e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

ARTIFICIAL INTELLIGENCE: LEVERAGING AI-BASED TECHNIQUES FOR SOFTWARE QUALITY

Naimil Navnit Gadani[*1]

[*1]Senior Software Developer, Content Active LLC, Houston, Texas, USA.

DOI: https://www.doi.org/10.56726/IRJMETS60018

ABSTRACT

AI-based techniques have been developed and employed to detect improper use of the software or identify unexpected behaviors. In this context, AI-based methods have been used to find improper use of the system by analyzing the logs generated by the system. Similarly, AI has been used to find improper use by performing a statistical profile of use-cases or anomalies in the internal state of the system, i.e., detect runtime errors. Machine learning and deep learning are now commonly being applied to software development, monitoring, and maintenance to improve the software quality; i.e., AI can be used to construct a model to monitor how software is used in production and aid in the prediction of which code would introduce vulnerabilities. Model-based Bayesian deep learning technique is used for regression testing, where input entails the probability distribution range while output conveys how many probability distributions are compliant under such a range. [5]

AI is an umbrella term, encompassing both supervised/unsupervised learning approaches based on regression/classification problems to an agent-based approach in which one would simulate human behavior. AI can analyze data to identify patterns and create models to make decisions (or predictions) based on probabilistic reasoning. AI can be used when establishing a combination of input attributes along with the needed output (discrete or continuous values) is difficult to obtain using the classical, algorithmic approach.

Keywords: Artificial Intelligence, Software Development, Software Quality, Software Testing Techniques.

I. INTRODUCTION

Artificial Intelligence in Software Development

AI has significantly evolved over the years, and it continues to seep into various facets of industries, including the aspect of software development. For example, AI has been used to predict software defects, recommend final solutions, predict bugs for fixing, validate test cases, build software agents that are capable of analyzing and fixing software bugs, model and predict refactoring activities, predict RESTful API documentation exceptions, fix the syntax and semantic errors in source code, and more. On the other hand, AI research has also benefited from the massive investments in open-source software and shares AI research models and tools through open-source platforms. Participants in the Hey program have developed several AI-based techniques by creating systems for a variety of application purposes, such as tour planning, automatic coding, web automation, etc. [1][2][3]

The landscape of artificial intelligence (AI) has hit its golden era, captivating various industries, including the software industry. Although software development has leveraged software tools with AI capabilities as assistance for developers, there is still an immense potential to harness AI techniques as intelligent explorers for a majority of software development activities. Notably, software quality is a prerequisite for ensuring that the software functions as intended and without errors, which is achievable via AI-based exploration. Hence, this review aims to explore the possibility of leveraging AI-based techniques for improving software quality. [4]

II. FUNDAMENTALS OF SOFTWARE QUALITY ASSURANCE

Software consists of traditional expectations (performance, reliability, and functionality), and there is also faster time to market. The main purpose of the agile approach is to be faster. For example, in time-based competition, the advantage can be in finishing an activity faster than the two-week sprint. Three questions are important in this area of time-based competition: What is the likelihood of deadlines being missed? What is the nature of that risk? How are those risks being better managed?

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

Objectives of System Design

Figure 1:

Source: https://www.geeksforgeeks.org/goals-and-objectives-of-system-design/

This table summarizes key areas and processes to ensure software quality in software engineering.:[6]

Table 1:

