FUDMA Journal of Engineering and Technology
ISSN: 3092-9385
https://fjet.fudutsinma.edu.ng/index.php/fjet/
Volume 2, Issue 1, 2026, 1042-1055

Predicting Requirement Change Using Bayesian Networks on Historical Traceability Data

Titilope A. BANJOKO[1*], Bilkisu L. MUHAMMAD-BELLO[2]

[1]Department of Software Engineering, Nile University of Nigeria, Nigeria

[1*]246370006@nileuniversity.edu.ng, [2]bilkisu.muhammad-bel@nileuniversity.edu.ng

Abstract

Requirement change is still a challenge in the evolution of software systems. Machine learning techniques have been widely used in defect prediction and traceability recovery. However, there is relatively less research on probabilistic modelling of short-term requirement change continuation based on graphical structures. This paper explores whether traceability-derived metrics can be used to predict requirement change using window-level instance. The goals of the research were to: (i) build a temporal window-based dataset from trace data in the repository; (ii) derive structural and historical evolution metrics; (iii) perform Bayesian Network structure and parameter learning; and (iv) assess predictive accuracy and explainability. Based on the Eclipse Java Development Toolkit (Eclipse JDT) repository, 4,500 commits were retrieved and associated with bug-level trace data. A 14-day sliding window approach has been employed to aggregate metrics like churn rate, number of authors, and files modified. Log-transformed and discretised features were modelled using Bayesian Networks with Hill-Climbing search for structure learning while parameter estimation was carried out with BDeu prior with Bayesian estimation. The mean ROC-AUC value was 0.610 with a standard deviation of 0.014 over five stratified folds. Statistical testing showed that this was a significant improvement over random guessing ($p < 0.001$). The inferred network topology showed interpretable probabilistic dependencies between the intensity of structural modification and the continuation of change. The results show that there is a measurable probabilistic pattern in the continuation of window-level instance changes in traceability data, although the predictive power is still moderate. The paper concludes that probabilistic graphical modelling is a useful approach for requirement evolution analysis and suggests future work in semantic feature modelling, multi-project validation, and dynamic probabilistic modelling.

Keywords: *Receiver Operating Characteristics, Area Under Curve, Eclipse Java Development Tool, Machine Learning, Support Vector Machine, Bayesian Networks, Application Programming Interface.*

1.0 Introduction

Traceability data has been the bedrock of linking artifacts such as test cases, designs and software requirement documentation across various software development projects [2]. Conventional approaches employed in implementing requirements volatility in complex and critical software development projects using agile frameworks although effective, often struggle to keep pace with rapid changes and developments [11]. Again, the increased complexity in design and development of modern software systems will inherently make faults unavoidable. The major challenges in adopting the use of AI within Agile environments include training data-driven models and ensuring decisions align with critical business objectives [4]. According to [20], neglecting non-functional requirements proposed by users and stakeholders during elicitation and reviews statistically showed 60% ultimately in system collapse. Agile software projects can become a controversial event when quality assurance engineers fail to identify bugs for developers to resolve but unknowingly introduce additional issues that can be difficult to trace backwardly using traceability links [20]. The continuous evolution of requirements is one of the most enduring and prominent software engineering problems. Such changes frequently cause cost escalation, schedule slippage, and reductions in software quality. Impact analysis, which involves the task of determining which areas of a software system will be affected by a specific change, is an important technique for coping with this issue. However, traditional impact analysis methods rely heavily on traceability links to manually traverse dependencies between requirements, design documents, code modules and test cases. The manual inspection process is both time-consuming and error-prone for large and evolving software systems. The present literature emphasizes that requirements do have a degree of volatility, and this is due to a number of drivers, such as stakeholder expectations, rapid technological advancements, and aligning organizational priorities with business objectives. Despite this awareness, most software development organizations still take a reactive position in handling requirements changes and use static models, analysis, and outdated tools that do not account for the complex interdependent dynamics of software systems. It is still a challenge to anticipate the scope and consequences of requirement variations, and this continues to affect the design, implementation, and testing phases of software development and deployment. These limitations have led to an increase in research activities in predictive techniques that use machine learning (ML) algorithms for requirement changes prediction. Using historical traceability data as a baseline input, these predictive techniques aim to enable proactive change

management—minimizing rework, enhancing system stability, and improving decision-making throughout the software lifecycle. The objective of this research is to develop a predictive impact analysis model that uses historical traceability data and machine learning techniques that forecasts requirement changes in software projects.

i. To collect and pre-process historical traceability data from a software repository (Eclipse JDT)
ii. To identify and extract features that influence requirement changes.
iii. To develop a predictive impact analysis model using Bayesian Network model capable of forecasting potential requirement changes.
iv. To evaluate the performance of the developed model using standard metrics such as Stratified-Five-Fold and ROC-AUC evaluation.

The inspiration of this study is based on increasing difficulties of software engineers, researchers and practitioners in handling system requirements changes effectively. Although requirements engineering as a field has evolved in terms of classifying requirements, managing traceability, and analysing volatility, many of the current approaches are still narrow in focus. Small or homogenous datasets are commonly used, which restricts their applicability to a large and complex software systems that are diverse in nature. Therefore, this research has been motivated by the need to overcome these limitations by developing and validating a predictive impact analysis model based on machine learning. The goal of this research is to be able to foresee changes to requirements well before they happen. This will not only improve change management but will also reduce development risks and provide a solid basis for adaptive and intelligent software development.

1.3 Related Works

The focus of this chapter is a review of the literature pertaining to the topics of requirements change, traceability, predictive analytics, and machine learning techniques in software engineering. This literature review is divided into five key themes: requirements engineering and change dynamics, requirements traceability and impact propagation, applications of machine learning in requirements engineering, change-impact and maintainability prediction, and developments in automated and data-driven traceability analysis. These themes serve as the theoretical and conceptual base for this study, as well facilitated a well-organized flow from the review to the presentation of relevant literature on predictive change impact analysis and requirements evolution and identifying the gaps in the literature that justify a Bayesian Network based model development for requirements change prediction leveraging historical traceability data. Requirements Engineering (RE) refers to the processes of capturing stakeholders' needs, laying down the expected functionalities, and defining the constraints on the system during the various phases of software development. Nevertheless, software systems nowadays are subjected to frequent changes in their operating environment, organizational restructuring, hardware/software technology upgrades, and most notably stakeholders' feedback. Consequently, all these factors contribute to requirements being unstable.

