Faculty of Natural and Applied Sciences Journal of Computing and Applications
Print ISSN: 3026-8133
www.fnasjournals.com
Volume 2; Issue 2; March 2025; Page No. 10-18.

Strategies for Early Identification of Failures in Agile Software Development Projects: A Review

[1]Nanwin, D.N., [2]Agaji, I., [2]Ogala, E., & [2]Gbaden, T.
[1]Department of Computer Science, Ignatius Ajuru University of Education, Port Harcourt, Nigeria
[2]Department of Computer Science, Joseph Sarwuan Tarka University Makurdi, Benue State, Nigeria

[*]**Corresponding author email**: kakusman@yahoo.com

Abstract
Agile software development has gained widespread adoption due to its iterative and adaptive approach to project management. However, despite its benefits, Agile projects are susceptible to failures that can impede project success. This paper focuses on developing effective strategies for early identifying and preventing of these failures in Agile software development projects. The aim is to provide project teams and stakeholders with actionable insights to mitigate failures and enhance project outcomes. The methodology involves a comprehensive review of literatures on Agile project management, early failure detection, and classification analysis using a novel failure detection analysis (FDA) model/ framework. Expected results include the formulation of a practical framework comprising proactive measures and best practices for early detection and prevention of software project failures. Suggestions for implementation include implementing refined machine learning algorithms, exploring performance metrics, conducting longitudinal studies, and empirical studies in diverse contexts while leveraging Agile project management tools for continuous monitoring and adaptation. In conclusion, by implementing the proposed strategies, Agile software development teams can proactively identify and mitigate potential failures, leading to improved software project success rates and stakeholder satisfaction.

Keywords: Failure Detection Analysis, Agile Software Development Lifecycle, Software Failure, Software Development

Introduction

A framework or approach applied to the creation of a software product is called a software development process. The software life cycle and software process are other names for it. A framework is a physical or conceptual structure that is meant to support and direct the construction of something that enlarges it into a more useful form (Lutkevich, 2020). There are several models for such processes, each describing approaches to a variety of software development process methodologies. Professional dissatisfaction with traditional approaches and the high failure rate of software development projects, along with the necessity for speedier software development to keep up with the rapidly changing business environment, gave rise to the concept and methodology of agile software development (Elbanna, 2014). It has been proposed that software development projects can fail for a variety of reasons. But the majority of the research on project failure tends to be quite generic, offering us lists of risk and failure criteria along with an emphasis on the detrimental effects of the failure on the company. Few studies have made an effort to thoroughly examine a number of unsuccessful initiatives in order to pinpoint the precise causes of the failure (Verner *et al.*, 2008). In this work, we develop an architectural framework for early detection of software development project failures. Agile software development places regard on working software rather than on comprehensive documentation.

The problem associated with other software project failure detection frameworks is that they are reactive; that is software project fault is already introduced into the system before the faults are detected. They did not consider a central software fault detection framework for the system, also not all the parameters that could lead to software project failure are considered by them. They do not give better and more accurate results. They cannot easily identify trends and patterns and cannot handle multi-dimensional and multi-variety data. However, it is a known fact that the No-

Cite this article as:
Nanwin, D.N., Agaji, I., Ogala, E., & Gbaden, T. (.2025). Strategies for early identification of failures in agile software development projects: a review. *FNAS Journal of Computing and Applications, 2*(2), 10-18.

Free-Lunch-Theorem (NFLT) is indeed a major barrier for such solutions as it is not realistic to make conclusions on one algorithm over another as no algorithm is inherently superior over another (Wolpert & Macready, 1997).

Relevant Literature

Song et al. (2019) introduced a system reliability modelling approach for analyzing software failures, particularly focusing on aerial support systems. While reliability is crucial for software quality, typical analysis methods may not adequately address the complexities of such systems. The proposed approach integrates System Theoretic Accident Modeling and Processes (STAMP) with system reliability modelling to address these challenges. The framework outlines reliability constraints and control models, offering a method for analyzing anomalies and inadequate control measures. Experimental results suggest that this novel approach may more effectively reveal software reliability issues in airborne support systems compared to existing methods. Capers (2010) identified several key causes of software project failure, including inadequate user input, stakeholder conflicts, vague requirements, poor cost and schedule estimation, mismatched skills, hidden costs, lack of planning, communication breakdowns, poor architecture, and late warning signals. Also, Zahid et al. (2018) conducted a critical analysis of situational factors contributing to software project failure and hindering success. They highlighted issues such as insufficient quality standards, limited understanding of development processes, and incorrect utilization of development approaches as primary causes of failure.