Direction	Description
Peer Reviews	Reviews carried out by programmers and system implementers of a project deliverable to identify defects. Defect fix rates for all projects can be estimated several weeks or months after the project has started. Managing commercial value (CV) and scoring is essential.
Scientific Measurement	Used in software engineering to ensure quality, such as verifying the number of drivers and representations of components like TCP in IDPS systems. Quality in business agreements may involve assessing models or risk profiles.
Configuration Management	Identifies software configuration at different points in time to manage changes during development. Involves configuration control, addressing issues, assessing changes, and enforcing configuration management across the development organization.
Requirement Engineering	Includes cost estimation, project planning, scheduling, and setting quality goals. Defines attributes and metrics such as size, time complexity, and reliability. For large systems, includes dynamic behaviors and security issues.
Measurement & Obstacle Analysis	Involves estimation of CPUs and execution time, software size and cost, problem complexity, algorithm identification, relative size and difficulty of algorithms, and system efficiency.
Software Testing	Identifies test cases based on requirement matrices, executes test cases, and reports defects. Includes verification and validation through reviews, audits, test planning, specification, execution, and supervision.
Quality Management	Involves setting up quality plans, standards, policies, guidelines, and creating project-specific procedures and templates. Promotes software quality using methodologies like Waterfall, V-model, RUP, and Agile.
Software Quality Assurance	Methods and tools to ensure software performs its functions effectively and is free of defects. Measures and analyzes product conformance to user needs, schedule, and cost constraints. Includes details necessary for meeting production schedules.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

2.1 Key Concepts and Techniques in Software Quality Assurance

To understand AI and the related AI Knowledge, it is important to have a basic understanding of the types of software testing and the purpose of each type of testing. Types of software testing that relate to AI are Unit testing, also known as component testing, refers to tests carried out on individual units of source code so that the functional correctness of a unit is related to micro-level system functions. Integration testing usually takes place after component testing and connects the components through the application of data flow. Integration testing may take a different level of effort depending on when files and databases are integrated into the application process. System testing usually takes place at component testing and tests across an application, looking at primary interactions of the system.

1. **Acceptance Testing**: This testing directly addresses the issue of a system's usefulness by testing whether the software product fulfills business requirements matched in a project's client's specification.
2. **Quality control**: It is a technique where organizations identify and incorporate quality elements in developed software before releasing it to clients. Quality controllers can use plans, policies, and procedures; customer and project data; and production and inspection records to achieve the goals of this process. - Techniques of testing: There are classical testing methodologies in testing the developed software and applications that developed quality metrics. Evolution has seen aging but applicable to specific types of software and apps over time as new software quality control strategies materialize.
3. **Quality Metrics**: A Quality Metric is a measure of software system concept according to some dimension of software testing methodology to predict a characteristic of software quality such as defect proneness, or that must be in a software product in order to pass functional testing. These are appraised for the management, developer, and individuals with a specific level of experience and knowledge to get an idea about the system quality. Quality control strategy: It is an approach used to validate the developed software. Depending upon the testing parameters, it's classified into: -

a. White Box Testing: Software testers, developers, and business stakeholders are used in evaluating the software application for defects.

b. Black Box Testing: Concept behavioral testing or requirements-based testing, During this process, the functionalities of an application/software are tested without having an internal coding structure or software program knowledge.[25]

To understand the application of scientific fields like AI, embedded technologies, and math, an understanding of the key concepts of SQA is necessary. According to IEEE Standard 729-2009, Software Quality, quality is the degree to which a software product meets the requirements or satisfies the needs of its users or customers. Quality assurance is the process of evaluating overall project performance on a regular basis to provide confidence that the software will conform to established development requirements as long as the products are being developed. For software, there is no easy way of defining quality that is universally accepted, but it is possible to measure it in terms of developed software artifacts' quality attributes in a given context. [

III. INTERSECTION OF AI AND SOFTWARE QUALITY

Classical software quality practices are ripe for AI-based disruption. Since the early 1990s, there was more evidence that software should be managed and not just released. Shortly after that, and with increased interest from practitioners since around 2016, several AI underrepresented techniques are gaining larger interest, particularly: transfer learning, hard parameter sharing (HPS), multi-task learning (MTL), or different instantiations of it like the liquid time constant multi-task learning (LTCM), or end-to-end memory networks to manage software stakeholder requirements. All these are proposed in general because of the need to understand how software requirements can be managed managed as one. For that reason, most of the works for requirement quality assessment focused on the development of critical success factors, the elicitation and the association of stakeholders and users related to the software, and the required quality of requirements (QoR), as a culmination of all stakeholders involved.