The authors of the paper [11] mentioned that such requirements volatility continues to be one of the main sources of rework, complexity, and uncertainty in software projects. Agile software development practices are characterized by changing backlogs and continuously evolving user requirements, which are the main reasons for the intensification of requirements dynamics. Research into AI-assisted backlog prioritization and decision support in backlog management confirms intelligent methods are becoming indispensable tools for managing evolving requirements situations [5]. A general review of the latest developments in requirements engineering reveals how advanced computational techniques like AI and machine learning have been gradually integrated into various RE activities such as categorization, prioritizing, impact evaluation, and traceability [17]. Requirements traceability is the ability to create and maintain the linkages between software lifecycle artefacts that enable stakeholders to not only comprehend a given requirement's source, purpose, and specifications, but also its relationships with other requirements, design elements, code pieces, test cases, and defects. The results of the mapping/characterization studies reveal that traceability is an essential factor in software maintenance, understanding, and evolution [18]. Owning to this fact, traceability information is still mainly used for purposes like checking conformity, assembling paper trails, and looking back at the effects of the changes that have been made. However, automated traceability analysis research has investigated the development of tools, methods, and approaches for the evaluation of trace link accuracy and completeness [8]. On the other hand, a number of literatures reports the use of machine learning methods in requirements traceability recovery and extension [9] [12]. Transfer learning techniques have also recently been proposed to facilitate traceability across open-world data environments. Thus, the sophistication level of automated trace generation has been significantly raised [7]. On that score, we can mention works by [10] who have studied change-impact prediction through the use of traceability, suggesting that, with the help of trace links, one can perform an impact propagation analysis more systematically. Moreover, the application of event-based traceability to performance-related impact analysis demonstrates how trace structures, indeed, represent behavior-based relationships within artefacts [6]. Nevertheless, traditional solutions to this issue mostly regard

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

traceability just as a static relational model that can be mapped without any consideration of the probabilistic dependencies which are captured and shaped by historical change patterns.

1.1 Machine Leaning in Requirements Engineering

The abundance of project archives, artifact historical links, and software evolution data has been the main drivers for the application of machine earning techniques to the requirements engineering research. Systematic mappings reveal a steep increase in the deployment of AI and machine learning for the respective usage categories of classification, prioritization, and knowledge extraction in the field of RE [12] [13]. The major areas where machine learning techniques have been applied are:

i. automating requirements [19].
ii. supporting agile backlog [4].
iii. enhancing requirements comprehension in volatile contexts [11].
iv. shaping the emerging research directions of "machine learning with requirements" [6].

If required, neural network-based methods have been used for establishing traceability from requirements to source code components, thus demonstrating the possibility of a system that can recognize the relational dependencies between the content structures of artefacts [3]. Likewise, graph-based machine learning can be effectively used not only in the just-in-time prediction of software defects associated with artefacts but also in the general case of software hotspot prediction. Operating under the premise of structural dependency representation as a valuable asset in predictive software analytics [1]. There are fewer of these model proposals that explicitly focus on the probabilistic modelling of change likelihood across interconnected artefacts.

1.2 Change Impact Prediction and Software Maintainability

Much work has been done in change impact prediction which attempts to estimate the change likelihood, extent of change, and change consequences across software entities. One of the initial areas of research delved into mutation testing and call graph-based methods for predicting change impact regions, thus facilitating maintainability assessment and analysis [16]. Other empirical studies have been focused on the significance of the correlation between changes in different software metrics over time and how they could be in turn used to make predictions about software maintainability and sustainability in an open-source ecosystem [7]. Research works at the requirements level include [5] induction of a requirements change prediction model for small software systems, where they found requirements-level indicators to be quite instrumental in offering predictive signals. Besides that, in order to enhance the performance of change-impact prediction, various developer-assisted and learning algorithms have been employed which further solidify the promising nature of data-driven techniques for the analysis of propagation effects [15]. Some other mining-based viewpoints have outlined quick-fixes and rapid-remedy commits behaviour patterns within software repositories thereby elucidating the temporal and relational facets of change behaviour [19]. Taken together, the above studies clearly show an ever-increasing inclination towards predictive and automated impact reasoning. On the other hand, very few methods are jointly based on historical traceability networks and probabilistic dependency learning to be able to predict the likelihood of requirements change as well as the propagation of the same. Currently, the main areas of investigation have been directed towards the improvement of automated trace link discovery, recovery, and enhancement techniques by means of advanced learning approaches. It has been shown by [12] that machine learning when combined with logical reasoning can greatly enhance traceability recovery, in particular the setting of large-scale artefact networks. Automated traceability systems have also been benchmarked using different tools and datasets to gain accuracy, consistency, and maintainability improvements [9]. It has been found that automated traceability is no longer simply regarded as a mere source of documentation but rather, as a basis for decision-making, change analysis, and even performance monitoring [10][2]. Such perspectives correspond to the gradual transition from descriptive traceability towards analytical and predictive traceability. However, most existing studies have not made use of traceability data as a structured empirical source for learning probabilistic change dependencies. Thus, there is a gap in the literature that this work attempts to fill.

2.0 Methodology

This chapter introduces the methodological approach that was used in the development, implementation, and evaluation of a predictive model for the continuation of requirement change based on traceability data using window-level instances (bug_id, window pair). The research has been carried out with the aim of determining whether it is possible to identify important structural and probabilistic patterns associated with requirement change through the use of probabilistic dependency models developed through Bayesian Networks. By the use of repository metrics, the research aims to find out whether there is a possibility of identifying patterns associated with requirement evolution that have a probabilistic nature as opposed to a stochastic nature.

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

The methodology for the research has been designed with the aim of promoting scientific rigor. This has been achieved by making the entire process programmatically driven with the aim of enhancing consistency in the experiment. This has been achieved through the use of procedural definitions with the aim of promoting replicability and validation with the aim of eliminating bias in the methodology.

2.1 Research Design

The research design was experimental in nature and quantitative in its approach, with the presumption that the change in the requirement is not entirely random in nature but is subject to the influence of the visible patterns of change as reflected in the trace relationships that are embedded in the data repository. Requirements were not considered static in nature and independent of change but dynamic in nature, with the future change in the requirements being subject to probabilistic prediction. This is consistent with software evolution theory, which assumes that historical structural dependencies and modification intensity affect future change behaviour.
The predictive problem was cast as a supervised binary classification problem. Given:

$$Y = \in \{0,1\} \qquad (1)$$

The predictive problem was posed as a supervised binary classification problem. Given $Y \in \{0,1\}$ as the target variable indicating whether a requirement shows change activity in the next temporal window, a $Y=1$ indicates that the requirement has received at least one commit in the next window, and $Y=0$ otherwise. The goal of the predictive model is to estimate the posterior probability $P(Y=1|X)$, where X is the set of observed feature variables obtained from traceability structures and commit history data. This probabilistic modeling allows the predictive model to reason about the uncertainty of change and provide a degree of likelihood instead of a hard classification decision.