Christiansen et al. (2015) introduced a multiple regression model to predict risk factors involved in software development projects. Utilizing risk stratification and causal risk factor analysis combined with logistic regression, the model aimed to predict the probability of success or failure of software project development. While the model effectively grades and identifies risk factors crucial to the development process, it focuses on risk factors rather than failure factors. Additionally, it does not demonstrate whether identification occurs early in the software development phase. Also, Anju and Judith (2019) presented a data mining technique for predicting software defects, using efficient classification algorithms to anticipate errors before they manifest. This model operates by predefining the number of defects in a specific software product and predicts defects based on its size. The results are leveraged to enhance software quality and optimize resource allocation. However, a limitation lies in the inability to adjust the defect prediction model based on different project data, potentially leading to decreased test efficiency when encountering variances in training data characteristics.

Reddy and Babu (2013) developed a Logistic Regression Early-Estimation Model for predicting software project failure. This model provided accurate parameter estimates during the testing phase, leading to improved software reliability. However, a drawback is that failure behaviour detection typically occurs near project completion, during the testing and debugging phase, which may not allow sufficient time for cost and time-saving interventions. Procacinno et al. (2002) introduced a Case Study Approach for the early prediction of software project success or failures, highlighting factors such as weak specifications, lack of managerial support, and limited client engagement as influential. The study involved organizational case studies and participant surveys to explore these factors. While customer and user engagement were identified as crucial for project performance, a limitation is that the earliest prediction or detection occurs during the development phase, potentially limiting the opportunity for timely intervention. Fitzgerald et al. (2011) introduced an automated tool-implemented system for building failure prediction models, which compares various prediction techniques and their cost-benefit analyses to determine the likelihood of success in predicting failures. While the model's results demonstrated that automated prediction models outperform baselines for several failures, a limitation is that it focuses on avoiding failures rather than detecting them.

Additionally, Suma et al. (2014) presented a forecast of software project success using the Random Forest Classifier model, which employs bagging and feature randomness to build an ensemble of trees. The model's results showed that its output significantly differs from other approaches, with Random Forest proving effective across various project domains and complexities. This capability enables project managers to forecast project performance based on empirical investigations conducted with Random Forest. The model discussed presents a limitation in that it focuses solely on the success of software projects rather than failure, lacking insight into early project failure indicators. Conversely, Jeon et al. (2015) developed a probabilistic approach to predict the risks associated with software project failure by exploring software data repositories. This method generates a catalogue of possible events, their probabilities, and associated losses, providing a comprehensive view of future risks beyond historical data. Using Markov Chain, the model maps defect attributes across the software development lifecycle to predict future risk levels.

Cite this article as:
Nanwin, D.N., Agaji, I., Ogala, E., & Gbaden, T. (.2025). Strategies for early identification of failures in agile software development projects: a review. *FNAS Journal of Computing and Applications, 2*(2), 10-18.

While effective in determining risk threat levels in real-world mobile software ventures, the model's drawback lies in the absence of clarity regarding how early defects or failures can be predicted.

Wolf et al. (2009) utilized Social Network Analysis to predict build failures by analyzing developers' communication patterns. The study aimed to understand developer communities by mapping their relationships and identifying key individuals and groups. Investigating communication mechanisms within development teams with high collaboration needs, the study used data from IBM's Jazz project. Results showed that developer contact significantly impacted the quality of app integrations. Although no single measure could predict build success or failure, a combination of communication structure measures enabled the creation of a predictive model. This model achieved recall values ranging from 55% to 75% and precision values from 50% to 76% when applied to five project teams. However, the model's limitation lies in predicting failure during software application integration, potentially after significant time and resources have been invested in the project. Singh and Verma (2015) conducted research on fault prediction in the early stages of software development using cross-project data, focusing on design metrics. They performed empirical analysis to validate design metrics for cross-project fault prediction, employing the Naïve Bayes machine learning technique for evaluation. The study utilized seven datasets from NASA Metrics Data Program, incorporating design and code metrics. Software fault prediction models were developed using source code, processed metrics, and related fault data from the same or previous versions of code. Results indicated successful cross-project fault prediction during the design phase across seven public domain software development datasets. Additionally, the study recommended using the Area Under the Curve (AUC) as the primary accuracy indicator for comparative studies in software fault prediction, as it effectively separates predictive performance from class and cost distributions.