[7]Artificial Intelligence plays a considerable role in leveraging software quality. Besides improving the performance of the underlying algorithms, its higher computational power, expressiveness (or learning new representations), and generalization properties can offer a more substantial and accurate vision of the system

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

to reason about. The symbiosis between AI techniques and software quality was labeled as "AI-based software quality" in Industry 4.0.

Software quality is critical for stakeholders, as poor software consolidates harmful potential. It increases resource spending, makes users unsatisfied, or directly impacts negatively the business. Therefore, it is important to ensure that we are delivering high-quality software in which different quality aspects have been suitably addressed.

3.1. Challenges and Opportunities in Integrating AI into Software Quality Assurance

AI can work as an unbiased tool which complements humans in opinion formulation, decision optimization and process planning. The equipoise switching capability is another feature, which involves the flexibility of switching between multiple tools to handle gaps in one tool. Automated software quality assurance solutions powered by AI still have a long way ahead to be universally applicable and efficient. The claimed benefits of AI will require accurate algorithms and efficient techniques of integrating the present AI methods into the software quality management processes. To this end, one important aspect to consider is the availability of good quality data for training AI algorithms specific to the software industry.

AI has revolutionized many industries in the last decade and shows considerable promise for making systemic advancements in software quality assurance. One of the key reasons why one cannot directly apply conventional AI techniques to software quality assurance is that the field of software engineering has its own unique features and characteristics. For example, the landscape of software engineering is highly dynamic, and it is necessary for the AI techniques to consistently adapt according to the changing situations. Furthermore, data quality has always been a major concern in the application of AI. Specifically, the human-authorship nature of the data used for training and testing AI models can also introduce bias into such models. Since the software artifacts are all written or developed by humans, this represents an intrinsic barrier. While others can pretend that the AI does not or should not replace the work of human developers, AI tools have already become increasingly effective and important. Fortunately, there are subtle differences in the extent of application, where AI can perform tasks, such as bug detection, verification of API properties, etc. [1]

IV. MACHINE LEARNING FOR SOFTWARE TESTING

Various algorithms have been employed for using machine learning models in software testing, widely divided into three categories: supervised, unsupervised, and semi-supervised algorithms. Besides, machine learning-based test-suite optimization, fault prediction, bug profiling, and automatic fixing are also among other prominent tasks that make use of machine learning in software engineering. Not all domain practitioners are aware of the potential of the application of artificial intelligence-based tools and techniques to their domain and their impact. Testing of complex systems requires the evolution of new testing paradigms. ML can potentially greatly influence these new paradigms, as it already leverages existing testing technologies.

Machine learning (ML) has proved to be effective in numerous applications of software engineering. In software testing, automation of testing tasks is a challenging problem. One of the major challenges of software testers is in dealing with the oracle problem. The oracle problem arises when testers spend time investigating whether some observed behavior is indeed a fault or not. One can often address this problem by employing machine learning on a large amount of historical data to predict the probability of the existence of faults in the observed behavior. Using such prediction models, the testing/troubleshooting effort can be focused on the areas with a higher likelihood of containing the faults. Additionally, testing and troubleshooting processes can be further optimized by recommending fixes. [8]

4.1. Types of Machine Learning Algorithms for Software Testing

Automatic Test Case Generation (ATCG) Algorithms: Basically, this can be divided into traditional and search-based algorithms. ATCG algorithms are the most common type of algorithms in software testing. The search-based technique is based on ML techniques. Some of these techniques like Genetic algorithms, Search algorithms, etc. can be used for automatic generation of the test cases.

