International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 10 Issue 3, May-Jun 2026 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Comparative Analysis of Software Defect Estimation Techniques

Tehseen Fatma[1], Dr. Najeeb Ahmad Khan[2], Dr. Saoud Sarwar[3]

[1,3]Department of Computer Science and Engineering, Al-Falah University, Faridabad, Haryana, India
[2]Department of Computer Science, Arunachal University of Studies, Namsai, Arunachal Pradesh, India

ABSTRACT

Software defect estimation is a critical aspect of software development that helps manage project schedules, allocating resources effectively, and ensuring software quality. Various techniques have been proposed and used to estimate software defects, including historical data analysis, expert judgment, function point analysis, software metrics, and machine learning approaches. This comprehensive research survey paper aims to review and compare these techniques in terms of their strengths, weaknesses, and applicability in different software development contexts. The findings from this analysis will assist researchers, practitioners, and decision-makers in selecting appropriate defect estimation techniques for their specific software projects. Through an extensive literature review and analysis of case studies, this paper evaluates the strengths and weaknesses of each technique, highlighting their accuracy, precision, data requirements, scalability, transparency, interpretability, flexibility, and resource requirements. Furthermore, the applicability of these techniques in various software development contexts, such as small-scale projects, medium-scale projects, large-scale projects, agile development environments, safety-critical systems, open-source projects, and distributed development teams, is examined. It also identifies areas for future research and improvement, paving the way for the development of more accurate and effective defect estimation methods.

KEYWORDS: *Software defect estimation, machine learning, artificial intelligence.*

How to cite this paper: Tehseen Fatma | Dr. Najeeb Ahmad Khan | Dr. Saoud Sarwar "Comparative Analysis of Software Defect Estimation Techniques" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-10 | Issue-3, June 2026, pp.1281-1290, URL: www.ijtsrd.com/papers/ijtsrd133330.pdf

IJTSRD133330

1. INTRODUCTION

Software development projects are complex endeavors that involve multiple stakeholders, resources, and activities. One of the key challenges in software development is accurately estimating the number of defects that may arise during the development life cycle [1–4]. Defects, also known as bugs or issues, can have a significant impact on the overall quality, functionality, and reliability of software products. Therefore, it is crucial for project managers and development teams to have a reliable estimate of the expected defects to effectively plan resources, schedule activities, and allocate efforts for quality assurance and testing. Numerous studies examined the efficacy of developing metrics for defect estimation. Schroter et al. [5] demonstrated that software architecture and failure records might be leveraged to create a predictor of faulty components. Jiang et al. [6] evaluated the effectiveness of design-level prediction models to those based on code-level criteria. They discovered that models constructed using two level measures performed better than those developed using either one of the two groups of metrics individually. The research done by Krishna et al. [7] shown that open-source object-oriented design complexity measures were significantly associated with faults. Ohlsson et al. [8] conducted an empirical investigation on the Ericsson Telecom AB project, and their findings suggested that design metrics may be used to forecast which modules are most likely to have failures. In addition to this, the design metrics, which were derived from the design phase artefacts as well as design diagrams like flow graphs and UML diagrams, were utilized at the NASA IV&V facility [9]. Defect estimation techniques provide a systematic approach to predicting the number of defects that can be anticipated in a software project. These techniques rely on various methods, models, and metrics to analyze historical data, expert