Golnoush and Selamat (2015) introduced the majority-ranking fuzzy Clustering method to enhance the accuracy of software fault prediction by addressing the impact of irrelevant and inconsistent modules. Their study aimed to mitigate this effect by clustering all project modules within a new framework. Results demonstrated that fuzzy clustering reduced the negative impact of irrelevant modules on prediction performance. Evaluation using eight datasets from NASA and Turkish white-goods software showcased the model's superiority in terms of false positive rate, false negative rate, and overall error compared to other prediction models. Specifically, the approach achieved a 3% to 18% improvement in false negative rate and a 1% to 4% improvement in overall error across more than half of the testing cases when compared with other proposed models (ACF and ACN). However, the model's limitation lies in its failure to specify how early software faults can be detected and clustered to prevent losses. Hu et al. (2009) proposed an intelligent model to predict and manage risks inherent in software projects, recognizing the high failure rates associated with software development. The model employed machine learning algorithms such as Artificial Neural Networks (ANN) and Support Vector Machine (SVM) to identify risks by gathering real-life instances from software development companies. Data collected through questionnaires informed risk prediction across various projects, although the model did not specify the timing of risk identification or the project phase in which risks were identified. Conversely, Batarsch & Gonzalez (2015) presented a data analytics model for predicting failures in agile software development, utilizing mean-time between failures and regression modelling, powered by Analytics Driven Testing (ADT). The model leveraged R statistical language for data analysis and mining, providing graphical representations and machine-learning approaches to aid decision-making. However, the model did not explicitly detail the phase of the agile software development life cycle in which failures are detected or how early they can be identified. Ibraigheeth & Fadzli (2020) developed a Logistic Regression Model for software project failure prediction by collecting real-life data from reports, case studies, and surveys. The model estimated project outcomes and provided probabilities of failure to aid decision-making, although it did not specify the phase of failure detection or how early it occurred. Kaur & Senlgupta (2011) presented a research method for software project failure analysis, analyzing current process models during development, but did not conduct the analysis at appropriate phases, making it challenging to determine early failure detection. Bicer et al. (2011) proposed defect prediction using social network analysis on issue repositories, aiming to understand developer interactions and their impact on product quality. While the model effectively lowered false alarm rates or increased detection rates, it did not specify the timing of predictions or the phase of failure detection.

Pandey and Kumar (2023) conducted a survey on recent developments in software project failure detection for imbalanced data, addressing issues prevalent in analysis with such datasets. They found SMOTE to be a commonly used sampling technique to address data quality issues. However, the study mainly focused on reviewing existing literature rather than proposing new models or methods.

Cite this article as:
Nanwin, D.N., Agaji, I., Ogala, E., & Gbaden, T. (.2025). Strategies for early identification of failures in agile software development projects: a review. *FNAS Journal of Computing and Applications, 2*(2), 10-18.

Batool and Khan (2022) conducted a systematic literature review using data mining, deep learning, and machine learning approaches to detect software project failure. They analyzed previously published surveys, reviews, and related work to extract and respond to questions that were either unanswered or needed further exploration. However, their focus was on developing and answering new questions rather than examining the various phases of the software development life cycle (SDLC).

Methodology

The methodology used in the identification of software failures in an Agile Software development project is the failure detection analysis (FDA) method a novel approach to early detection of software failure and classification. Given that, failure can occur in any phase of the software development process and project, this methodology helps to scan through the process in each phase of the Agile software development life cycle. The architecture is shown in Figure 1