Anomaly Detection Algorithms: This is a technique used for detecting the abnormal behavior of software when it fails to behave according to the predefined requirements or responses. It compares the observed data with the learned data. If learning data is perfect or close to perfect, then remaining may be considered as anomalous

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

data. For this technique, the various algorithms that can be used are Neural network (Autoencoder), One-class SVM, K-nearest Neighbor, local outlier factor, k-means clustering algorithm, DBSCAN algorithm, Custodes. These techniques have limited empirical research, the reasoning being the may which is required to be labeled classified due to time, and also the implanted sensitivity of the signed.

Predictive Testing Algorithms: Most research has reported predictive algorithms as one of the most preferred algorithms in ML for software testing. Predictive testing aims to predict the test case that is most likely to fail when run. The techniques that can be used for building such predictor are SVM, C4.5, KNN, Neural Networks, Decision Tree. [9]

V. NATURAL LANGUAGE PROCESSING FOR REQUIREMENTS ANALYSIS

Natural language processing (NLP) is the new trendsetter in requirements extraction or categorization. In an overview of the application of AI techniques in software engineering, highlighted the application of classical and modern AI techniques with NLP for performing various analytics to analyze the complexities and other prospects in software engineering. The authors proposed a symbiotic approach of combining classical, post-modern, and novel AI techniques to perform various software artifact processing tasks, like extraction, comprehension, analytics, and validation, to obtain intended software quality attributes like understandability, maintainability, minimization of underfitting, and overfitting in the resultant systems. When non-functional requirements are gathered, they are usually design constraints or factors that affect how the system is designed. All end users and stakeholders must understand and analyze these requirements in order to find solutions to the problems that were described. Thus, the representation of those requirements must be easily understood and analyzed, so that end results are accurate and clear for both the customer and all actors involved in developing software. presented the flexibility and effectiveness of an application of NLP in the software domain, especially in requirements extraction. They proposed a framework using an NLP technique such as open-door text to automatically extract useful pieces of information using data from various data sources like Bing, Google, or others. It is implemented to fetch data related to Islamic data, which results in the extraction of software-specific requirements.

Requirements analysis is an essential stage of software development and maintenance that directly affects the quality of the resulting system. Process improvements in the requirements analysis can significantly increase development productivity and enhance communication among participants end-users, analysts, engineers, and managers. Requirements are generally expressed in a natural language, which makes them a challenging task to manage, evaluate, and measure due to their subjectivity, ambiguity, and incompleteness. However, the massive amount of knowledge and information present in the natural language text can be analyzed to produce valid information. A natural language processing (NLP) technique has been widely used to analyze complex requirements by extracting and structuring meaningful information from the requirements, along with the validation of the end results. [10][11]

5.1. NLP Techniques for Extracting and Analyzing Software Requirements

Gandomani et al. focus on the extraction of two types of initial software requirements from texts: functional requirements and quality requirements. The former are specified in use cases and the latter in non-functional requirements expressed as patterns. We participated in the 3rd and final iteration of the REVEAL Text analytics research program of the Australian government. This paper expands on the Results and Discussion chapter of the research report as presented at Australian Innovations and Knowledge Discovery Research Showcase (iKID) (Ker 2021) and provides an introduction to and overview of the research. [12][13]

Different NLP-based methods: Couclelis et al. suggest a quantitative model of the basic ways that people interact with geographic artifacts in order to find the general features that users need for a variety of tasks and that a geographic information system (GIS) needs to be set up for. This study is based on an analysis of more than 10,000 geographic information usage scenarios. Yhi et al. develop a framework for the formalization of GIS requirements based on both the concepts posited by Giddens's structuration theory and Searle's social ontology. The framework focuses on explicating the software requirements that pertain to the agent-internal structures that underpin work practice, developing the capability to complete one's part of an ongoing social practice, and meeting the relevant performance indicators at a granular level. Rubio et al. propose a pipeline to

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

support the discovery, management of open data from the software requirements specification (SRS) through to verification of the linked data expressed in the semantic resource description framework (RDF) format. [14][15][16]

