXiv.1210.1161.

[38] K. Gao, T. M. Khoshgoftaar, H. Wang, N. Seliya, “Choosing software metrics for defect prediction: an investigation on feature selection techniques,” Software: Practice and Experience, vol. 41 (5), pp. 579–606, 201110. DOI: 10.1002/spe.1043.

[39] ISO/IEC/IEEE, “International Approved Draft Standard - Systems and Software Engineering - Life Cycle Processes - Risk Management, in ISO/IEC/IEEE P16085/ FDIS, August 2020, pp.1-60.

[40] C. Arnold, L. Biedebach, A. Küpfer, M. Neunhoeffer, “The role of hyperparameters in machine learning models and how to tune them,” Polit. Sci. Research and Methods, vol. 12 (4), pp. 841-848, doi: 10.1017/psrm.2023.61.

[41] H. D. P. Carvalho, M. C. A. Lima, M. C. A., W.B. Santos, R. A. Fagunde, “Ensemble Regression Models for Software Development Effort Estimation: A Comparative Study,” Int. J. of Soft. Eng. & Applic., vol. 11, n0. 03, pp. 71–86, 2020, https://doi.org/10.5121/ijsea.2020.11305.

[42] X. F. Liu, G. Kane, M. Bambroo, “An intelligent early warning system for software quality improvement and project management,” J.. Syst. Softw., vol. 79, pp. 1552–1564, doi:10.1016/j.jss.2006.01.024.

[43] M. Shepperd, C. Schofield, “Estimating software project effort using analogies,” IEEE Trans. Softw. Eng., vol. 23, pp. 736–743, 1997, doi:10.1109/32.637387.

[44] H. K. Dam, T. Tran, J. Grundy, A. Ghose, Y. Kamei, “Towards effective AI-powered agile project management,” in Proceedings of International Conference on Software Engineering, May 2019, Montreal, Canada, https://doi.org/10.48550/arXiv.1812.10578.

[45] A. Moradbeiky, A. Khatibi Bardsiri, “A new architecture based on artificial neural network and PSO algorithm for estimating software development effort,” J. of Telecomm. Electr. and Comp. Eng., vol. 9., pp. 13-17, 2019.

[46] K.K. Reddy, H.S. Behera, “Software effort estimation using particle swarm optimization: Advances and challenges,” in Computational Intelligence in Pattern Recognition, A. Das, J. Nayak, B. Naik, S. Dutta, and D. Pelusi, Eds. Singapore: Springer, 2020, vol. 1120, doi: 10.1007/978-981-15-2449-3_20.

[47] B. Soongpol, P. Netinant, M. Rukhiran, “Practical Sustainable Software Development in Architectural Flexibility for Energy Efficiency Using the Extended Agile Framework,” Sustainability, vol. 16, pp. 5738, 2024, https://doi.org/10.3390/su16135738.