judgment, functional complexity, software metrics, or even leverage machine learning algorithms to make accurate estimations [10]. After estimating defects, project managers can proactively manage risks, allocate appropriate resources, and ensure that sufficient time and effort are dedicated to identifying and resolving defects throughout the software development life cycle. Accurate defect estimation is essential for several reasons. Firstly, it helps project managers allocate resources effectively [11]. Having a rough idea of how many bugs will likely be present helps project teams set up sufficient time, resources, and manpower for quality assurance activities like testing, code reviews, and inspections. Second, estimating the number of defects helps in organizing time. It allows project managers to allocate sufficient time for testing and bug fixing, preventing schedule overruns and potential delays in software delivery. Thirdly, defect estimation enables the assessment of software quality [12]. Stakeholders in a project are better able to evaluate the software's quality and pinpoint areas that may need more work if they have an idea of how many bugs are likely to be there. Emerging demand was recently identified by Barry [13] as one of the five expected difficulties in measuring software. Bell et al. [10] performed one of the early investigations on the significance of the software prerequisites. Software requirements were deemed crucial, and their issues were found to be consistent among different projects. Requirements that pertain to a system's quality qualities are extremely helpful in creating high-quality software. When requirements are stated quantitatively, or when they can be measured, they may be utilized to construct a predictor of fault-prone modules. Defect prediction models were developed by Jiang et al. [14] using requirements, product/module code metrics, and a combination of the two. They hypothesized that the measurements gathered early in a system's life cycle may be crucial for foreseeing potential failures. Effectiveness, cost, and time constraints are only some of the difficulties in software estimating. It is predicted that the majority of the available methods for doing so will be utilized. These methods are straightforward and simple to apply to a basic project, but their inaccuracy and erroneous estimations are a major drawback. The accuracy of estimations can be improved by the use of hybrid and combined two or more techniques. The rest of the paper is organized as follows. In Section 2, we present a comprehensive review of the key studies related to software defect estimation techniques. Section 3 provides an overview of various estimation techniques employed in the field. In Section 4, we classify these techniques and conduct a comparative analysis, considering factors such as accuracy and precision. Finally, in Section 5, we conclude the paper, summarizing the findings and discussing their implications.

2. REALTED WORD

The concepts and techniques for estimating software costs have been categorized and reported on by Suri at al. [15]. Estimation via Analogy, Expert Opinion, Putnam's Software Life-cycle Model (SLIM), and COCOMO are the most common approaches for determining how much time and cost will be needed to create a piece of software. Various studies have been conducted on the estimation of software defects. Estimating software defects refers to the process of predicting or determining the number of defects or bugs that are likely to be present in a software system or application. This estimation is typically done during the software development life cycle to assess the quality and reliability of the software.

In order to enhance the accuracy of software approximation techniques, Waghmode et al. [16] merged algorithmic and non-algorithmic approaches, Function point size, COCOMO, and ANN. The research used a hybrid model consisting of all three methods, and found an improvement in accuracy. Time and precision are two areas where the suggested model excels. As a result, there has been an improvement in the accuracy and correctness of software estimation. While the suggested model gives various classification options, the accuracy is only mediocre at best; adding a KPCA-based model would significantly boost it.

Suri et al. [15] proposed a back propagation neural networks method to accomplish software cost estimate in their study. An Adaptive Learning Approach to Software Cost estimate, which takes an adaptive learning approach. The model is trained and tested with the COCOMO dataset and the COCOMO NASA 2 dataset. These data sets are derived from simulations and working prototypes. The blended, integrated neural network back propagation's accuracy surpasses that of the conventional COCOMO.

A proposal has been put forward to enhance the efficiency and accuracy of the COCOMO model, which is a software effort estimation system [17]. This proposal suggests using artificial neural networks to improve the prediction accuracy. The approach involves using a multi-layer feed-forward neural network along with its associated parameters. Authors trained this network using a backpropagation learning algorithm with a COCOMO dataset and results of the study indicate that the proposed neural network model improves the accuracy of the COCOMO model's estimation. However, a drawback

of neural networks is their reliance on the training set. Despite this limitation, neural networks are currently considered the best technique available. The neural network model demonstrated better results compared to the traditional COCOMO model. Additionally, another estimation model has been proposed that combines different techniques to enhance cost estimation accuracy, providing an alternative to the limitations of existing approaches.

According to Gupta et al. [18] a failure to accurately estimate the number of software defects in the product's early phases can result in significant losses in terms of both money and time. Using a decision table, the suggested rule-based test case reduction approach demonstrated that problems might surface in later phases of software development. In addition, software development organizations are likely to lose a significant amount of money and time due to faults in the first stage of SDLC that are not responded to accurately and effectively. In addition, the proposed method doesn't assume that the tester be fluent in either programming languages or logic. The model's efficiency is demonstrated by the fact that it reduces duplication by around 33%, which has a major impact on both cost and time savings. Authors [19] reviewed the Delphi approach and set out to investigate whether or not it could be used to gather complicated qualitative data from consultants and competent experts. This necessitates that the data be as accurate and trustworthy as feasible. Another study by Nielsen [20] presented an integrated approach to software estimating. Managers of projects should think carefully about conducting extensive study before settling on a cost estimating approach. The process of estimating software costs is meant to be guided by logic and expert advice.