Natural language processing (NLP) techniques and tools have been widely employed to extract and analyze software requirements, which are typically specified in natural language. The NLP-based methodologies generally focus on the detection of ambiguities, inconsistencies, duplications, and other types of issues in the software requirements, either collaboratively (e.g., user requirements and supplementary specifications, validation of formal models) or individually (e.g., to validate the consistency of requirements modeled as use cases, to identify implicit requirements, to measure the understandability of requirements and to prioritize functional requirements). These kinds of methodologies use NLP techniques to analyze the textual artifacts produced during the requirements analysis phase, i.e., user stories, use cases, glossaries. Therefore, it is essential to be acquainted with the NLP techniques that can be employed to elicit, analyze, and negotiate software requirements.

VI. AI-DRIVEN CODE REVIEW AND STATIC ANALYSIS

The focus of this paper is on AI-driven automated code review and analysis as a widely acknowledged area of software development where successful AI is likely to have the greatest impact on software quality. Sparked by Tesla's new and advanced approach to AI in automated code review and static analysis, we aim to catalyze research efforts in this field and promote the development and recognition of valuable AI-inspired techniques that could be implemented. We firmly believe that if successful, phenomena could be co-created between the key players and adopters that would result in widespread adoption of state-of-the-art code analysis in the space of cloud development. We envisage better collaborations between developers, security specialists, and cloud developers as a result, with an eagerly adopted myriad of plugins that would make traceability and guidance of the process a great deal easier.

Recent advances in artificial intelligence (AI) have raised the possibility of new and more effective techniques that could help improve the actual quality of the reviews and analysis available in the field of cloud computing. This is because AI is good at processing information from data that cannot be codified but is expressed in examples. AI can also help make appropriate recommendations for certain patterns and practices in an ever more complex software engineering space. This might be possible because software repositories contain very large collections of artifacts, some of which are already known to be of high quality (e.g., files and artifacts of completed systems), possibly containing training data for learning about software quality from existing projects. All of these factors (which we describe in detail below) open the door for utilizing machine learning and AI-based techniques for decisions that have often been thought to require human insight.

Static code analyzers are intended to automate the process of code review by identifying potential issues in a codebase, but they generate many false positives by matching any code that fits a certain pattern without asking whether the match is relevant. This can make such analyzers labor-intensive to use, with the need for a heavy review of their output before their results can be trusted. This can discourage the adoption of such tools by developers who are already under pressure to deliver the new features often being requested of them. As a result, the adoption of tools in the industry today is low, leaving many coding mistakes uncaught.

Code review is known to be an effective way for improving software quality, design, and catching vulnerabilities even in the early stages of development. However, manual code reviews are slow and costly, which demands effective handling of large volumes of changes, especially with respect to cloud computing, where it is common for very large web systems to be updated every hour or so. Consequently, reviewers tend to overlook important issues, leaving most of a system's changes uninspected. [17]

6.1. Automated Code Review Tools and Techniques

The main characteristics of these AI-based techniques mainly rely on automated pattern recognitions, traffic (messages or data) analysis, automated code review basis, big data analysis, distributed system communications for firewalls, cloud services, and service-oriented architecture. Given the wide development of AI-based techniques, researchers have made efforts to apply these techniques in code review processes to make the process efficient and more accurate. In this way, machine learning also plays an effective and efficient

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

approach to improve the code review process. However, how machine learning could be employed to automate the code review process was unclear in various aspects, i.e. software quality, testing, systematic literature review, and current issues in machine learning-based tools for code review. Therefore, we explored how machine learning could help develop code review tools in more intelligent and sophisticated ways.

