IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

EVALUATING THE EFFECTIVENESS OF SOFTWARE TESTING DEFECT PREDICTION METHODS

[1]M.MANI MEKALAI, [2]DR.S.VYDEHI

[1]Research scholar, [2]Associate Professor, Department of Computer Science,

[1,2]Dr.S.N.S.Rajalakshmi College of Arts and Science,

[1,2]Chinnavedampatti Post, Coimbatore, Tamilnadu, India.

Abstract- This literature survey explores the significance and methods of effectively utilizing historical data in software testing defect prediction. With the growing complexity of software systems, predicting and preventing defects has become paramount in ensuring software quality. Leveraging historical data, such as past defects and testing outcomes, can provide valuable insights into potential vulnerabilities and areas of improvement. The abstract delves into various approaches and techniques employed in harnessing historical data for defect prediction, including machine learning algorithms, statistical analysis, and data mining methodologies. Furthermore, it investigates the challenges and limitations associated with utilizing historical data in software testing, such as data quality issues, feature selection, and model validation. By synthesizing existing research findings and methodologies, this literature survey aims to provide a comprehensive understanding of how historical data can be effectively leveraged to enhance software testing defect prediction strategies, ultimately leading to improved software quality and reliability.

Keywords: Software testing, Defect prediction, Software maintenance, data analysis, Regression analysis, Predictive modeling, Feature selection, Data mining.

1. Introduction

Software testing defect prediction is a crucial aspect of software quality assurance, aiming to identify potential defects in software systems before the manifest in production environments. Historical data, accumulated from past software development projects, holds significant promise for enhancing the accuracy and effectiveness of defect prediction models. Leveraging this historical data enables organizations to anticipate and mitigate potential risks, allocate resources efficiently, and ultimately deliver higher-quality software products. The utilization of historical data in software testing defect prediction is a multifaceted endeavor, encompassing various techniques, methodologies, and tools. The literature on this subject presents a rich tapestry of research efforts, empirical studies, and practical applications aimed at harnessing the power of historical data to improve defect prediction accuracy and reliability. In figure 1 the types of software testing that have been described.

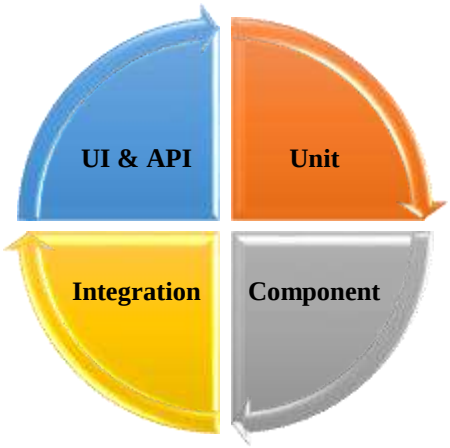

Figure 1.Types of Software Testing

One of the fundamental approaches in utilizing historical data for defect prediction is the application of machine learning algorithms. These algorithms analyze historical data sets, extracting patterns, trends, and relationships between various software metrics and defect occurrences. By training predictive models on this data, organizations can identify factors indicative of potential defects, enabling proactive intervention during the software development lifecycle. Furthermore, researchers have explored the integration of diverse data sources to enhance the predictive capabilities of defect prediction models. Beyond traditional software metrics, such as lines of code or complexity measures, additional data sources, including version control repositories, issue tracking systems, and developer collaboration patterns, offer valuable insights into software quality and defect proneness. In addition to predictive modeling, the literature also delves into the assessment of the effectiveness and efficiency of utilizing historical data in defect prediction.

Empirical studies evaluate the performance of various prediction models in real-world software development contexts, shedding light on the strengths, limitations, and practical considerations associated with leveraging historical data. Moreover, advancements in data mining, statistical analysis and artificial intelligence have spurred innovative approaches to harnessing historical data for defect prediction. Techniques such as ensemble learning, deep learning, and hybrid models combine multiple sources of historical data and employ sophisticated algorithms to improve prediction accuracy and robustness. In this survey, it explores the diverse landscape of research and practices surrounding the effective utilization of historical data in software testing defect prediction. By synthesizing existing knowledge and identifying emerging trends, this survey aims to provide insights that can inform future research directions and guide practitioners in leveraging historical data to enhance software quality and reliability.