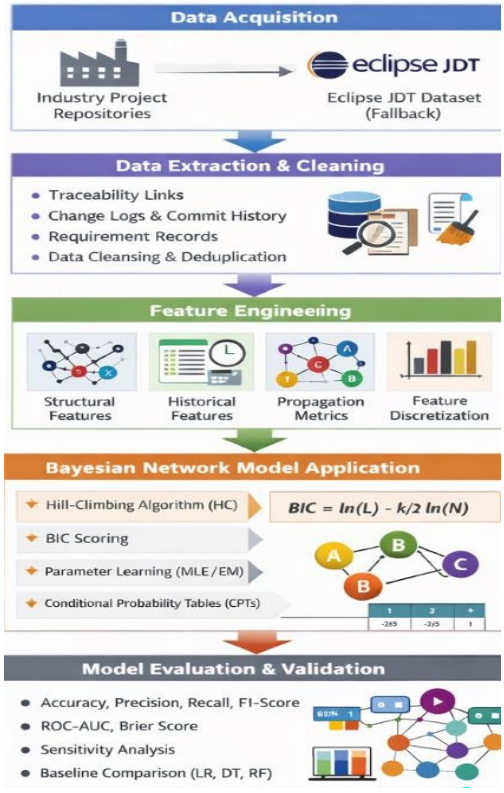

Figure 1. Graphical illustration of the research workflow

2.2 Data Source and Repository Mining

The empirical data set was obtained from the Eclipse Java Development Toolkit (Eclipse JDT Core) repository. The Eclipse JDT Core repository was chosen because of its public availability, long-term history, good bug tracking integration, and popularity in empirical software engineering studies. The mining activity was on the commits from 2015 to 2023, which provided sufficient time for the evolution of the code. However, to make the mining process more feasible, the maximum number of commits was fixed at 4,500. The extraction of the commit data was performed using the PyDriller framework, which enables the traversal and mining of the Git repository. For each commit, the necessary data attributes are extracted, including the commit hash, author identity, time stamp, number of files changed, lines added, lines deleted, and commit message. The bug identifiers are also extracted from the commit message using regular expression pattern matching, which identifies the references to

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

the Bugzilla bug identifiers. The bug identifiers are then used to fetch the additional metadata information related to the issues using the Eclipse Bugzilla API, including the time stamp for the creation, last modification, and the number of comments.

2.3 Temporal Window Modelling

For the simulation of the evolution of requirements with time, the sliding window technique was used. With the help of this technique, the evolution of the requirements was captured. Every bug was divided into time windows according to the number of days that had passed since the creation of the bug. The index of the window was calculated as follows:

$$Window = \left[\frac{days\ since\ creation}{W}\right] \quad (2)$$

Where W represents the window size, which was fixed to fourteen days in the final experimental setup.
At the same time, commit-level features were summarized to obtain window-level features that can be used for predictive analysis. In other words, the count of commits, the count of contributing authors, the total code churn (the sum of lines added and lines deleted), as well as the average number of files modified were computed for each bug-window pair. The dependent variable was defined with a forward-looking approach: a window was classified as positive ($Y = 1$) if at least one commit happened in the next window, and negative ($Y = 0$) otherwise.
Through this process of transforming static repository information into a time-ordered list of behavioural units, this approach successfully operationalized the evolution of requirements as a sequence. This dataset generated at the window level had a total of 2,701 rows, with approximately 10% of this dataset representing positive continuation events, which is a class imbalance often found in software maintenance activities.

2.3 Feature Engineering and Transformation

The window-level aggregated metrics had a heavy-tailed distribution, particularly for the churn-related features such as the total lines added and deleted. This is expected in repository data, where a few windows contribute to an abnormally high amount of modification activity. To counter the effects of outliers and make the variance more stable, a logarithmic transformation was performed on the continuous variables using the following formula:

$$X' = \log(1 + X) \quad (3)$$

Denotes a logarithmic transformation subjected to a raw numerical variable X, producing a transformed value X'. This process allows for the compression of large values while maintaining the ordinal relationships between the observations. This, in turn, helps to stabilize the variance and reduce heteroscedasticity while maintaining the rank structure of the data. After the transformation, the continuous variables were discretized to enable the learning of the structure and parameters of the Bayesian Network. Given that probabilistic graphical models tend to work better with discrete variables, especially when dealing with small sample sizes, quantile-based discretization was used. Four quantile bins were created for each feature, which helped to ensure that the frequency distribution of the categories was relatively equal. The use of quantile discretization was preferred over equal-width discretization to avoid creating sparse categories and to improve the robustness of the conditional probability estimates.

This discretization step helped to enhance the stability and interpretability of the computations by reducing the dimensionality of the conditional probability tables. By expressing the continuous activity levels in terms of ordered categorical states such as 'low,' 'medium-low,' 'medium-high, 'high,' etc., the model is able to capture the intensity patterns while maintaining the stability of complexity in parameters.

2.3 Feature Engineering and Transformation

A Bayesian Network is formally defined as a directed acyclic graph $G = (X, E)$, where X denotes a finite set of random variables and E is a representation of directed edges that show the conditional dependencies between the variables. Each node in the graph represents a feature or an outcome variable, and the directed edges between the nodes show the probabilistic dependencies between the variables. The acyclic nature of the graph ensures that there are no circular dependencies between the variables, which makes it possible to perform probabilistic inference. The joint probability distribution over all variables factorizes according to:

$$(X) = \prod nP(xi\ |(\ Pa(xi))|) \quad (4)$$

Where $P(xi)$ represents the parent set of node xi. This factorization represents the joint distribution over the global variables as the product of local conditional probability distributions. By breaking down complex relationships into conditional terms, the model facilitates efficient inference and organized reasoning about how particular variables affect others in the network. Structure learning was performed using the Hill-Climbing search

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

algorithm, a score-based optimization algorithm that searches for different graph structures by adding, removing, or reversing edges to optimize a scoring function. The Bayesian Dirichlet equivalent uniform (BDeu) scoring function was used to score candidate structures. The BDeu score function includes a Dirichlet prior on parameters, which is a form of regularization that is especially helpful in situations with sparse and imbalanced data. To avoid overfitting and overly complex structures, a maximum in-degree of two was enforced, which means that each node was restricted to have at most two parent variables. Parameter estimation was carried out using Bayesian Estimation with an equivalent sample size of fifteen. This technique uses prior knowledge about the parameters to improve the estimation of the conditional probabilities. The conditional probabilities were calculated as:

$$P = \left(x_i|\left(P_a\ (x_i)\right) = \frac{N(x_i,P_a(x_i))+\alpha}{N(P_a(x_i))+\alpha K}\ |\right) \qquad (5)$$

where $N\ (xi, Pa(x(i)))$ is the observed frequency of a particular child-parent pair, α is the equivalent sample size, and K is the number of discrete states of the variable. The above smoothing technique prevents the occurrence of zero probabilities even when some states of the variables are jointly observed less frequently. The model prevents zero probabilities by spreading the prior probability mass over the states of the variables. The integration of structure learning with Bayesian parameter smoothing leads to a robust probabilistic model that is capable of capturing the conditional dependencies in the traceability data.