Gotovac et al. [21] used effort estimation methods to mimic expert effort estimate by creating data from a real-world setting. The difficulties encountered by specialists during estimating are highlighted in their study. The concept is limited to settings where the use of expert techniques is prevalent throughout software product development. The best predictors that may be used for accurate estimate can be generated by the model. During the prediction process, the predictors are utilized to determine which input variables are crucial. Despite their best efforts, experts continue to face problems stemming from the estimating process itself and the sources of mistake they encounter.

Karna et al. [21] examined many methods for estimating software costs. Estimation and its impact on software development expenses were among the topics covered in this article. However, models like Fuzzy logic and logarithms may be used to get to the cost and expert effort level. The two models are also obviously superior than other models in terms of accuracy. In contrast, the hardest task in any software industry is estimating how much time and money something will take to develop.

3. AN OVERVIEW OF SOFTWARE ESTIMATION TECHNIQUES

Software estimating methods are often divided into the six groups as illustrated in Figure 1. Software cost and effort may be calculated with the use of mathematical and trial equations or a formula provided by a traditional Algorithmic technique[15]. This technique relies on calculations based on inputs and historical data, including scale factors, cost variables, and so on. Furthermore, COCOMO, FPA, COCOMOII, etc. are some of the cutting-edge prototypes in the realm of algorithms. In addition, a non-algorithmic approach was used by making use of previously collected data that fit comparable criteria [22].

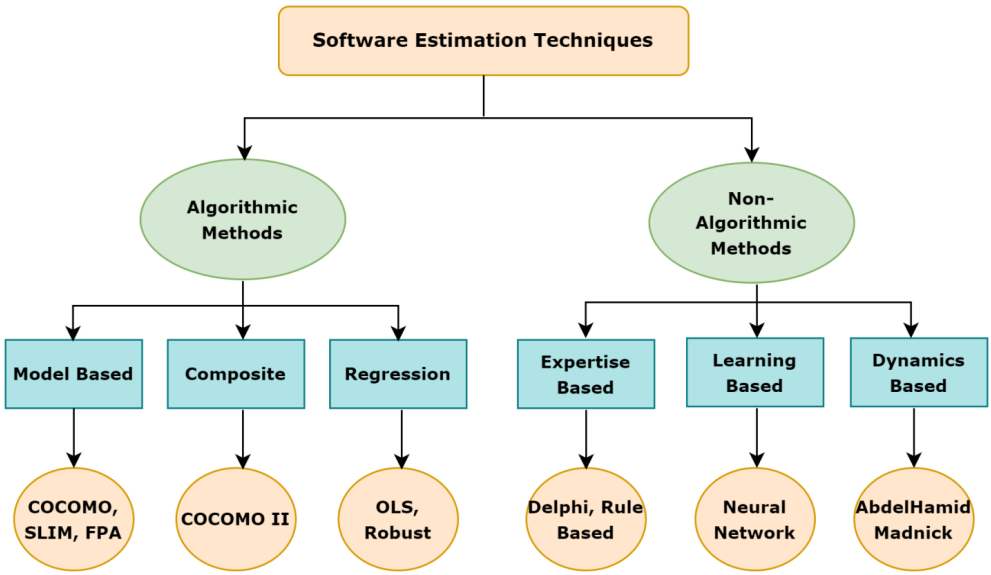

Figure 1. Software Estimation Techniques

Novel ways that rely on soft calculation, such as equivalency, judgements of the expert, Delphi, etc., are utilized in this method of estimating in place of the more traditional hard calculation. Both algorithmic and non-algorithmic software estimating approaches may be applied to the development of object-oriented systems. Let's delve deeper into each of the classification points for software estimation techniques applicable to the development of object-oriented systems. We are able to divide them up based on the following criteria:

3.1. Model-Based Estimation Techniques

Model-based estimation techniques involve constructing mathematical or statistical models that represent the relationships between various software development factors and the resulting effort or time required [23]. These models are based on historical data, industry best practices, and expert knowledge. Examples of model-based estimation techniques include COCOMO (COnstructive COst MOdel) and Function Point Analysis (FPA).