2. Literature Survey

1. A. M. Khan (2021) et.al proposed AUTILE Framework: An AUTOSAR Driven Agile Development Methodology to Reduce Automotive Software Defects. The AUTOSAR Driven Agile Development (AUTILE) Framework is introduced as an innovative methodology aimed at mitigating defects and minimizing their severity in automotive software development. The conventional approach to developing automotive software, often requiring complete rewrites for hardware modifications or the addition of new features, poses significant challenges. The evolving landscape of automotive technology, characterized by megatrends like connectivity, electrification, and autonomous driving, intensifies the demand for complex software functionalities. The lack of a standardized approach exacerbates quality issues when integrating additional features into no standardized automotive software. In response to these challenges, AUTILE advocates for the utilization of the automotive open system architecture standard as the foundational framework, promoting modular and open software architecture. This approach aligns with the need for standardized solutions in the industry. Utilizing agile approaches, AUTILE incorporates modular architecture into their software development process to maximize its benefits. Despite the potential advantages of agile methodologies, their widespread adoption in automotive software development remains limited. The framework addresses the enablers and barriers associated with implementing agile methodologies in this context. Practical application of the AUTILE Framework to automotive software projects validates its effectiveness in defect reduction. By following the recommended steps, this methodology showcases its potential to enhance the quality and efficiency of automotive software development, marking a significant stride towards addressing industry challenges.

2. H. GU (2021) et.al proposed Specification-Driven Conformance Checking for Virtual/Silicon Devices Using Mutation Testing. Modern software systems, including both system and application software, are increasingly built upon virtualized software platforms. These platforms may be designed to operate on virtual machines or have the flexibility to transition to physical machines when required. The essential aspect in either scenario is that the devices, whether virtual or silicon-based, within the target system must adhere strictly to the

specified standards upon which the software systems are developed. Any deviation from these specifications can potentially lead to catastrophic failures in the software. This research introduces a novel mutation-based framework designed for the efficient and effective conformance checking between virtual/silicon device implementations and their specified requirements. The framework employs mutation operators to automatically instrument device specifications with weak mutant-killing constraints, capturing potential erroneous behaviours'. A cooperative symbolic execution approach is used to automate test case development and compliance testing for virtual and silicon devices in order to assure thorough validation. The approach properly determines the degree of validation and finds discrepancies between device specifications and implementations by symbolically executing thetraces of virtual or silicon devices combined with instrumented specifications obtained from the cooperative execution.The efficacy of this method in validating conformance for both silicon and virtual devices is proven by extensive tests performed on two industrial network adapters and their virtual devices, indicating that it has the potential to enhance the reliability and resilience of modern software systems. Extensive tests conducted on two industrial network adapters and their virtual devices demonstrate the effectiveness of this method in verifying conformity for both silicon and virtual devices, confirming its potential to improve the resilience and dependability of contemporary software systems.

3. A. Núñez (2021) et.al proposed TEA-Cloud: A Formal Framework for Testing Cloud Computing Systems. Validating cloud systems poses challenges due to their scale, concurrent user demands, and the complexities introduced by virtualization. Conventional testing methods struggle to address these issues effectively. This article introduces the TEA-Cloud framework, offering a novel approach by integrating simulation and testing methodologies for comprehensive validation of cloud system designs. TEA-Cloud covers both functional and non-functional aspects, including performance and cost considerations. The framework facilitates the modeling of both software and hardware components, enabling automated testing to validate cloud system integrity in a cost-effective manner. TEA-Cloud employs metamorphic testing to address the absence of a definitive oracle for verifying observed behaviors during testing. This approach relies on metamorphic relations (MRs) and introduces three MR families targeting critical aspects such as performance, resource provisioning, and cost. Through an empirical research utilizing fault seeding and ten MRs for various cloud configurations, TEA-Cloud demonstrated promising results by successfully identifying all seeded faults. The framework's capability to navigate complex cloud environments, simulate real-world scenarios, and uncover potential vulnerabilities positions TEA-Cloud as a valuable tool for enhancing the validation processes associated with cloud system development.