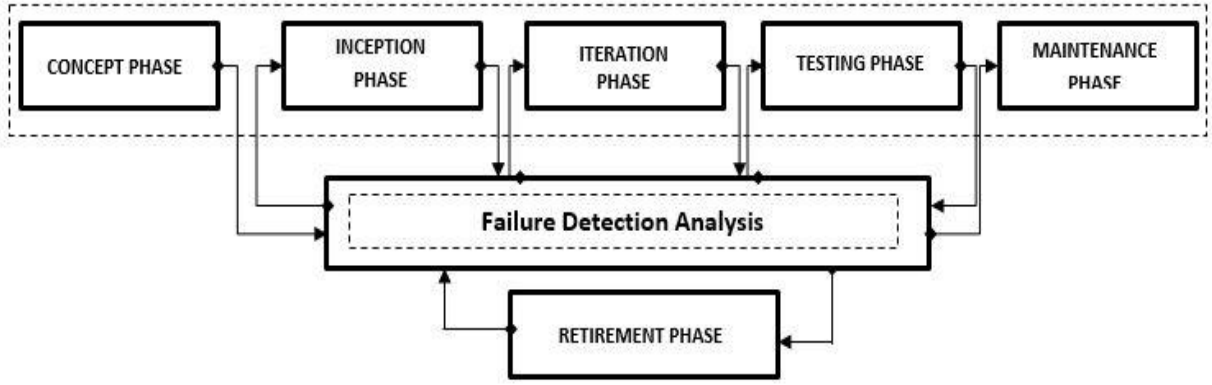

Figure 1 System Structural Framework for Identification of Failures in Agile Software Development Projects

Figure 1 provides a structural overview of the developed general framework, emphasizing the potential occurrence of software project failure from the concept/scoping phase to the retirement phase. Notably, inadequate scope-out analysis is a significant contributing factor to most software project failures. The illustration incorporates a failure detection analysis check during and at the conclusion of each phase before transitioning to the next phase of the Agile software development lifecycle. This ensures that failure analysis is conducted at every phase to identify early indicators of failure and classify potential reasons for project failure. Each phase's output undergoes rigorous failure detection analysis before proceeding to the next phase.

Components and Strategy of the Methodology

Concept Phase

The failure detection analysis is carried out during the concept/ scope-out phase to determine if the scope of the project, the priority list and the requirement analysis of the software project is feasible. The FDA carried out, a check mates the features and the proposed results and estimated the time and cost potentials of the software project to see if it is attainable. Where there is any lag in the cycle the FDA detects the possible failure and classifies it.

The failure detection analysis is done at the end of the concept phase. Making sure that everything is in place and no detail is neglected can cause possible failure of the software project and if any is identified it is quickly classified as a possible failure sign. Some of the signs to look out for are to make sure that there is no "ambiguity in system requirement" and the definitions and clarifications of the requirement must be made available.

Inception Phase

In the inception phase, when carrying out the FDA, a check is done to make sure that the right team has been selected and their available for the development of the software project. The team makes sure that all the necessary tools and resources are available to carry out the project. The user interface mock-up and the built architecture are also scrutinized. All stakeholders of the system must be fully involved to determine the product's functionality.

During the project, regular check-ins are done with the failure detection analysis module or steps to ensure that the requirements are followed through with the design process. Every step of the failure detection analysis is duly followed to ensure a successful software project.

Cite this article as:
Nanwin, D.N., Agaji, I., Ogala, E., & Gbaden, T. (.2025). Strategies for early identification of failures in agile software development projects: a review. *FNAS Journal of Computing and Applications, 2*(2), 10-18.

Iteration Phase
In this phase of the Agile software development life cycle, the developers, using the Design Document Specification (DDS); the actual coding of the system is started. The FDA ensures that the product requirements and customer feedback are met during the code development. By the end of the iteration or sprint, the FDA ensures that the functionality goals are met. The FDA team works together with the software development team to ensure a working software is produced.
The failure detection analysis is done throughout the entire iteration phase because it is the longest phase since the bulk of the work is done in this phase. During the various rounds of revision, the FDA is used to scan the process and as the project requirements expand.

Release Phase
The release phase is very important in that, the software product is about to be released into the world. Hence, there is a need to carry out system testing, and to finalize the system and user documentation before the software is released to the world. During the system testing, the FDA is used to scan for bugs that may arise during the running of the system and can possibly lead to failure. The functionality of the software is also analyzed. Detected defects are addressed and tested again to finalize system and user documentation.