3.2. Expertise Based

Expertise-based estimation techniques rely on the knowledge and experience of software development experts. These experts use their domain-specific knowledge, judgment, and intuition to estimate the effort, time, and resources required for a project. Expertise-based techniques often involve expert judgment, expert opinion, and analogies with similar projects. Delphi Technique, Wideband Delphi, and Expert Judgment are some examples of expertise-based estimation techniques.

3.3. Learning Based

Learning-based estimation techniques involve using machine learning algorithms to analyze historical project data and identify patterns and relationships between various project factors and the resulting effort or time. These techniques leverage past project data to make predictions about future projects. Machine learning algorithms, such as regression analysis, neural networks, and decision trees, are commonly used in learning-based estimation.

3.4. Regression Techniques

Regression techniques are statistical methods used to model the relationships between independent variables (project factors) and dependent variables (effort, time, etc.). Regression analysis helps identify the mathematical equations or models that best represent these relationships. Software estimation techniques that employ regression analysis include linear regression, multiple regression, and non-linear regression.

3.5. Composite

Composite estimation techniques combine multiple estimation methods or factors to arrive at a more accurate estimate. These techniques leverage the strengths of different estimation approaches to compensate for their limitations. For example, a composite technique may incorporate both expert judgment and historical data analysis to improve estimation accuracy. COCOMO II is an example of a composite estimation technique that combines expert judgment, historical data, and mathematical models.

3.6. Dynamics Based:

Dynamics-based estimation techniques focus on capturing and analyzing the dynamic aspects of software development projects. These techniques consider factors such as project complexity, team dynamics, resource allocation, and project progress to refine and update the initial estimates throughout the project lifecycle. Agile estimation techniques, such as Planning Poker and Burn-down Charts, fall under the dynamics-based category. It's important to note that these estimation techniques are not mutually exclusive, and they can be combined or adapted based on the specific project requirements and available data. Software estimation is a complex task, and selecting the most appropriate technique depends on factors such as project size, complexity, available resources, and the level of accuracy required.

Table1: Comparative analysis of algorithmic and non-algorithmic models in a broad sense.

Category	Sub-category	Techniques	Superiority	Inferiority
Algorithmic	Model-based	COCOMO, Function Point Analysis, Slim	Capable of producing consistent estimates repeatedly.	Incapable of handling exceptional situations.
	Composite	COCOMO II	Easily adaptable to enhance input data, refine formulas, and convert them.	Expert judgment cannot be easily quantified.

	Regression	Ordinary Least Squares, Robust	Efficiently facilitates sensitivity analysis.	
Non-Algorithmic	Expertise	Delphi, Rule-based	Beneficial in the absence of historical data. Beneficial when project scope is limited. Enables obtaining a concise and rapid outcome. Essentially, it allows for straightforward predictions.	An expert team must reach a consensus for it to be useful. It requires a significant amount of time due to the involvement of numerous participants. It is a costly approach.

4. CLASSIFICATION OF SOFTWARE DEFECT ESTIMATION TECHNIQUES

This section provides an in-depth overview of each software defect estimation technique, namely historical data analysis, expert judgment, function point analysis, software metrics, and machine learning approaches. Each technique is described in terms of its underlying principles, process, and key components and shown in Figure 2.

4.1. Historical Data Analysis

Historical data analysis is a widely used technique for estimating software defects based on past project data. It involves analyzing historical defect data, such as the number of defects found during different development phases, to forecast the number of defects that may occur in future project [24].