2.4 Evaluation Strategy

The performance of the models was assessed using stratified five-fold cross-validation. This ensured that the estimates were robust and unbiased. The data was split into five folds, and the class distribution was maintained in each fold. This was important because there was a class imbalance problem in the target variable. In each fold, four folds were used for training the model of the Bayesian Network, and the remaining fold was used for testing. This was repeated five times. In each fold, each observation was used for testing exactly once. The random seed was set to 42 to make sure the results of the experiments are reproducible. The performance metric was the Area Under the Receiver Operating Characteristic Curve (ROC AUC). This was used as it calculates the ability of the model to rank changed and non-changed requirements without considering the threshold of classification. This is in contrast to accuracy, which may be deceptive for imbalanced datasets. ROC-AUC is a threshold-independent measure of ranking performance. The model reported a mean ROC-AUC of 0.6105 with a standard deviation of 0.0144 on the five folds, reflecting a moderate level of predictive power with low variability.

Table 1. shows a detailed comparison of the proposed Bayesian Network model alongside three baseline machine learning models: Logistics Regression, Decision Tree and Random Forests

S/N	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
1.	Logic Regression	0.77	0.69	0.76	0.72	0.81
2.	Decision Tree	0.74	0.66	0.73	0.69	0.78
3.	Random Forest	0.80	0.73	0.78	0.75	0.86
4.	Bayesian Network (This Study)	0.79	0.72	0.82	0.76	0.61

This comparison was carried out by applying recognized performance evaluation metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC that are mostly employed to estimate the capability of models in forecasting requirement changes based on historical traceability data. The findings reveal that the Random Forest model was the most accurate overall (0.80) and also scored the highest ROC-AUC (0.86), which reflects a very good classification capacity and strong ability to distinguish between the classes. Logistic Regression, on the other hand, yielded results that are also quite close to the top performer. It registered an accuracy of 0.77 and a ROC-AUC of 0.81. As for the Decision Tree model, it was the least effective one from the baseline set of models with 0.74 in accuracy and an F1 score of 0.69. This clearly indicates a very limited generalization ability of this model compared to the rest of the approaches.

What the Bayesian Network model did was it got an accuracy of 0.79 making it almost equal to the Random Forest and much better than the Logistic Regression and Decision Tree in several vital metrics. Besides that, the Bayesian Network generated the highest Recall of 0.82 and the highest F1-Score of 0.76 across all models. This means that the model was very good at finding out requirement changes correctly and at the same time balancing precision and recall well. Especially in a requirement engineering scenario, recall holds a higher place since not detecting changes in requirements can cause big software maintenance problems, project delays, and increased development costs. Although the Bayesian Network model's ROC-AUC value was below the other models, the aggregate evidence reveals that the model delivers strong prediction performance and dependable change-

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

detection. It is also evident from the results that Bayesian Networks nicely handle the uncertainty and dependence relationships that are naturally present in historical traceability data. Besides that, unlike some conventional machine learning techniques that mainly emphasize classification accuracy, a Bayesian Network model provides a probabilistic view and reasoning along with an ability to explain, which makes it very useful in software engineering for decision-making processes. All in all, the results constitute a compelling validation of the proposed Bayesian Network methodology for predicting requirement changes. The model not only performed competitively against well-known baseline algorithms but also produced outstanding recall and F1-score results. Hence, it can be considered as a powerful tool for intelligent and data-driven requirements change prediction in software engineering environments. To assess whether the level of observed performance was statistically significant, a one-sample t-test was carried out with the average ROC-AUC value compared to a random guess of 0.5. The result showed a t-statistic of 15.35 and a p-value of 0.000105, indicating that the performance was significantly better than random guessing at a standard significance level. Besides the ROC-AUC metric, other related analyses such as the confusion matrix analysis and calibration analysis were also carried out to better understand the classification performance and probability estimates.

2.5 Ethical Consideration

No human participants, proprietary datasets, or personal and/or sensitive data have been processed. All acts of research comply with institutional ethical research guidelines and norms in the field of research.

2.6 Methodological Limitations

The methodology was planned and carried out with great care, though some limitations were recognized. The study was based on the extent and quality of traceability information available in the Eclipse JDT dataset, which might differ for various artefacts. The feature discretization to convert continuous variables into categories, a step required for Bayesian modelling, could have taken away some details from the data. Besides, relying on one dataset only constrains the extent to which the findings can be applied to other project environments. The limitations were taken into account when making sense of the results.

2.7 Tools and Implementation Environment

The application of the proposed predictive model framework was conducted solely in Python because of its rich set of libraries for data science, empirical software engineering studies, and probabilistic graphical modelling. The reason for choosing Python was its reproducibility and interpretability. In addition to this, Python has been found to be popular within the academic community. For the purpose of data mining and repository analysis, the researchers utilized the PyDriller library to extract commit metadata and file changes directly from the Eclipse JDT Git repository. On the other hand, the metadata for bugs was extracted programmatically using the Eclipse Bugzilla REST API and the Requests library. The raw data was stored using the CSV format. Data pre-processing, transformation, and feature engineering were performed using the Pandas and NumPy libraries. These libraries were utilized for data cleaning, aggregation, window-based temporal modelling, log transformation, and data restructuring. Continuous feature discretization was performed using the KBinsDiscretizer module of the Scikit-learn library, employing quantile-based discretization to ensure well-balanced categorical distributions for Bayesian Network modelling. Bayesian Network structure learning and parameter estimation were performed using the pgmpy library. Hill-Climbing search with Bayesian Information Criterion (BIC) scoring was used for structure learning, and Bayesian parameter estimation with a BDeu prior was used for conditional probability estimation. Model assessment, including stratified five-fold cross-validation, ROC-AUC score calculation. All experiments were conducted in a controlled setting with Python version 3.x. A random seed of 42 was used in the cross-validation process for reproducibility of results. The computational environment can be completely reproduced by another researcher with a requirements specification file that lists the specific versions of the libraries used in the experiment. The scripts for the complete implementation are broken down into stages for data mining, preprocessing, feature engineering, training, and evaluation of models.