Maintenance
To make sure the deployed system is functioning as required, the FDA team monitors the maintenance/ support team to make sure they provide the required support and maintenance working on user feedback and complaints. The software project product ends at this stage. But, if there is any plan for retirement of the software product, then the next phase comes in.

Retirement Phase
The retirement phase of the software is also very important, especially, when it is a migration to an updated version of the system. The FDA is used to check that the various activities or major functions of the older version is also domiciled on this new version except for the part that is being dropped entirely. The FDA also checks that no data is lost during migration before retiring the older version of the system and before support is removed entirely from the system, end-of-life activities is carried out.

Failure Detection Analysis
The proposed system is an efficient framework for the early detection of software development project failure and in order to achieve this early detection, we have to look into the various stages in the development life cycle from which attention is given can detect if the software project is failing or there is reasonable progress with the process. The failure detection analysis can run side by side with the development process and/ or at the end of each phase using the SDLC stages as a guide. Figure 2 is a sub-framework from the general architectural framework that carries out the failure detection analysis of the proposed system.

Phase Activity Enquiry
Phase activity enquiry checks the various activities carried out in each Agile phase. During this enquiry process, eight (8) early signs of software project failure were used as a benchmark to analyze the phases. These signs include working in parallel, a big team from the beginning, documentation piled up, a No Ask – No Tell policy, requirements clarifications delayed, crucial tasks outsourced, integration of multiple products/ technologies and management ignorance. These signs or signals are checked for throughout all the phases of the development life cycle. Documents are compiled to help for a clearer analysis of what is being transpired in the development process of the system.
This stage is very important in the failure detection analysis in the sense that, data are being compiled in this stage. The data compiled are documented for analytical purposes from which indications for failures can be identified. The data gathering method used here is through questionnaire and interviewing (interacting with) key players in the industry. The documentation of the system requirement, functions, features (what must be done) and tools to be used for the development of the software project is reviewed and documented.

Failure Activity Identification
The section of the failure detection analysis is done based on the documented enquiries from the phase activities enquiry section. The task here is to be able to identify these signs as each phase of the software development is going

Cite this article as:
Nanwin, D.N., Agaji, I., Ogala, E., & Gbaden, T. (.2025). Strategies for early identification of failures in agile software development projects: a review. *FNAS Journal of Computing and Applications, 2*(2), 10-18.

on and to scale it with respect to a certain percentage. From these enquiries and documentation, the activities of the software project are identified and if there is any failure signal, they are detected.
From the failure activity enquiry, some information were recorded during the enquiry or enquiries. The machine learning algorithm, and decision tree were used to perform an analysis and based on the result(s) of the analysis, a decision was reached. When these activities are identified, corrections can be made immediately. But, that will be determined by the percentage of their occurrence of that sign or signal during the software development process. Every step of the software project has a target and tools required to accomplish that phase.

Failure detection

This section discusses how the failure of the software project can be detected early in the development process. It is the main idea of this framework; to be able to detect software project failure before it is too late and one cannot remedy the situation. This will encourage software project continuity and the general success of a software development project.
Failure detection is possible after the various failure signals in each phase have been identified and analyzed. Looking at their statistics, one can spot the red flags and at what aspect of the Agile software development lifecycle phases there is likely to be failure and correction measures or due attention can be given to that phase.
The application of the Ensemble learning model, Bagging, Decision Tree, Random Forest and Gini Impurity machine learning techniques are used to analyze the outcome of the identified signals during the phase activity enquiry and failure activity identification. These machine learning techniques were used to perform the failure detection analysis throughout the process. The data are fed into the algorithms of each machine learning technique and a comparison is done on the output result of each of the algorithms. With the results obtained, conclusions were reached on whether the software is going to be successful or it's going to fail.

Failure Classification

Classification or categorization is done to keep things under certain headings to be able to attach a name or identity to a given failure. This section helps us to achieve this purpose of giving the failure signal an identity; to know where the failure is coming from in each phase of the software development life cycle.
The detected signal can be classified to be a requirement gathering/ analysis problem, user/ customer problem, top-level management problem, technical problem and/ or development problem. Classifying it this way will further help to segment and isolate the solution process so that it can be tackled on time.
During the failure detection analysis of each phase of the system, we look out for the failure sign or signal as stated earlier in this work and classify the detected failure signal with respect to the different failure signals as expected in each phase of the Agile software development lifecycle.