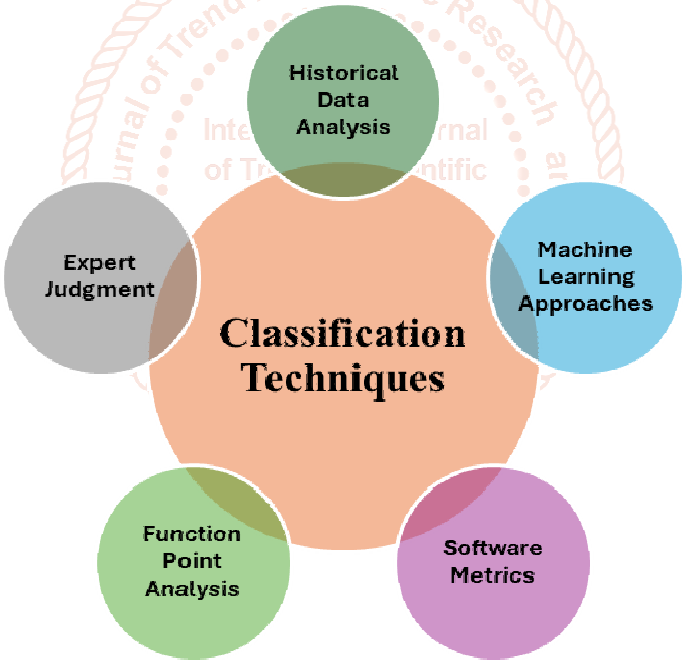

Figure 2. Classification techniques

The technique leverages the assumption that the historical defect patterns are likely to repeat in similar projects. The process typically involves collecting and organizing historical defect data, performing statistical analysis, and using various models or algorithms to extrapolate defect estimates. One of the main strengths of historical data analysis is its reliance on empirical evidence. After analyzing actual project data, this technique provides a realistic and context-specific estimation of defects. It considers the unique characteristics and patterns observed in past projects, enabling project managers to gain insights into the defect-prone areas and allocate resources accordingly [25]. Moreover, this technique has certain limitations. It assumes that the historical defect patterns will remain consistent in future projects, which may not always hold true, especially if there are significant changes in project scope, technology, or team composition [26]. The accuracy of the estimates heavily relies on the availability and quality of historical data, and if the data is limited or of poor quality, it may lead to inaccurate predictions. Additionally, historical data analysis may struggle to account for external factors that could impact defect occurrence, such as changes in industry standards or evolving user expectations.

4.2. Expert Judgment

Expert judgment relies on the knowledge and experience of subject matter experts, such as project managers, developers, testers, and domain experts, to estimate software defects. These experts use their expertise, domain knowledge, and intuition to assess the potential defects that may arise in a software project [27]. The estimation process involves experts providing their opinions, judgments, and assessments based on their understanding of the project, its complexity, and their past experiences. Expert judgment is applicable in situations where there is a lack of historical data or when there is a need to capture qualitative insights and expert knowledge. It is commonly used in early project phases, where data may be scarce or unreliable. One of the primary strengths of expert judgment is its ability to incorporate human expertise and domain knowledge. This technique allows for subjective factors and tacit knowledge to be considered, which may not be captured by quantitative methods. Expert judgment is also a flexible technique that can be applied in situations where historical data or specific metrics may be lacking [28].

4.3. Function Point Analysis

Function point analysis is a technique that estimates software defects based on the functional complexity of the software system. It quantifies the functionality provided by the software and assigns weights to different functional components based on their complexity [29]. The estimation process involves identifying and categorizing the functions of the software, determining their complexity levels, and applying predefined weights to calculate the function points. These function points are then used to estimate the number of defects based on historical defect rates per function point. Function point analysis is applicable in projects where the functional requirements are well-documented and can be quantified. It is commonly used in projects following a structured software development methodology, such as waterfall or iterative approaches. This technique is particularly beneficial for projects with a strong emphasis on functionality, where the estimation of defects is closely tied to the complexity and scope of the software's functionality [30]. Function point analysis provides a structured and standardized approach to estimating defects by considering the complexity of the software's functionality. It focuses on the user's perspective and provides a measure of the software's functionality, which help in assessing the potential defect-prone areas. This technique is independent of the programming language or technology used, making it applicable across different software systems [31] Function point analysis relies on assumptions about the relationship between the complexity of software functions and the occurrence of defects. These assumptions may not hold true in all cases, especially in projects with unique characteristics or non-standard functional requirements. Which is one of the limitations of this technique.

4.4. Software Metrics

Software metrics-based techniques involve the measurement and analysis of various software attributes, such as code complexity, code churn, code coverage, and defect density. These metrics are used to estimate software defects by establishing relationships between the measured metrics and the occurrence of defects [31]. The estimation process typically involves collecting software metrics throughout the development process, performing statistical analysis, and applying models or algorithms to predict defect counts based on the observed metrics. They are commonly used in projects that prioritize data-driven decision-making and have established measurement processes in place. These techniques are particularly beneficial in projects with a focus on continuous improvement and process optimization, as they provide insights into the relationship between software attributes and defect occurrence [32].