The most commonly used code review is a manual type, where the actual developer is responsible to review his/her respective colleague's code. This type of code review is also termed as "peer review" or "suggestive review". While discussing automated code review, this type of future review is also termed as "definitive review" or "assessive review". Due to the increasing number of projects, it becomes very complicated and time-consuming to wait for someone's answer, which is very difficult to get in the manual review. Once others review your code and they identify the issue, then a feedback response comes back in step-by-step. Whenever you receive feedback, then it's very easy and simple just to modify your code according to the feedback. As the system is getting more complex, it becomes necessary to have automated code review, which is used to clean the code. This type of system is also very essential in the system because of code optimization as well as the avoidance of security vulnerabilities by leveraging automated machine learning approaches.

Code review is another important aspect that plays a significant role in the software development lifecycle. Once the coding is done, the next important step is to review that code, which is important in order to keep a check on the development quality. This code review is basically of two types: manual code review and automated code review. [11]

VII. PREDICTIVE MAINTENANCE AND FAULT PREDICTION IN SOFTWARE SYSTEMS

Fault prediction is the area of predictive maintenance that aims to estimate and forecast any potential future faults. Predictive maintenance is a proactive maintenance strategy that uses AI to analyze data, identify expected faults or equipment failures, and bring them to the attention of PM staff so that repairs can be performed before the failure occurs and causes production downtime. Predictive maintenance time expects analysis to determine the defects, anomalies, and potential of any future defects accumulated in different products. Common statistical techniques include Weibull analysis, regression analysis, and failure mode and loss analysis (FMEA). AI-based techniques are a core foundation of predictive maintenance for software. The goal of fault prediction is to determine whether or not a fault will occur. There is somewhat of a confusion barrier between fault prediction and fault location, but fault prediction goes further by providing critical success factors to demonstrate why the fault is most likely to occur.

Our society is characterized by an increasing reliance on software systems across economic and social domains. From autonomous transportation to personalized healthcare devices, the reliability and predictability of software systems determine their acceptance and societal impact. IT providers and businesses need reliable and stable software systems to build and offer valuable services. Hence, there is a need for additional innovation methods that apply the knowledge of AI in various software application development sectors. Leveraging AI-based techniques can create discontinuous innovation in various contexts and sectors by considering architecture and algorithms that are developing to achieve a high level of software quality.

7.1. Proactive Maintenance Strategies using AI

AI techniques for Software Maintenance:

1. **Predictive Analytics:** Predictive models are created and require training through a learning process. By using predictive modeling, it identifies patterns that lead to failure. A strong AI model prediction is shared as new unseen data is processed.
2. **Anomaly Detection**: A machine learning algorithm can detect events that do not conform to an expected pattern. Using historical system statistics, a model can be trained to detect anomalies.
3. **Adaptive Maintenance Algorithms**: These algorithms are AI-based monitoring systems that adapt maintenance in real-time to avoid catastrophic failures. A real-time reduction engine is used to avoid an overwhelming number of warnings. The adaptive maintenance algorithms can adjust safety measure thresholds, ensuring system availability while keeping a system in a safe state.

Proactive maintenance reduces the frequency of costly repairs and increases the reliability and quality of software systems. Predictive maintenance uses the historical information of operational data for predicting

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

failure. However, accurate predictive results are hard to obtain since historical databases are often messy and can lead to biases. Anomaly detection is an AI-based maintenance strategy that is used for gathering useful information from noisy and messy historical databases. Furthermore, in recent times, with advancements in adaptive and self-adapting systems, AI techniques adaptive to dynamic failures are gaining increasing attention. AI-based proactive maintenance minimizes large costs by better utilization of systems and reduction in safety costs. AI-based software maintenance tools can increase the safety and lifetime of constructed complex systems. A general outline for such a maintenance approach is shown in Figure and elaborated below.