Detailed System Design

The proposed framework will adopt the flowchart diagram in the description of the system. A flowchart diagram is used to demonstrate the functionality of the framework. The proposed framework for early identification of software development project failures has been represented in the flowchart diagram as shown in Figure 2.

System Flowchart Diagram

The system flowchart presents the flow of control during the failure detection analysis process. The controls scan through the different phases of the SDLC. This is demonstrated in Figure 2.

Cite this article as:
Nanwin, D.N., Agaji, I., Ogala, E., & Gbaden, T. (.2025). Strategies for early identification of failures in agile software development projects: a review. *FNAS Journal of Computing and Applications, 2*(2), 10-18.

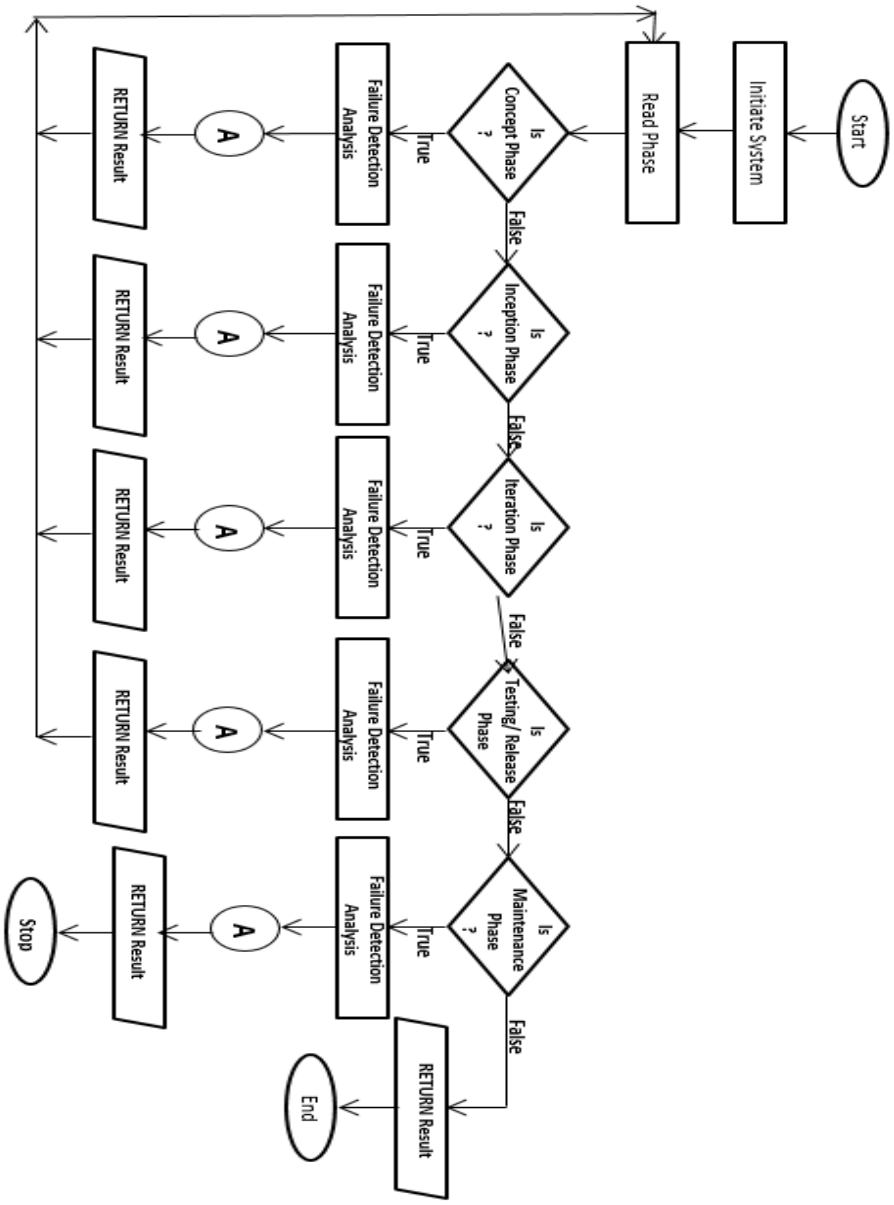

Figure 2: System Flowchart for the Proposed Framework.