4. C. -A. Sun (2021) et.al proposed METRIC${+}$+: A Metamorphic Relation Identification Technique Based on Input plus Output Domains. Metamorphic testing, renowned for mitigating the oracle problem in software testing, operates by verifying identified metamorphic relations (MRs) across multiple the way a software system is executed. The crucial task of MR identification prompted the development of METRIC (METamorphic Relation Identification based on Category-choice framework). However, METRIC primarily emphasizes the input domain, neglecting the output domain, which hampers its effectiveness. In response, it extended METRIC into METRIC+ by incorporating output domain information for enhanced MR identification. METRIC+ was implemented as a tool, and two rounds of experiments involving real-life specifications were conducted. The results affirm METRIC+'s high effectiveness and efficiency in MR identification. To further assess METRIC+'s performance, it conducted experiments comparing its fault detection capability with that of μMT, another MR identification technique. The comparative results validate METRIC+'s MRs as highly effective in fault detection. The extension of METRIC into METRIC+ addresses the input-output domain imbalance, fortifying its applicability and demonstrating its superior effectiveness in identifying robust metamorphic relations. The development of METRIC+ contributes significantly to advancing the field of metamorphic testing, providing testers with a comprehensive tool that incorporates both input and output domain considerations for more accurate and efficient MR identification.

5. N. Aljawabrah (2021) et.al proposed Automated Recovery and Visualization of Test-to-Code Traceability (TCT) Links: An Evaluation. Because of unclear documentation, maintaining traceability links between unit tests and code in the software development process often necessitates manual dependency identification. This manual effort during the comprehension process results in a significant time and resource investment. While various methods exist for inferring these links based

on different phenomena, it tends to generate diverse sets of traceability links. By combining a number of traceability recovery techniques, this research expands on earlier research on traceability link recovery and visualization. These methods automatically retrieve links and visualize them, providing developers with an overview of links inferred by various recovery techniques. The approach aims to facilitate the selection of appropriate relations for analysis. The usability research findings show that the visualization model that is being given facilitates Test-to-Code Traceability (TCT) link browsing, comprehension, and maintenance in a system in an efficient manner. Additionally, visualizing TCT links from multiple sources proves to be more advantageous than visualizing links from a single source, contributing to a more comprehensive understanding of the interconnected relationships between tests and code.

6. Y. Huang (2021) et.al proposed A Learn-to-Rank Method for Model-Based Regression Test Case Prioritization. Regression testing is a pivotal component of software maintenance, involving the retesting of modified software to identify potential introduction of new faults. Despite its significance a high fault detection rate in regression testing demands substantial effort. To tackle this challenge, the adoption of test case prioritization techniques becomes imperative, allowing the adjustment of test case execution order to enhance fault detection. Existing approaches in model-based regression test case prioritization often focus on single aspects of model-related information derived from previously executed test cases. In this research, it propose a novel learn-to-rank technique aimed at prioritizing test cases by integrating multidimensional features extracted from the Extended Finite State Machine (EFSM) under test, thus improving fault detection rates. It method employs the random forest algorithm to amalgamate various heuristic prioritization methods. It noticed promising findings through comprehensive trials assessing the performance of the suggested technique, especially in terms of Average Percentage Fault Detected (APFD). The mean APFD value achieved by it method is 0.884 across five subject EFSMs, representing a noteworthy 33.9% improvement compared to existing methods. This innovative approach showcases the efficacy of leveraging multidimensional features in EFSMs for regression test case prioritization, contributing to more efficient fault detection in software maintenance.

7. J. Caba (2021) et.al proposed Towards Test-Driven Development for FPGA-Based Modules across Abstraction Levels. High-Level Synthesis (HLS) tools serve as invaluable aids for engineers grappling with the intricacies of constructing heterogeneous embedded systems that leverage reconfigurable technology. While HLS facilitates the integration of well-established software industry practices such as Test-Driven Development (TDD) into the development flow of custom hardware components, its support for verification activities remains constrained, typically focusing on the initial design stages. In order to expedite component on-board verification, this research presents a novel hardware testing framework that enables TDD on reconfigurable hardware and implements the Unit Testing Paradigm. The proposed solution encompasses a hardware/software introspection infrastructure, facilitating the verification of system modules at various stages and across multiple abstraction levels without necessitating additional effort or component redesign. The CHStone Benchmark uses the framework, which was created for the Xilinx ZynQ FPGA-SoC architecture, to validate C-kernels. Integration into the Xilinx Vivado design flow is seamless, supported by automatic generation scripts tailored for this purpose. Experimental results highlight a significant acceleration in verification time and expose disparities between the on-board latency measured by it framework and the co-simulation estimations provided by Xilinx tools. This research contributes to enhancing the efficiency of HLS tools by addressing limitations in verification support, ultimately advancing the reliability and accuracy of reconfigurable hardware design.