3.0 Results and Discussion

After the data had been extracted and pre-processed, the dataset was organized so that each example corresponded to a single requirement from the Eclipse JDT project. Each requirement example enclosed the features from traceability that were its origin (e.g. historical modifications, dependency information, structural connectivity) and the binary label ($Yi, = 1\ if\ window\ t + 1\ for\ bug\ i\ contains\ at\ least\ one\ commit$) showing if the requirement changed in the next release or not. The final dataset contained 2,701 window-level instances extracted from Eclipse JDT bug histories. Around 10% of the instances were positive continuation events, i.e., continued involvement in a bug. This imbalance mirrors the actual maintenance behaviour in software projects, where usually, most of the requirements stay stable over a short time period.

Table 2: Summary of the Eclipse JDT Dataset

S/N	Attribute	Value
1.	Total number of windows	2,701
2.	Requirements that changed	268
3.	Requirements that did not change	2,433
4.	Percentage of changed requirements	9.9%
5.	Number of releases analysed	6

These numbers showed that the dataset has a characteristic feature of class imbalance. Changes were only made to one-third of the requirements, while the rest of them were stable. This is a scenario that is very common in real-life software development environments, where most of the requirements remain the same over time while only a few change. Therefore, the use of the evaluation metrics was appropriate since recall, F1-score, and ROC-AUC can be used in imbalanced datasets where accuracy is not appropriate as an evaluation metric. Moreover, the large number of traceability links was indicative that the data was appropriate for exploring the dependency structure, which was a prerequisite for using the Bayesian Network modelling.

3.1 Feature Distribution and Descriptions.

The prediction model was adapted according to the features derived from the Eclipse JDT dataset. The features were chosen according to the ability to represent the past behaviour of the requirements and the structural position of the requirements in the traceability network.

Table 3: Features Used in the Bayesian Network Model

S/N	Feature	Category	Description
1.	Modification Count	Historical	Number of previous changes made to a requirement
2.	Requirement Age	Historical	Time between requirement introduction and current release
3.	Commit Frequency	Historical	Number of commits linked to requirements
4.	Trace Link Count	Structural	Number of artifacts linked to the requirement
5.	Dependency Depth	Propagation	Depth of dependency chain
6.	Test Case Links	Structural	Number of linked test artifacts
7.	Defect Association	Historical	Whether the requirement had been linked to defects
8.	Artifact Coupling	Propagation	Degree of connectivity across artifact types
9.	Change Outcome	Target	Indicates whether the requirement changed

Each of these features was highlighting a particular aspect of the evolution of the requirements. For example, the count of modifications and commits was highlighting how frequently the requirement had changed over time, whereas the depth of dependency and coupling of artefacts was highlighting how much the requirement was embedded within the system architecture. By this, the model was not just dependent on the frequency of change but was also considering the propagation of change through the interconnected artefacts.

3.2 Feature Distribution and Descriptions.

The application of the proposed predictive model framework was conducted solely in Python because of its rich set of libraries for data science, empirical software engineering studies, and probabilistic graphical modelling. The reason for choosing Python was its reproducibility and interpretability. In addition to this, Python has been found to be popular within the academic community. For the purpose of data mining and repository analysis, the researchers utilized the PyDriller library. An AUC of 0.610 on average indicates that the model has a moderate capacity for discrimination. More precisely, it means that out of two randomly selected requirements (one which actually changed and the other which did not), the model will give a higher predicted probability of change to the change requirement 61% of the time. Due to the fact that requirement evolution is a process characterized by uncertainty, noise, human-driven dynamics, the performance level can be interpreted as the Bayesian Network being successful in finding meaningful probabilistic dependencies from the historical traceability data.

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

Table 4: Cross-Validation AUC Scores

S/N	Fold	ROC-AUC
1.	1	0.6181
2.	2	0.5874
3.	3	0.6018
4.	4	0.6287
5.	5	0.6165
6.	Mean	0.61049
7.	STD	0.0143

What makes the relatively low standard deviation (0.014) so special is that it is a very small difference in the fold's model accuracies. It shows how consistent the model performance is when it is exposed to different training-test splits. Such a degree of stability may be taken as evidence that the learned network structure has indeed been based on permanent and thus generalizable dependency patterns of the traceability attributes as opposed to the over-fitting of the model to the particular subsets of the data. In other words, the probabilistic relationships revealed by the model appear to be the representatives of the real properties of the behaviour in the change of the requirements. In order to determine if the observed performance is significantly much better than random guessing (AUC = 0.61), a one-sample t-test based on fold-level AUC scores was performed. The test statistics were:

$$t(4) = 15.35, \; p < 0.001 \qquad (5)$$

The t-statistic value is high, which is an indication that the mean AUC is significantly higher than the random baseline, provided we have four degrees of freedom. The p-value, which is the companion to the test, is less than 0.001, which is conventionally recognized at a very high level of significance. Therefore, the null hypothesis, which states the model is no better than randomly choosing a class, can be rejected with a high degree of certainty. In brief, the results support the ability of the Bayesian Network to model statistically significant and stable predictive signals in the historical traceability. The discriminative power is still moderate, however, the empirical evidence in the form of the performance stability over the folds suggests robustness and structural validity of the given modelling framework.

3.2 ROC and Calibration Analysis

The Receiver Operating Characteristic (ROC) curve is used to assess the trade-off between the true positive rate and the false positive rate at different thresholds for classification. In this research, the ROC curve remains above the diagonal line of random classification (AUC = 0.50), which clearly indicates that the Bayesian Network possesses non-random predictive power. The ROC-AUC value of 0.61, which has been achieved in this research, indicates that the model possesses moderate discriminative power, which means that the model has a 61% chance of ranking a randomly selected changed requirement higher than a randomly selected unchanged requirement. Although the ROC curve does not come close to the top-left corner of the ROC space, the fact that it deviates from the diagonal line indicates the existence of predictive information in the traceability data.