Figure 2 demonstrates the flow of instructions on how the system scans through the various phases of the ASDLC using a WHILE loop construct. The 'while loop' for each phase runs the FDA and returns the result of the analysis. The returned result is the output result of each phase and could be either a positive or negative response to the failure detection analysis. This is because, at each phase, the FDA analyzes the phase looking for any failure signal. In the event of any detected failure signal, it will be outputted to show the class of failure it belongs for correction.

From the flowchart in Figure 2, the system is initiated and scans for the first phase "Concept/ Scope-out Phase" of the Agile software development life cycle, the FDA steps as seen in the flowchart are followed and the result is returned to the next step. The output returned at each iteration is the presence or absence of failure signs of each phase of the Agile development life cycle which may include ambiguity of the requirement and features, wrong team selection, wrong tool selection, ambiguous mock-ups, low-quality product, unfixable bugs or defect and so on. The return goes to the beginning of the WHILE LOOP and scans for the next phase. The WHILE LOOP will terminate only when it has been checked or scanned through all the phases of the Agile software development life cycle. It starts from the first phase to scan, if it is not that phase it follows the false direction to the next phase and if it is that phase it performs

Cite this article as:
Nanwin, D.N., Agaji, I., Ogala, E., & Gbaden, T. (.2025). Strategies for early identification of failures in agile software development projects: a review. *FNAS Journal of Computing and Applications, 2*(2), 10-18.

the FDA steps and returns the result to the next phase. When it gets to the final phase, it returns the result and stops to signify the end of the process. The FDA follows through the flow of control while performing the failure detection analysis during the various phases of the agile software development lifecycle. The FDA starts with the phase activity enquiry and tries to identify failure activity(ies) and during the process, if any of the failure signal is spotted, the failure detection step detects the signal and is classified by the failure classification step of the FDA. The failure signal for the particular phase is returned if the condition is positive.

Technical Suggestions

Refine Machine Learning Algorithms: Future research should focus on refining the selection and integration of machine learning algorithms within our framework or strategy. Improving the predictive accuracy and robustness of these algorithms will enhance the effectiveness of our framework in identifying software project failures.

Explore Additional Metrics: Exploring additional metrics and features that may contribute to project failure prediction could further enhance the predictive capabilities of our framework. By considering a broader range of factors, we can better anticipate and mitigate potential failures throughout the software development lifecycle.

Conduct Longitudinal Studies: Longitudinal studies tracking project outcomes over time would provide valuable insights into the evolution of failure indicators. This data could inform proactive failure management strategies and help refine our framework to better adapt to changing project dynamics.

Empirical Studies in Diverse Contexts: Conducting empirical studies in diverse software development contexts and industries would validate the generalizability and effectiveness of our framework. By testing our approach across various scenarios, we can ensure its applicability across different organizational settings.

Overall, continued research in these areas holds promise for advancing early detection techniques and improving the success rates of Agile software development projects.

Conclusion

This paper addresses the importance of early identification and prevention of failures in Agile software development projects. Despite the widespread adoption of Agile methodologies, project failures remain a concern. The paper's focus on developing effective strategies for early detection aims to provide actionable insights for project teams and stakeholders. The methodology involves a thorough review of the literature, culminating in the formulation of a practical framework for failure detection and prevention. A novel failure detection analysis has been introduced which is the main engine for identifying faults in the Agile software process. Suggestions for implementation include refining machine learning algorithms, exploring performance metrics, and conducting longitudinal and empirical studies. By leveraging Agile project management tools for continuous monitoring and adaptation, teams can proactively identify and mitigate potential failures, ultimately leading to improved project success rates and stakeholder satisfaction.

References

Anju, A. J., & Judith, J. E. (2019). Software Prediction Using Efficient Classification Algorithm. *International Journal of Recent Technology and Engineering*, 8 (3), 301 – 304.

Batarseh, F. A., & Gonzalez, A. J. (2015). Predicting failures in agile software development through data analytics. *Software Quality Journal*, 26(1), 49-66.

Batool, I., & Khan, T. A. (2022). Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review. *Computers and Electrical Engineering*, 100, 107886.