8. Z. Q. Zhou et.al proposed Metamorphic Robustness Testing: Exposing Hidden Defects in Citation Statistics and Journal Impact Factors. It proposed robustness testing approach addresses software systems handling extensive data volumes, utilizing metamorphic relations to assess output reliability without a concrete test oracle. It applied this method to scrutinize two prominent citation databases, Scopus and the Web of Science, uncovering an unexpected discovery related to hyphens in research titles impacting citation counts. This revelation underscores the vulnerability of citation database systems in managing hyphenated research titles, a finding with broad applicability across diverse fields, including chemistry. Notably, it examined journals such as IEEE Transactions on Software Engineering (TSE) and ACM Transactions on Software Engineering and Methodology, and found a significant inverse relationship between the percentage of hyphenated research titles and journal impact factor (JIF). Surprisingly, higher-ranked JIF journals, including TSE, exhibited a lower percentage of papers with hyphenated titles,

challenging the widely-held belief that citation counts and JIFs reliably measure research and journal impact. It research prompts a reconsideration of the perceived accuracy of these metrics, demonstrating their susceptibility to distortion due to the presence of hyphens in research titles, emphasizing the need for a more nuanced evaluation of scholarly impact in the domain of software engineering.

9. A. Agrawal (2021) et.al proposed How to "DODGE" Complex Software Analytics. Enhancing the efficiency of machine learning techniques applied to software engineering tasks is a key focus, and one avenue for improvement lays in hyperparameter optimization automatic tools that seek optimal settings for a learner's control parameters. Within this context, it present the DODGE (Dynamic Optimization of Decision Tree enGinE) approach, specifically DODGE (E), a tuning tool designed to expedite the hyperparameter optimization process. Notably, DODGE (E) addresses the issue of unnecessary slowness encountered in traditional hyperparameter optimization, particularly when optimizers spend time exploring "redundant tunings." These redundancies refer to pairs of tunings that lead to indistinguishable results. Operating orders of magnitude quicker than previous state-of-the-art methods, DODGE (E) achieves a tremendous acceleration by deliberately avoiding such superfluous tunings. Despite its accelerated pace, DODGE (E) not only expedites the optimization process but also outperforms previous methods by generating learners with more accurate predictions. The focus on eliminating redundant tunings underscores the significance of optimization efficiency in the context of hyperparameter tuning, making a significant contribution to the complicated software analytics discipline. The results highlight DODGE (E) as a promising tool for practitioners seeking improved performance and precision in machine learning applications within the realm of software engineering.

10. G. K. Rajbahadur (2021) et.al proposed Impact of Discretization Noise of the Dependent Variable on Machine Learning Classifiers in Software Engineering. In the realm of research, the common practice of discretizing a continuous dependent variable into two target classes, often through the introduction of an artificial threshold like the median, presents a potential challenge. This discretization, while facilitating analysis, may introduce noise, specifically discretization noise, arising from the ambiguous loyalty of data points near the artificial threshold. Regretfully, current research does not provide a clear guide on how discretization noise affects classifiers and how to deal with it. This research addresses this gap by proposing a comprehensive framework to systematically assess the impact of discretization noise on classifiers. The framework evaluates its effects on various performance measures and the interpretation of classifiers, offering valuable insights for researchers and practitioners. Through a case research involving seven software engineering datasets, the findings underscore the nuanced impact of discretization noise on classifier performance across different datasets. Notably, the research reveals that while discretization noise affects various performance measures disparately, the interpretation of classifiers, on the whole, is influenced. However, crucially, the top three most important features remain unaffected by discretization noise. Consequently, the research recommends the adoption of this framework to enable practitioners and researchers to discern and quantify the impact of discretization noise on classifier performance. Furthermore, it advocates for the strategic elimination of precise amounts of discretization noise from datasets to mitigate potential adverse effects. This approach ensures a more informed and nuanced understanding of the impact of discretization noise, thereby enhancing the robustness and reliability of classifiers in practical applications.