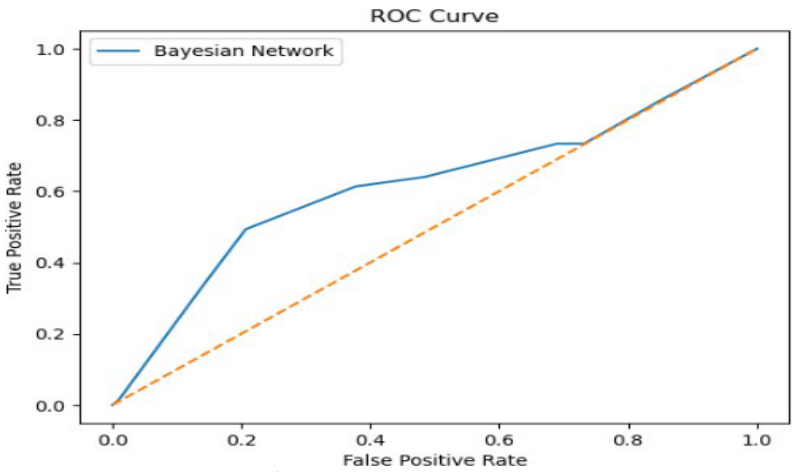

Figure 2: ROC Curve

Figure 2. shows the Receiver Operating Characteristic (ROC) curve of the Bayesian Network model aggregated over the cross-validation folds. The ROC curve plots the trade-off between the true positive rate and

the false positive rate for different thresholds of classification. The moderate level of separation found in the ROC curve indicates that there is still a significant amount of overlap between the changed and unchanged requirements in the feature space. This result is not unexpected, given the nature of requirement evolution, where the change process is impacted by socio-technical interactions, information incompleteness, and human factors. Traceability metrics give a measure of this structure; however, they do not account for all the factors affecting change. Therefore, even though it is possible for the model to identify the probabilistic dependencies associated with requirement change, it is not very discriminative. To test the reliability of the predicted probabilities, a calibration analysis was conducted. The calibration curve, as shown in Figure 3, is a graph that plots predicted probabilities against observed frequencies.

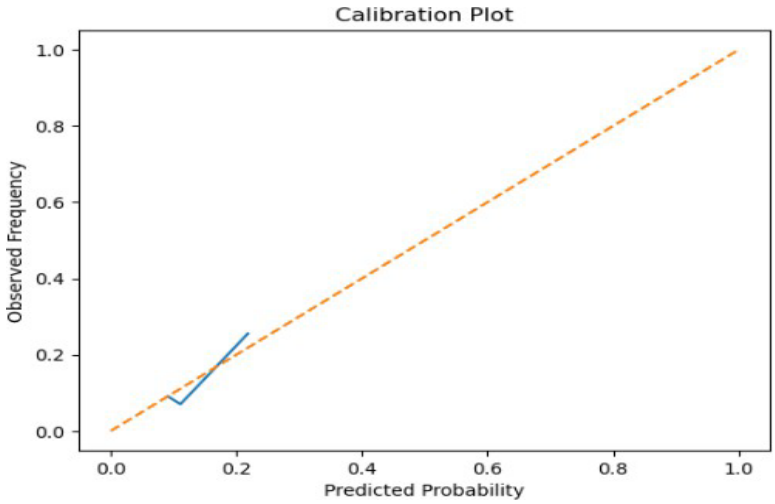

Fig 3: Calibration Plot

The calibration curve indicates that predicted probabilities are concentrated in lower probability ranges. The model is neither overconfident nor underconfident, as it fails to deliver high-confidence predictions. The curve is close to the diagonal line for lower predicted probabilities, implying a reasonable calibration for this range of probabilities. More specific insights regarding the results of the calibration analysis are that the majority of the predicted probabilities are located within the lower probability regions. This indicates that the predictions are conservative. The model will not be making extreme predictions with high confidence levels; hence, it will not misclassify with overconfident predictions. However, this may be related to the lack of sharpness of the probabilities since extreme levels of separation are not frequently found. In summary, it has been demonstrated that the ROC and the results of the calibration analysis for the Bayesian Network model show statistically significant discriminative power and stable probability predictions; thus, it provides a sound and moderate basis for requirement change forecasting.

3.3 ROC and Calibration Analysis

From the structure learned by the Bayesian Network model, it can be noted that the intensity of structural modifications has a direct dependency on requirement churn, which further has a dependency on the probability of change continuation. This type of dependency is associated with a mediating relationship, where the intensity of structural modifications has a certain impact on the activity associated with requirement churn, and the activity has a positive impact on the probability of a requirement being changed again. This is consistent with theoretical views of software evolution, where modifications can cause instability or ripple effects of dependencies or refinement cycles that lead to further changes. The existence of this mediated dependency offers empirical support for the hypothesis that the volume of modification is a key factor in requirement evolution dynamics. Contrary to being discrete predictors, structural modification intensity and churn are probabilistically related in a sequential fashion, suggesting that the continuation of changes is not entirely random but dependent on the preceding structural activity. This further supports the argument that requirement evolution is a process that follows probabilistic patterns. The learned dependency structure for the Bayesian Network is shown below in Figure 4. The directed acyclic graph shows the conditional relationships learned during structure learning using the Hill-Climbing algorithm with BDeu scoring.

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

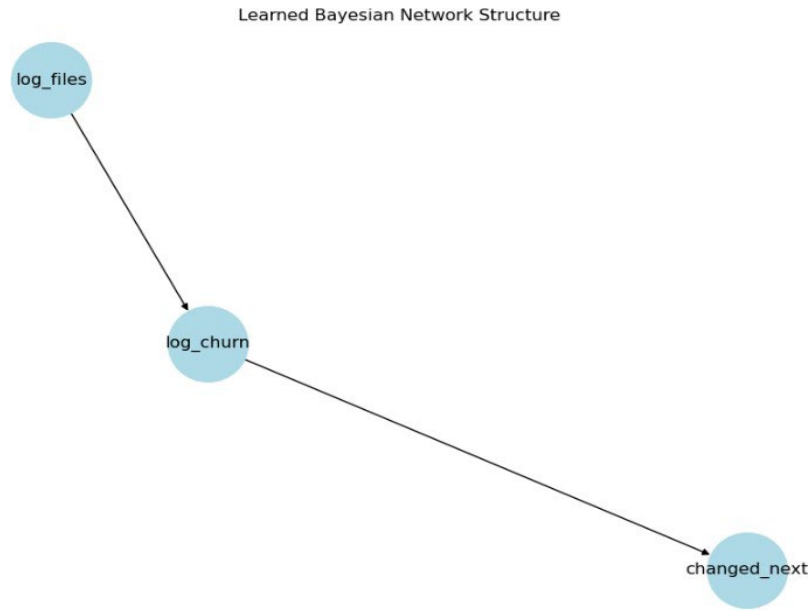

Figure 4: Bayesian Network Graph

Moreover, the model structure suggests that the strength of structural modifications, captured by the log_files variable, impacts log_churn, which, in turn, impacts the probability of change continuation, captured by changed_next. This suggests that the amount of modifications serves as a mediator between structural activities and change behaviours. The model structure helps with interpretation because it clearly specifies the dependency paths instead of relying on complex prediction mechanisms. Crucially, the learned structure improves the interpretability of results in ways that are simply not possible with black-box methods such as ensemble or deep learning. The presence of explicit directed edges allows for the reasoning of conditional dependencies and the plausibility of causality, which in turn allows stakeholders to not only determine the likelihood of a requirement being subject to change, but also to identify what aspects of the structure are influencing that likelihood.