Bicer, S., Bener, A. B., & Çağlayan, B. (2011,). Defect prediction using social network analysis on issue repositories. *In Proceedings of the 2011 International Conference on Software and Systems Process*, (pp. 63-71)

Capers, J. (2010). Patterns of Software Systems Failure and Success. *International Thompson Computer Press*, Boston, Mass.

Christiansen, T., Pongpisit W., Somchai, P., & Sakda A. V. (2015). Prediction of Risk Factors of Software Development Project by using Multiple Logistic Regression. *ARPN Journal of Engineering and Applied Sciences*, 10(3).

Elbanna, A. (2014). Identifying the Risks associated with Agile Software Development: an Empirical Investigation. *Mediterranean Conference in Information Systems (MCIS)* At: Verona, Italy.

Fitzgerald, C., Letier, E., & Finkelstein, A. (2011). Early Failure Prediction in Feature Request Management Systems. *In Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference*, Trento, Italy, 2 September 2011; 229–238.

Golnoush, A., & Selamat, A. (2015). Increasing the Accuracy of Software Fault Prediction Using Majority Ranking Fuzzy Clustering. *International Journal of Software Innovation*, 2(4), 61 – 71.

Hu, Y.; Zhang, X., Sun, X.; Liu, M.; Du, J. (2009). An intelligent model for software project risk Prediction. *In Proceedings of the 2009 International Conference on Information Management, Innovation Management and Industrial Engineering*, Xi'an, China, 27 December 2009; 629–632.

Ibraigheeth, M. & Fadzli, S. A. (2020). Software Project Failures Prediction using Logistic Regression Modeling. *IEEE 2020 2nd International Conference on Computer and Information Sciences* (ICCIS).

Jeon, C., Kim, N., & In, H.P. (2015). Probabilistic Approach to Predicting Risk in Software Projects Using Software Repository Data. *International Journal of Software Engineering and Knowledge Engineering*, 25, 1017-1032.

Kaur, R., & Sengupta, J. (2011). Software Process Models and Analysis on Failure of Software Development Projects. *International Journal of Scientific and Engineering Research*, 2(2).

Lutkevich, B. (2020). Framework: IT Standards and Organization. TechTarget. https://www.techtarget .com/ whatis/ definition/ framework. 21/03/2024

Pandey, S., & Kumar, K. (2023). Software Fault Prediction for Imbalanced Data: A Survey on Recent Developments. *Procedia Computer Science*, 218, 1815-1824.

Procaccino, J. D., Verner, J. M., & Overmyer, S. P. (2002). Case Study: Factors for Early Prediction of Software Success & Failure. *Information and Software Technology*, 44(1), 53-62.

Reddy, K.V.S., & Babu, B.R. (2013). Logistic Regression Approach to Software Reliability Engineering with Failure Prediction. *International Journal of Software Engineering & Applications*, 4(1), 55.

Singh, p., & Verma, S. (2015). Cross Project Software Fault Prediction at Design Phase, World Academy of Science, Engineering and Technology. *International Journal of Computer and Information Engineering*, 9(3).

Song, J., Zhao, H., Li, X., Yang, Y., Liu, C. & Li, H., (2019). A New Software Failure Analysis Method Based on the System Reliability Modelling. *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, 1143-1149.

Suma, V., Pushphavathi, T.P. & Ramaswamy, V. (2014). An Approach to Predict Software Project Success Based on Random Forest Classifier. *Journal of Advances in Intelligent Systems and Computing*.

Verner, J., Sampson, J., & Cerpa, N. (2008, June). What factors lead to software project failure? In *2008 second international conference on research challenges in information science* (pp. 71-80). IEEE.

Wolf, T., Schroter, A., Damian, D., & Nguyen T.H.D. (2009). Predicting Build Failure using Social Network Analysis on Developer Communication. *IEEE 31st International Conference on Software Engineering*, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings.

Wolpert, D. H., & Macready, W. G., (1997). No Free Lunch Theorems for Optimization IEEE Transaction on Evolutionary Computation 1(1), 67 – 82.

Zahid, A.H.A., Haider, Farooq, M.W., Abid, M.S., & Ali, A. (2018). A Critical Analysis of Software Failure Causes from Project Management Perspectives. *VFAST Transactions on Software Engineering*, 6(1), 62-68.