Software metrics-based techniques provide a quantitative approach to defect estimation, leveraging objective measures of software attributes. Software metrics can provide early warning signs of potential defects, allowing for proactive defect prevention and mitigation [33]. Moreover, these techniques can be used to monitor and track the progress of defect management activities throughout the software development life cycle. Although these techniques require the availability and collection of relevant software metrics, which may not be feasible in all projects, particularly those with resource or time constraints. The selection and interpretation of metrics require expertise and domain knowledge to ensure their relevance and accuracy. Additionally, the relationship between metrics and defects may vary across different projects or domains, making it challenging to develop universally applicable models or algorithms [34].

4.5. Machine Learning Approaches

Machine learning approaches leverage algorithms and models to automatically learn patterns and relationships from historical data and use them to estimate software defects [35]. These techniques involve training a machine learning model using historical data that captures the relationship between various factors and the occurrence of defects. The trained model is then used to predict the number of defects in new projects based on the input data.

These approaches are often employed in projects with a data-driven culture and a focus on predictive analytics. Machine learning can be particularly useful when combined with other techniques or when used in combination with human expert judgment to augment the accuracy and reliability of defect estimation [36]. Machine learning approaches have the potential to capture complex patterns and relationships in large volumes of data, making them suitable for analyzing and estimating software defects. These techniques can handle a wide range of input data, including various software metrics, historical defect data, and project-specific attributes. Machine learning models can adapt and improve over time as new data becomes available, enhancing their accuracy and effectiveness [37]. These approaches can also provide insights into the factors that contribute most significantly to defect occurrence, enabling focused defect prevention efforts. Moreover, the complexity and interpretability of machine learning models can make them less transparent, making it difficult to understand the reasons behind the predictions. Additionally, these techniques require expertise in machine learning algorithms and data analysis, which may not be readily available in all software development teams.

Table2: Comparison of various machine learning approached

Components	Bayesian Belief Network (BBN)	Neural Network (NN)	Fuzzy Logic	Support Vector Machine (SVM)	Expectation-Maximization (EM)	Case-Based Reasoning (CBR)
Time	Moderate	High	High	Moderate	High	Low
Requirements for data	High	High	Low	Low	High	High
Binary Classification	No	Yes	Yes	Yes	Yes	Yes
Multi-class classification	Yes	No	Yes	No	No	Yes
Knowledge expert	Low	High	Moderate	Low	Low	High

In conclusion, each software defect estimation technique offers distinct advantages and has its own set of limitations. Historical data analysis leverages past project data to provide context-specific estimates, while expert judgment incorporates human expertise and domain knowledge. Function point analysis quantifies software functionality to estimate defects, and software metrics-based techniques rely on objective measures to predict defect counts. Machine learning approaches utilize algorithms and models to learn patterns from data and make predictions. The choice of technique should be based on project characteristics, data availability, expertise, and the specific requirements of the software development context. By understanding the strengths, weaknesses, and applicability of each technique, practitioners can make informed decisions when selecting the most suitable software defect estimation technique for their projects Table 3.

Table 3. Comparative Analysis of Software Defect Estimation Techniques - Transparency and Interpretability

Technique	Transparency	Interpretability
Historical Data Analysis	Can provide transparency if the analysis techniques and methodologies are well-documented and understandable	Interpretability may vary based on the complexity of statistical models or analysis techniques used
Expert Judgment	Lack of transparency as it relies on subjective opinions and expert insights	Interpretability may vary based on the explanations and reasoning provided by experts
Function Point Analysis	Transparent as it follows standardized rules and formulas for assessing functional complexity	Interpretability may vary based on the understanding and documentation of function classification methods
Software Metrics	Can provide transparency if the metrics and analysis methodologies are well-documented and easily understandable	Interpretability may vary based on the relevance and clarity of the selected software metrics
Machine Learning	Less transparent and interpretability may be challenging, especially with complex models like deep learning algorithms	Interpretability can be enhanced by using techniques such as feature importance analysis and model explainability methods

Moreover, comparative analysis of software defect estimation techniques in terms of accuracy and precision is important for optimizing resource allocation, informed decision-making, improving software quality, and effective risk management. It enables project managers to allocate resources effectively, set realistic goals, and

make informed decisions. Accurate estimation supports quality improvement and facilitates continuous process enhancement. Additionally, benchmarking and risk management benefit from reliable estimates, helping organizations stay proactive and minimize defects' impact. We have shown in Table 4.