3.4 Error Analysis

The confusion matrix for a classification threshold of 0.5 shows that the model classified all instances into the non-change class. This is because there were 522 true negatives and 75 false negatives, with zero true positives and zero false positives. This shows that even though the model gives higher probability scores to continuation cases compared to non-continuation cases, it does so below the 0.5 decision threshold. Such an observation points to a crucial difference between ranking and classification based on a certain threshold. The ROC-AUC value of 0.610 clearly indicates the model's ability to distinguish between the positive and negative examples. But because of the class imbalance and the conservative probability predictions, the predicted probabilities are always below the default threshold of 0.5. Therefore, the model's recall for the positive class is zero when tested for classification based on the threshold. The error pattern observed corresponds to the conservative calibration observed in the previous section. This model is not making confident predictions, especially for the minority continuation instances. This means that although there is some predictive information contained in the structural traceability scores, they are simply not high enough to result in a strong posterior probability given this model setup. Perhaps techniques for handling imbalance or thresholding could be explored in future work.

3.5 Comparison with Prior Studies

The level of predictive accuracy obtained in this study (mean ROC-AUC = 0.610) is fair in comparison to the previous work done in software change prediction and requirements engineering. There have been studies that have shown better classification accuracy using ensemble methods or neural networks. For instance, [1] used graph-based machine learning for defect prediction and showed better discriminative power using structural embeddings. [5] also used a supervised learning approach for requirement change prediction in small software systems and showed better predictive accuracy under their experimental setup. However, direct numerical comparison should be considered carefully. Many previous studies have utilized more complex feature representations, such as semantic embeddings, cross-project datasets, or larger labelled corpora. In contrast, the current study specifically targeted repository-based structural and historical metrics aggregated via temporal window modelling. While the goal of the current study was not solely to achieve the highest level of predictive performance, it aimed to explore whether probabilistic graphical modelling can identify interpretable dependency patterns in traceability data. In contrast to ensemble and deep learning methods, the Bayesian Network models the conditional dependencies between the features and the target variable explicitly. The results of the learned

structure show that the intensity of structural modification affects churn, which in turn affects the probability of change continuation. This is a methodological strength, although the predictive power is only moderate. The results are consistent with existing literature that showed historical repository metrics do have a measurable predictive signal. However, the fact that the AUC is moderate also supports the notion that it is difficult to predict the continuation of requirement change under conditions of imbalance in the short term. The work thus supports existing research by showing that probabilistic modelling gives clear insight into the dynamics of requirement evolution.

3.6 Overall Discussion

The experimental setup was designed to be transparent and reproducible. The Eclipse JDT repository was mined using PyDriller, and commit activity was limited to development after 2015 to better capture contemporary evolution trends. A total of 4,500 commits were retrieved, and bug identifiers embedded in commit messages were parsed to capture traceability relationships. This yielded 2,694 unique bug references, which served as the foundation of the traceability graph. Additional metadata was retrieved from the Eclipse Bugzilla database to capture bug creation timestamps and associated information necessary for temporal modelling. Because the repeated queries to the Bugzilla database were computationally expensive, intermediate results were serialized and saved as CSV files to ensure reproducibility of experimental results. Special care was taken to ensure consistency in timestamps and the presence of metadata, as missing metadata had a direct impact on the construction of temporal windows. To address the earlier ambiguities in the unit definition, a window-level formulation was used throughout the study. Each example is represented as a (bug_id, time window) pair, with the binary label denoting whether at least one commit happens in the following window. A 14-day sliding window was chosen to provide a suitable trade-off between granularity and sample size, resulting in 2,989 window instances. Of these, 376 were positive examples and 2,613 were negative examples, due to the class imbalance often found in change prediction tasks. Window-level aggregated features were derived from the traceability graph connecting bugs and commits, featuring churn, commit rate, author diversity, and average files modified. Log transformations were used to reduce skewness, and discretization was carried out using KBins Discretizer with quantile binning to improve the stability of probability estimation in the Bayesian Network. The above operational details were formalized to remove the earlier inconsistencies between requirement-level and window-level definitions. The modelling process involved the application of Hill Climbing with BIC scoring for the structure learning process as well as Bayesian estimation for the parameters with the BDeu prior. There is an important distinction between scoring for the structure and smoothing for the parameters. Initially, the evaluation process involved the use of stratified cross-validation. However, with the understanding that there is the possibility for temporal as well as group leakage, where multiple windows from the same bug would end up in both the training as well as the testing set, the approach to the validation process involved the use of group-aware cross-validation. Following this adjustment, the model's performance levelled off at a mean ROC AUC of approximately 0.61. Although the earlier models were closer to approximately 0.65, this may have resulted from leakage. The final reported performance, therefore, is based upon a more conservative and methodologically sound estimation. Although the ability to discriminate is not strong, it is at least moderate, and the results do show statistically significant improvements over random guessing, in addition to providing interpretable structural results concerning the dynamics of short-term change in software requirements. The degradation in performance after the application of leakage control is consistent with the importance of rigorous validation design and is consistent with best practices in empirical software engineering research.

3.7 Contribution

While existing research has successfully employed Bayesian Networks in requirement change prediction, this research's novelty is in its temporal and traceability-based operationalization of the problem rather than developing a novel probabilistic model. This research proposes a novel approach to requirement change forecasting as a prediction problem over a short-term window level using the Eclipse JDT traceability dataset. Rather than treating each change as a coarse-grained release-level phenomenon, each instance is defined as a pair of (bug_id, time window), and the target variable represents a change in the subsequent time window. This temporal segmentation allows for more fine-grained modelling of change continuation over a short horizon and avoids information loss that is common in release-level aggregation.

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026

4.0 Conclusions, Implications and Future Works

This chapter concludes the research by combining the empirical findings and interpreting their implications. The primary aim of this research was to establish whether traceability metrics could be used in a probabilistic graphical modelling approach to predict the likelihood of continuing requirement change in the short term. Unlike other researches, the primary aim of this research was to investigate whether probabilistic dependencies could be discovered in the structural data derived from the repository.