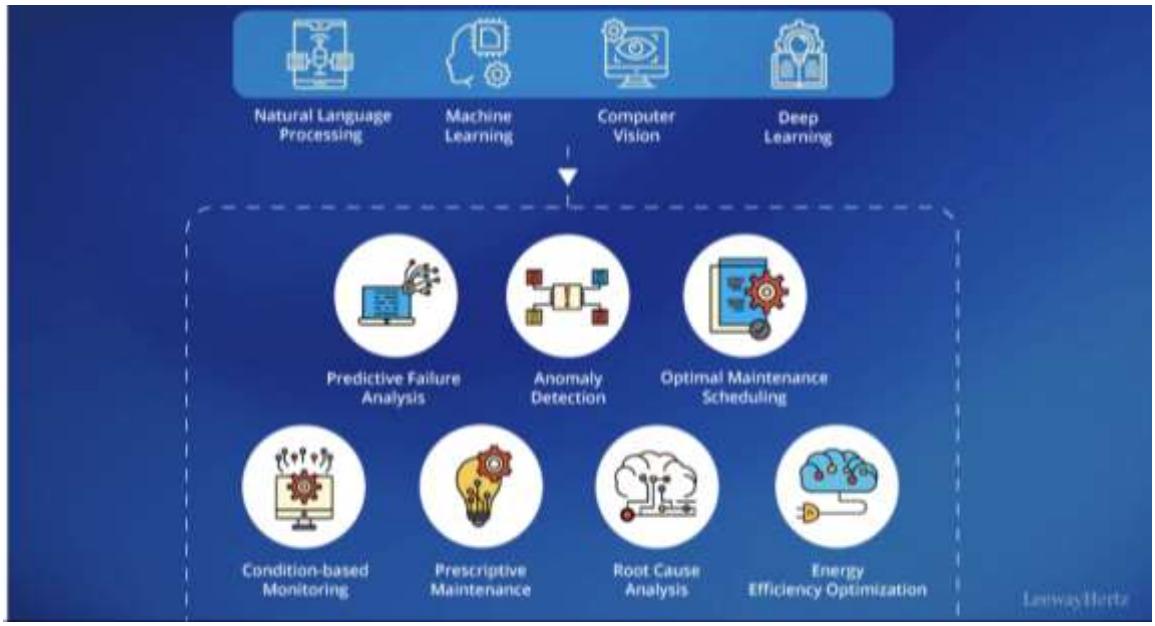

Figure 2:

Source: https://www.leewayhertz.com/ai-in-predictive-maintenance/

VIII. AUTOMATED BUG DETECTION AND RESOLUTION

Classical software development relies on human-run methods. A developer writes the code, then runs a set of system tests to detect bugs. The developer may also perform code reviews to detect bugs and enforce the quality of the code. DevOps and other forms of automated software development gradually replace traditional development, but even then, humans still carry out the software tests. Bug detection and bug resolution are still time-consuming and expensive tasks. Hence, there is a growing interest in automated bug detection and resolution. This survey paper aims to summarize the existing methodologies and concepts of automated bug handling. Some studies at least partially rely on AI techniques to tackle automated bug detection and/or bug resolution. The major part of the reviewed papers tackles either automated bug detection or resolution, not both. This survey paper is important in collecting and summarizing the state-of-the-art tools. The survey is up-to-date and provides a good sound basis for developers who might be interested in adopting automated tools for software development but are not familiar with the area.

Automated bug detection and resolution are becoming easily accessible in state-of-the-art software development thanks to AI-based approaches. There are several survey papers describing different tools and methodologies for automated bug detection and resolution that are often developed using AI. For example, the most recommended AI technique for these tasks is symbolic execution, discussed in depth in Section 8 of this paper. The results of analyzing existing tools show that new AI-based methods might contribute to increasing the reliability and robustness of software systems if one makes use of tools that consider the widest possible scenario, including those that have not yet been exploited. [1]

8.1. AI-based Bug Detection Tools and Approaches

1. **Automated Debugging**: Debugging is always given significant importance in the software development journey to ensure the quality of the product. Intelligence-based solutions drive the success of the automated debugging mechanism. AI-based techniques support automated debugging by identifying and fixing the bugs