11. M. W. Call (2021) et.al proposed Gamifying Software Engineering Tools to Motivate Computer Science Students to Start and Finish Programming Assignments Earlier. This gamification strategy employs a custom Leaderboards plugin within the Moodle Learning Management System (LMS) to incentivize computer science students in a data structures course (N = 50) to initiate and complete challenging programming assignments (PAs) promptly. The notable outcomes reveal a positive impact, with students demonstrating increased motivation to complete assignments ahead of schedule. Additionally, there was a notable rise in code commits to Github, adherence to version control best practices, and active participation in the class Q&A forum, showcasing a multifaceted improvement in students' engagement and collaboration. The application of self-determination theory to these findings indicates that integrating team-based leaderboards with an automated unit testing system, offering immediate feedback upon task accomplishment, can effectively cater to students' autonomy, competency, and relatedness needs. However, this research acknowledges limitations, particularly the evaluation occurring within a single semester disrupted by the COVID-19

emergency remote learning transition. Future endeavors aim to validate the Leaderboards' efficacy across diverse classes, explore online development environments with auto-grading for nuanced tracking of students' progress, and investigate alternative incentivization methods. In summary, the gamification approach not only introduced students to essential software engineering tools but also fostered excitement in overcoming unit test challenges, fostering increased dedication to the course.

12. G. Jahangirova (2021) et.al proposed An Empirical Validation of Oracle Improvement. It proposed methodology introduces a human-in-the-loop approach aimed at enhancing oracle quality and assesses its impact on developers' ability to create more effective oracles. In conducting two comprehensive human studies involving a total of 68 participants, it focused on both oracle assessment and improvement. The results of it research reveal a significant challenge in manually assessing assertion oracles, with developers demonstrating mere 29% accuracy in distinguishing false positives, false negatives, or the absence of both. This underscores the need for automated tools to detect and address such deficiencies, providing valuable assistance to developers. To address this gap, it present OASIs (Oracle Assessment and Improvement), a tool designed to elevate the quality of assertions. In the improvement research, participants utilizing OASIs achieved notable improvements, with 33% attaining full correctness and 67% reaching partial correctness. In contrast, participants without the tool achieved lower levels of correctness, with only 21% reaching full correctness and 43% attaining partial correctness. These findings underscore the efficacy of it human-in-the-loop approach, demonstrating its potential to significantly enhance the accuracy and quality of assertion oracles in developing software.

13. R. Verdecchia (2021) et.al proposed Know You Neighbor: Fast Static Prediction of Test Flakiness. Proposing an innovative approach under the motto "knows your neighbor," it introduce FLAST, a method for predicting flaky tests by leveraging the similarity in test code. FLAST effectively identifies test methods likely to expose flakiness based on their proximity to known flaky tests. Notably, FLAST demonstrates its efficacy as a predictor with minimal time and storage overhead, offering a valuable solution in the realm of test flakiness. A distinctive feature of FLAST is its ability to detect flaky tests fully automatically, even before their execution, preventing wasted testing efforts and minimizing delays in code velocity. While the research focus on test flakiness is relatively recent, a qualitative comparison of existing approaches positions FLAST as a promising avenue for addressing this challenge. The potential for FLAST extends beyond its current implementation; future work could explore embedding FLAST within Integrated Development Environments (IDEs) or Continuous Integration (CI) platforms. This integration could guide dynamic tools or automatically alert developers about the risk of introducing flaky tests during the test case or method creation process. Importantly, FLAST is not proposed as an alternative to existing dynamic solutions; rather, its synergy with dynamic approaches is emphasized. FLAST's static analysis efficiently filters out potential flaky tests, allowing dynamic methods like DeFlaker or Rerun to focus on a reduced set, optimizing resource utilization. The collaborative integration of FLAST and dynamic approaches presents a compelling objective for future research endeavors in the domain of software testing.