Table 4. Comparative Analysis of Software Defect Estimation Techniques – accuracy and Precision

Technique	Accuracy	Precision
Historical Data Analysis	Relies on historical trends; accuracy depends on the representativeness and quality of data	Can provide consistent estimates if historical patterns persist; subject to variations and biases
Expert Judgment	Subjective and may vary among experts; accuracy and precision depend on the expertise and knowledge of individuals	Potential for inconsistencies and biases
Function Point Analysis	Can achieve accuracy and precision based on standardized rules and well-defined functional complexity assessment	Accuracy and precision depend on the accuracy of function classification and weighting
Software Metrics	Can provide accurate estimates based on quantitative measures and data analysis	Accuracy and precision depend on the relevance and quality of the selected software metrics
Machine Learning	Accuracy and precision depend on the quality and representativeness of training data and the chosen machine learning algorithms	Can achieve high accuracy and precision with well-trained models

5. CONCLUSION AND FUTURE SCOPE

This study has provided a comprehensive review and comparison of software defect estimation techniques, including historical data analysis, expert judgment, function point analysis, software metrics, and machine learning approaches. Through the comparative analysis, the strengths, weaknesses, and applicability of each technique have been evaluated. The applicability assessment has discussed the suitability of these techniques in different software development. The key findings from this research survey paper highlight the importance of selecting appropriate defect estimation techniques based on project requirements. It is crucial to consider the specific characteristics of the software development project, assess the availability and quality of data, evaluate the expertise and resources available, and overall evaluate the suitability of the techniques. The selection of a defect estimation technique should be a thoughtful and deliberate process, considering the unique needs and context of each project. For researchers, this survey paper present opportunities for further exploration and advancement in the field of software defect estimation. Future research directions include the integration of machine learning with traditional techniques, handling uncertainty and risk, exploring new metrics and indicators, validation and generalization of techniques, developing real-time and dynamic estimation models, and incorporating non-technical factors. These areas of research can contribute to improving the accuracy, effectiveness, and applicability of defect estimation techniques, ultimately enhancing software quality and project success.

REFERENCES

[1] T. Hall, S. Beecham, D. Bowes, et al., "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012. doi:10.1109/TSE.2011.103.

[2] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4626–4636, 2011. doi:10.1016/j.eswa.2010.10.024.

[3] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012. doi:10.1016/j.infsof.2011.09.007.

[4] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7346–7354, 2009. doi:10.1016/j.eswa.2008.10.027.

[5] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," in *Proc. 5th ACM/IEEE Int. Symp. Empirical Software Engineering (ISESE)*, 2006, pp. 18–27. doi:10.1145/1159733.1159739.

[6] Y. Jiang, B. Cukic, T. Menzies, and N. Bartlow, "Comparing design and code metrics for software quality prediction," in *Proc. 30th*

Int. Conf. Software Engineering (ICSE), 2008, pp. 11–18. doi:10.1145/1370788.1370793.

[7] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003. doi:10.1109/TSE.2003.1191795.

[8] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Transactions on Software Engineering*, vol. 22, no. 12, pp. 886–894, 1996. doi:10.1109/32.553637.

[9] T. Menzies, J. S. Di Stefano, and M. Chapman, "Learning early lifecycle IV&V quality indicators," in *Proc. Int. Software Metrics Symposium*, 2003, pp. 88–96. doi:10.1109/METRIC.2003.1232458.

[10] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?" in *Proc. 2nd Int. Conf. Software Engineering*, 1976, pp. 61–68.

[11] T. K. Abdel-Hamid, K. Sengupta, and C. Swett, "The impact of goals on software project management: An experimental investigation," *MIS Quarterly*, vol. 23, no. 4, pp. 531–555, 1999.

[12] T. R. Nair, "Impact analysis of allocation of resources by project manager on success of software projects," *arXiv preprint arXiv:1407.5319*, 2014.

[13] B. Boehm, "Future challenges for software data collection and analysis," in *Proc. PROMISE*, 2009. doi:10.1145/1540438.1540440.

[14] Y. Jiang, B. Cukic, and T. Menzies, "Fault prediction using early lifecycle data," in *Proc. 18th IEEE Int. Symp. Software Reliability Engineering (ISSRE)*, 2007, pp. 237–246.