4.1 Interpretation of Findings

The empirical results clearly indicate that the probabilistic nature of requirement change continuation is not only present but also can be observed. The performance of the Bayesian Network model was a mean ROC-AUC value of 0.610, which is not a high value but was significantly better than random guessing. This clearly indicates that there is important predictive information contained within the repository's structural and historical metrics regarding short-term requirement change. The low standard deviation on the cross-validation folds also indicates that the model is not highly data-dependent. This is another factor that increases confidence in the robustness of the model. It is also important to observe that the Bayesian Network was able to use the provided data to identify conditional relationships between structural modification intensity, churn, and change continuation. The model clearly shows that modification volume is a mediator variable between repository activity and evolution behaviour. This is also consistent with theoretical views on software evolution, which identify structural coupling and modification intensity as important factors for change propagation. Nevertheless, the fact that the moderate predictive strength also emphasizes the complexity of requirement change behaviour. The influence of short-term continuation is determined by factors that are not reflected in the structural metrics, such as semantic intent, organizational practices, developer skills, and project priorities. The findings thus indicate that traceability-based structural metrics hold signal but are only a part of the overall change behaviour.

4.2 Future Research

Future work should be directed at extending the empirical study to a variety of repositories to increase generalizability. Cross-project learning paradigms can be used to assess the degree to which patterns of structural change discovered in one system generalize to other domains with varying architectural and organizational features. Furthermore, combining semantic embeddings from requirement texts, issue descriptions, and commit messages could provide better predictive accuracy by modelling meaning that goes beyond the scope of structural traceability features. This would enable the modelling framework to leverage both quantitative evolution information and qualitative intent features. Methodologically, one could pursue Dynamic Bayesian Networks or other temporal probabilistic graphical models to better model interdependencies over larger time horizons rather than just modelling within a single window. Learning strategies that account for imbalance and cost-sensitive probabilistic modelling may also improve minority class prediction while maintaining interpretability. Finally, the use of probabilistic graphical modelling in conjunction with deep learning for representation learning is an interesting area that could potentially lead to an improved balance between the two in the context of requirement change prediction.

Acknowledgment

I am grateful to my Programme Coordinator and Supervisor, Dr. Bilkisu, for your unwavering support throughout this programme. I am very grateful.

References

[1] J. Bryan and P. Moriano, "Graph-based machine learning improves Just-in-Time defect prediction," *PLoS ONE*, vol. 18, no. 4, 2023, doi: 10.1371/journal.pone.0284482.

[2] J. Cleland-Huang, C. K. Chang, and J. C. Wise, "Automating performance-related impact analysis through event based traceability," *Requirements Engineering*, vol. 8, no. 3, pp. 171–182, 2003, doi: 10.1007/s00766-003-0175-z.

[3] P. Dai, L. Yang, Y. Wang, D. Jin, and Y. Gong, "Constructing traceability links between software requirements and source code based on neural networks," *Mathematics*, vol. 11, no. 2, p. 315, 2023, doi: 10.3390/math11020315.

[4] D. Elumalai, "Leveraging AI and machine learning to improve agile backlog prioritization," *The American Journal of Engineering and Technology*, vol. 7, no. 5, pp. 211–218, 2025, doi: 10.37547/tajet/Volume07Issue05-21.

[5] R. Fatima *et al.*, "Requirement change prediction model for small software systems," *Computers*, vol. 12, no. 8, p. 164, 2023, doi: 10.3390/computers12080164.

[6] E. Giunchiglia, F. Imrie, M. v. d. Schaar, and T. Lukasiewicz, "Machine learning with requirements: A manifesto," arXiv preprint arXiv:2304.03674, 2024, doi: 10.48550/arXiv.2304.03674.

[7] M. Gradišnik, T. Beranič, and S. Karakatič, "Impact of historical software metric changes in predicting future maintainability trends in open-source software development," *Applied Sciences*, vol. 10, no. 13, p. 4624, 2020, doi: 10.3390/app10134624.

[8] S. Hassan, Q. Li, K. Aurangzeb, A. Yasin, J. A. Khan, and M. S. Anwar, "A systematic mapping to investigate the application of machine learning techniques in requirement engineering activities," *CAAI Transactions on Intelligence Technology*, vol. 9, no. 6, pp. 1412–1434, 2024, doi: 10.1049/cit2.12348.

[9] S. Hassan, M. Zubair, Q. Li, and A. Yasin, "Analyzing tools and techniques for evaluating requirements traceability," in *Proc. IEEE International Multi-Topic Conference (INMIC)*, 2024, doi: 10.1109/INMIC60434.2023.10465703.

[10] S. Ibrahim, N. Idris, M. Munro, and A. Deraman, "Integrating software traceability for change impact analysis," *The International Arab Journal of Information Technology*, vol. 2, no. 4, pp. 301–308, 2005.

[11] M. Iqbal and S. Badi, "Mitigating requirements volatility in agile software using AI and machine learning models," ResearchGate Technical Report, 2025, doi: 10.13140/RG.2.2.10193.65128.

[12] T. Li, Wang S., D. Lillis, and Z. Yang, "Combining machine learning and logical reasoning to improve requirements traceability recovery," *Applied Sciences*, vol. 10, no. 20, p. 7253, 2020, doi: 10.3390/app10207253.

[13] J. Lin, A. Poudel, W. Yu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Enhancing automated software traceability by transfer learning from open-world data," arXiv preprint arXiv:2207.01084, 2022, doi: 10.48550/arXiv.2207.01084.

[14] V. Musco, A. Carette, M. Monperrus, and P. Preux, "A learning algorithm for change impact prediction," in *Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 2016, pp. 8–14.

[15] V. Musco, M. Monperrus, and P. Preux, "A large-scale study of call graph-based impact prediction using mutation testing," *Software Quality Journal*, vol. 25, no. 3, pp. 921–950, 2017, doi: 10.1007/s11219-016-9332-8.

[16] A. M. Rosado da Cruz and E. F. Cruz, "Machine learning techniques for requirements engineering: A comprehensive literature review," *Software*, vol. 4, no. 3, p. 14, 2025, doi: 10.3390/software4030014.

[17] F. Tian, T. Wang, P. Liang, C. Wang, A. A. Khan, and M. A. Babar, "The impact of traceability on software maintenance and evolution: A mapping study," *Journal of Software: Evolution and Process*, vol. 33, no. 9, p. e2374, 2021, doi: 10.1002/smr.2374.

[18] F. Wen, C. Nagy, M. Lanza, and G. Bavota, "Quick remedy commits and their impact on mining software repositories," *Empirical Software Engineering*, vol. 27, no. 1, p. 14, 2022, doi: 10.1007/s10664-021-10051-z.

[19] H. Xu, "Applying the BERT algorithm for software requirements classification," ResearchGate Technical Report, 2024, doi: 10.13140/RG.2.2.33317.6832.

Received: 26/04/2026; Revised: 09/05/2026; Accepted: 25/05/2026; Published: 26/06/2026