14. Y. Wang (2021) et.al proposed Research on the Chamber Pressure Test Method of Small Caliber Weapons Based on a Double-Layer and Double-Grid Structure Strain Tester. Addressing the challenging issue of chamber pressure testing in internal ballistics, a novel method introduces a double-layer and double-grid structure strain tester for small caliber weapons. With its streamlined design and capacity to record the whole pressure test curve without jeopardizing the barrel structure, this novel method streamlines the conventional chamber pressure test procedure. Unlike conventional methods, this novel technique does not require destructive measures. The proposed method achieves high test accuracy at a low cost, ensuring reliability and an extended operational lifespan, coupled with excellent electromagnetic compatibility (EMC). The basic idea is to use the double-layer and double-grid strain tester to measure the strain on the outer barrel wall. This allows the chamber pressure to be calculated using thick-walled cylinder theory. The fourth-order Runge-Kutta algorithm was used to build a simulation model of chamber pressure after the mechanism was examined. Strain tester sensitivity was determined through system calibration. Comparative analysis with lead wire electronic manometer tests and simulation modeling methods demonstrated the method's accuracy and feasibility. Both simulations and experiments affirmed the new test method's effectiveness, highlighting its robust electromagnetic compatibility. This approach presents a significant advancement in chamber pressure testing, offering an efficient, cost-effective, and reliable solution for small caliber weapons.

15. S. Jiang (2021) et.al proposed An Integration Test Order Strategy to Consider Control Coupling. Integration testing stands as a pivotal phase in software testing, ensuring the seamless collaboration of individual components within a system. The primary focus of current methods for assessing stubbing costs in class integration test orders is on direct links between classes, like inheritance, aggregation, and association. However, the often neglect the influence of interclass indirect relationships introduced by control coupling. Control coupling can substantially impact both test orders and stubbing costs. In this research, it present an innovative integration test order strategy that incorporates control coupling considerations. It introduce the concept of a transitive relationship to characterize this interclass dependency and propose a novel measurement method to gauge the complexity of control coupling, specifically the intricacy of stubs generated for a transitive relationship. It evaluation of this integration test order strategy spans 10 programs of varying scales. The findings demonstrate that accounting for transitive relationships during the generation of class integration test orders significantly diminishes stubbing costs across most programs. Moreover, it proposed strategy yields satisfactory results more expeditiously compared to alternative methods. This research underscores the importance of recognizing and addressing control coupling in integration testing, offering a valuable contribution to the optimization of software testing processes and resource allocation.

3. Proposed Methods, Merits and Demerits

Authors	Proposed Method	Merits	Demerits
A. M. Khan (2021)	AUTILE Framework: An AUTOSAR Driven Agile Development Methodology to Reduce Automotive Software Defects.	AUTILE enhances agility in automotive software development, reducing defects through AUTOSAR-driven methodologies.	Requires specialized AUTOSAR knowledge; potential initial learning curve for development teams.
H. GU (2021)	Specification-Driven Conformance Checking for Virtual/Silicon Devices Using Mutation Testing.	Specification-driven conformance checking enhances accuracy by systematically validating virtual/silicon devices against predefined specifications using mutation testing.	Complexity may increase due to extensive specifications, potentially leading to longer development cycles.
A. Núñez (2021)	TEA-Cloud: A Formal Framework for Testing Cloud Computing Systems.	TEA-Cloud ensures rigorous testing of cloud systems, enhancing reliability and security in cloud computing environments.	Potential complexity may require specialized expertise, potentially leading to resource-intensive implementation challenges.
C. -A. Sun (2021)	METRIC${+} $+: A Metamorphic Relation Identification Technique Based on Input plus Output Domains.	Enhances software testing by robustly identifying metamorphic relations, ensuring comprehensive test coverage.	Potential complexity in implementing and managing input plus output domains for diverse applications.
N. Aljawabrah (2021)	Automated Recovery and Visualization of Test-to-Code Traceability (TCT) Links: An Evaluation.	Automated TCT enhances efficiency, ensuring accurate traceability, streamlining development.	Potential complexity and resource requirements, necessitating skilled implementation for optimal results.