[15] P. K. Suri and P. Ranjan, "Comparative analysis of software effort estimation techniques," *International Journal of Computer Applications*, vol. 48, no. 21, pp. 12–19, 2012.

[16] R. M. Waghmode, L. V. Patil, and S. D. Joshi, "A collective study of PCA and neural network based on COCOMO for software cost estimation," *International Journal of Computer Applications*, vol. 74, no. 8, pp. 25–30, 2013.

[17] A. Kaushik, A. Chauhan, D. Mittal, and S. Gupta, "COCOMO estimates using neural networks," *International Journal of Intelligent Systems and Applications*, vol. 4, no. 6, pp. 22–28, 2012.

[18] A. Gupta, N. Mishra, and D. S. Kushwaha, "Rule based test case reduction technique using decision table," in *Proc. IEEE Int. Advance Computing Conf. (IACC)*, 2014, pp. 1398–1405.

[19] K. K. Lilja, K. Laakso, and J. Palomäki, "Using the Delphi method," in *Proc. PICMET*, 2011, pp. 1–10.

[20] K. Nielsen, "Software estimation using a combination of techniques," 2013. *(Verify publication source.)*

[21] H. Karna and S. Gotovac, "Modeling expert effort estimation of software projects," in *Proc. 22nd Int. Conf. Software, Telecommunications and Computer Networks (SoftCOM)*, 2014, pp. 356–360.

[22] M. Madheswaran and D. Sivakumar, "Enhancement of prediction accuracy in COCOMO model for software project using neural network," in *Proc. 5th Int. Conf. Computing, Communications and Networking Technologies (ICCCNT)*, 2014, pp. 1–5.

[23] M. Aljohani and M. Qureshi, "Comparative study of software estimation techniques," *International Journal of Software Engineering and Applications*, vol. 8, 2017.

[24] S. Amasaki, "Cross-version defect prediction: Use historical data, cross-project data, or both?" *Empirical Software Engineering*, vol. 25, pp. 1573–1595, 2020.

[25] Z. He, F. Shu, Y. Yang, R. Li, and N. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[26] Q. Song, Z. Jia, M. Shepperd, et al., "A general software defect-proneness prediction framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, 2011.

[27] M. Kläs, H. Nakao, F. Elberzhager, and J. Münch, "Support planning and controlling of early quality assurance by combining expert judgment and defect data: A case study," *Empirical Software Engineering*, vol. 15, no. 4, pp. 423–454, 2010.

[28] Y. Zhou, N. Fenton, and M. Neil, "Bayesian network approach to multinomial parameter learning using data and expert judgments,"

International Journal of Approximate Reasoning, vol. 55, no. 5, pp. 1252–1268, 2014.

[29] M. J. Basavaraj and K. C. Shet, "Software estimation using function point analysis: Difficulties and research challenges," in *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, 2007, pp. 111–116.

[30] M. A. Parthasarathy, *Practical Software Estimation: Function Point Methods for Insourced and Outsourced Projects*. New Delhi, India: Pearson Education India, 2007.

[31] S. Furey, "Why we should use function points," *IEEE Software*, vol. 14, no. 2, pp. 28–30, 1997.

[32] S. Karim, H. L. H. S. Warnars, F. L. Gaol, et al., "Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset," in *Proc. IEEE Int. Conf. Cybernetics and Computational Intelligence*, 2017, pp. 19–23.

[33] K. Punitha and S. Chitra, "Software defect prediction using software metrics: A survey," in *Proc. Int. Conf. Information Communication and Embedded Systems (ICICES)*, 2013, pp. 555–558.

[34] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, 1999.

[35] T. Hall and D. Bowes, "The state of machine learning methodology in software fault prediction," in *Proc. 11th Int. Conf. Machine Learning and Applications (ICMLA)*, 2012, pp. 308–313.

[36] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *International Journal of Advanced Computer Science and Applications*, vol. 9, 2018.

[37] K. Anwar, J. Siddiqui, and S. S. Sohail, "Machine learning-based book recommender system: A survey and new perspectives," *International Journal of Intelligent Information and Database Systems*, vol. 13, no. 2–4, pp. 231–248, 2020. doi:10.1504/IJIIDS.2020.109457.