Y. Huang (2021)	A Learn-to-Rank Method for Model-Based Regression Test Case Prioritization.	Enhances regression testing efficiency through prioritization, optimizing resource allocation for improved software reliability.	Complexity may impede straightforward implementation, requiring careful consideration for effective application.
J. Caba (2021)	Towards Test-Driven Development for FPGA-Based Modules across Abstraction Levels.	Enhances reliability through rigorous testing, ensures functionality alignment, and promotes systematic development for FPGA-based modules.	Potential increased initial development time due to testing overhead; requires a comprehensive understanding of testing methodologies.
Z. Q. Zhou	Metamorphic Robustness Testing: Exposing Hidden Defects in Citation Statistics and Journal Impact Factors.	Metamorphic Robustness Testing reveals latent flaws in citation statistics, enhancing the reliability of impact factor assessments.	Implementation complexity may hinder widespread adoption; requires expertise for accurate defect identification.
A. Agrawal (2021)	How to "DODGE" Complex Software Analytics.	DODGE facilitates intricate software analysis, enhancing decision-making; streamlining complex processes for efficient software development.	Requires specialized training; initial implementation may pose a learning curve for development teams.
G. K. Rajbahadur (2021)	Impact of Discretization Noise of the Dependent Variable on Machine Learning Classifiers in Software Engineering.	Discretization noise analysis enhances classifier robustness, aiding accurate model interpretation in software engineering.	Overemphasis may lead to excessive complexity; context-specific effects on diverse software datasets require consideration.
M. W. Call (2021)	Gamifying Software Engineering Tools to Motivate Computer Science Students to Start and Finish Programming Assignments Earlier.	Gamifying software tools enhances student motivation, fostering early engagement and completion of programming assignments.	Potential distraction; must ensure game elements align with educational goals to avoid undermining learning.
G. Jahangirova (2021)	An Empirical Validation of Oracle Improvement.	Oracle Improvement" enhances software reliability; empirical validation ensures enhanced performance and accurate error identification.	Potential resource-intensive process during large-scale software; may face challenges with complex code structures.
R. Verdecchia (2021)	Know You Neighbor: Fast Static Prediction of Test Flakiness.	Know Your Neighbor offers rapid static prediction for test flakiness, enhancing software reliability and efficiency.	Limited adaptability to dynamic testing environments may lead to occasional inaccuracies in flakiness predictions.
Y. Wang (2021)	Research on the	The double-layer, double-	Increased complexity may

	Chamber Pressure Test Method of Small Calibre Weapons Based on a Double-Layer and Double-Grid Structure Strain Tester.	grid strain tester improves small calibre weapon testing precision, enhancing safety and reliability.	raise costs; potential need for specialized training in operating intricate testing equipment.
S. Jiang (2021)	An Integration Test Order Strategy to Consider Control Coupling.	Enhances system reliability; ensures effective communication among integrated components in a controlled manner.	Potential increased complexity; requires meticulous planning to avoid unintended consequences in integration testing.

4. Conclusion

In this research, the survey highlights the significance of leveraging historical data in software testing defect prediction. The findings underscore the potential of historical data to enhance the accuracy and efficiency of defect prediction models, thereby aiding in proactive risk management and resource allocation. Through various techniques such as machine learning algorithms, statistical analysis, and data mining, researchers have demonstrated the feasibility of harnessing historical data to anticipate and mitigate software defects. However, challenges such as data quality, feature selection, and model interpretability remain prevalent and require further investigation. Moreover, the diversity of software projects and contexts necessitates tailored approaches for effective utilization of historical data. Despite these challenges, the literature surveyed provides valuable insights and methodologies for integrating historical data into software testing processes, ultimately facilitating the development of more reliable and robust software systems. Continued research in this domain is crucial to advancing defect prediction techniques and optimizing software quality assurance practices.

Reference

1. A. M. Khan and T. D. Blackburn, "AUTILE Framework: An AUTOSAR Driven Agile Development Methodology to Reduce Automotive Software Defects," in IEEE Systems Journal, vol. 15, no. 3, pp. 3283-3290, Sept. 2021, doi: 10.1109/JSYST.2020.2999587.

2. H. Gu, J. Zhang, M. Chen, T. Wei, L. Lei and F. Xie, "Specification-Driven Conformance Checking for Virtual/Silicon Devices Using Mutation Testing," in IEEE Transactions on Computers, vol. 70, no. 3, pp. 400-413, 1 March 2021, doi: 10.1109/TC.2020.2988906.
3. A. Núñez, P. C. Cañizares, M. Núñez and R. M. Hierons, "TEA-Cloud: A Formal Framework for Testing Cloud Computing Systems," in IEEE Transactions on Reliability, vol. 70, no. 1, pp. 261-284, March 2021, doi: 10.1109/TR.2020.3011512.
4. C. -A. Sun, A. Fu, P. -L. Poon, X. Xie, H. Liu and T. Y. Chen, "METRIC^{+}+: A Metamorphic Relation Identification Technique Based on Input Plus Output Domains," in IEEE Transactions on Software Engineering, vol. 47, no. 9, pp. 1764-1785, 1 Sept. 2021, doi: 10.1109/TSE.2019.2934848.
5. N. Aljawabrah, T. Gergely, S. Misra and L. Fernandez-Sanz, "Automated Recovery and Visualization of Test-to-Code Traceability (TCT) Links: An Evaluation," in IEEE Access, vol. 9, pp. 40111-40123, 2021, doi: 10.1109/ACCESS.2021.3063158.
6. Y. Huang, T. Shu and Z. Ding, "A Learn-to-Rank Method for Model-Based Regression Test Case Prioritization," in IEEE Access, vol. 9, pp. 16365-16382, 2021, doi: 10.1109/ACCESS.2021.3053163.
7. J. Caba, F. Rincón, J. Barba, J. A. De La Torre, J. Dondo and J. C. López, "Towards Test-Driven Development for FPGA-Based Modules Across Abstraction Levels," in IEEE Access, vol. 9, pp. 31581-31594, 2021, doi: 10.1109/ACCESS.2021.3059941.
8. Z. Q. Zhou, T. H. Tse and M. Witheridge, "Metamorphic Robustness Testing: Exposing Hidden Defects in Citation Statistics and Journal Impact Factors," in IEEE Transactions on Software Engineering,

vol. 47, no. 6, pp. 1164-1183, 1 June 2021, doi: 10.1109/TSE.2019.2915065.

9. A. Agrawal, W. Fu, D. Chen, X. Shen and T. Menzies, "How to “DODGE” Complex Software Analytics," in IEEE Transactions on Software Engineering, vol. 47, no. 10, pp. 2182-2194, 1 Oct. 2021, doi: 10.1109/TSE.2019.2945020.
10. G. K. Rajbahadur, S. Wang, Y. Kamei and A. E. Hassan, "Impact of Discretization Noise of the Dependent Variable on Machine Learning Classifiers in Software Engineering," in IEEE Transactions on Software Engineering, vol. 47, no. 7, pp. 1414-1430, 1 July 2021, doi: 10.1109/TSE.2019.2924371.
11. M. W. Call, E. Fox and G. Sprint, "Gamifying Software Engineering Tools to Motivate Computer Science Students to Start and Finish Programming Assignments Earlier," in IEEE Transactions on Education, vol. 64, no. 4, pp. 423-431, Nov. 2021, doi: 10.1109/TE.2021.3069945.
12. G. Jahangirova, D. Clark, M. Harman and P. Tonella, "An Empirical Validation of Oracle Improvement," in IEEE Transactions on Software Engineering, vol. 47, no. 8, pp. 1708-1728, 1 Aug. 2021, doi: 10.1109/TSE.2019.2934409.
13. R. Verdecchia, E. Cruciani, B. Miranda and A. Bertolino, "Know You Neighbor: Fast Static Prediction of Test Flakiness," in IEEE Access, vol. 9, pp. 76119-76134, 2021, doi: 10.1109/ACCESS.2021.3082424.
14. Y. Wang, T. Ma, D. Pei and C. Chen, "Research on the Chamber Pressure Test Method of Small Caliber Weapons Based on a Double-Layer and Double-Grid Structure Strain Tester," in IEEE Sensors Journal, vol. 21, no. 17, pp. 18554-18561, 1 Sept.1, 2021, doi: 10.1109/JSEN.2021.3087615.
15. S. Jiang, M. Zhang, Y. Zhang, R. Wang, Q. Yu and J. W. Keung, "An Integration Test Order Strategy to Consider Control Coupling," in IEEE Transactions on Software Engineering, vol. 47, no. 7, pp. 1350-1367, 1 July 2021, doi: 10.1109/TSE.2019.2921965.