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

with minimal effort. Through reasoning, AI tools can replicate the defects and diagnose wherever necessary. By taking advantage of the AI-based problem diagnosis model, the aforementioned method will be implemented with some infrastructure. From the user's debugger input events, the problem diagnosis facility can retrospectively access the debugger's execution behavior. The retrospective analysis of debugger execution behavior can provide important cues regarding which code changes should be made to the software being debugged. unsupported code contract added to the procedure entry point in toString override. ShopSpecials are assigned to each object, but it may be more memory efficient to create a shop Special static member. Avoid making new objects through function calls if possible. You can add -DBENCHMARK_EQUALS to measure the performance of the equals function and optimize. Please benchmark it using a large object container that Spiff town Manager produced. Yield Contents().compute Flattened Container(). Let's see how close it is to the equivalent. The algorithm will definitely be shorter. If it's around 5% or so, then use it; otherwise, don't. Similarly, can add -DBENCHMARK_ITERATEATEQUALS to compare next() and deep equal compares. [18]

2. **Bug Pattern Recognition:** The bug patterns can be recognized, which will happen more in the new location due to the increase in feature conflicts or the declination of performance feature effects. Every time the genetic algorithm proposes most of the new association rules, and they are stored in the bug pattern database (BPD). We added a pruning function to the genetic algorithm because the size of the BPD is a constant amount that the user sets. A new bug is detected if the probability of a bug being present is greater than some user-defined threshold. In the practical implementation, we argue that a developer will use association rules from the more reliable classification method and use the rules mined from the extension as a form of double-checking by the analyst; therefore, the rule sets are performed sequentially.
3. **Bug Detection:** In recent years, AI, especially machine learning, has been widely used for bug detection. Several approaches have been proposed for this purpose, like learning text processing models for bug reports and work items. In recent years, a large number of such models have been proposed, and they can be largely divided based on their targets, like bug detection tools, debugging supports, program repair, etc. Besides, many techniques apply probabilistic logics and graphical models for bug detection, while some others use data mining to offer defects and risk classification.

In this subsection, we discuss AI-based approaches and tools to support bug detection and resolution. They can be classified into three main directions: bug detection, program repair, and other AI-based techniques, and the work considered in the literature is summarized as follows:

Table 2:

Direction	Description
Bug Detection	Techniques and tools designed to identify and report bugs in software. This includes static analysis, dynamic analysis, and other automated testing methods.
Program Repair	Approaches and tools aimed at automatically fixing bugs once they are detected. This can involve patch generation, code transformation, and other repair techniques.
AI-based Techniques	Various other methods utilizing artificial intelligence to improve software quality, including predictive models, anomaly detection, and optimization algorithms.

IX. ETHICAL AND LEGAL IMPLICATIONS OF AI IN SOFTWARE QUALITY ASSURANCE

This paper lists 11 types of ethical concerns that our goal for high-quality AI software should address. It also adds to the discussion about how important it is to set and follow clear ethical rules in programming, software quality, computer education, and standardization in the field. Our proposal in this paper directly addresses the latter category of ethical concerns associated with AI in software verification.

Codifying such fairness, transparency, and accountability with which AI systems should be made operable in software testing is a challenging multi-disciplinary task that lies beyond immediate empirical investigation. In

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

this paper, we describe these principles based on a thorough study of software testing and AI techniques for verifying and validating software, as well as new legal advice and European agency initiatives on AI. We also looked at the current state of responsible AI in software testing and machine learning in general. It is our aim to produce a multi-disciplinary consensus-building document where the specifics of practical implications are left to detailed legal and computing analysis in companion papers.

Ethical and legal guidelines must specify the conduct expected from practitioners in such cases, and may also be used to guide the development of an AI system itself. A legal entity, a person developing or using AI tools responsible for an independent decision, may be questioned about the compliance of their actions with certain principles, in the spirit of the law, even though they act within it. In this sense, both the person responsible for the quality of software and the person developing the tools for its verification may be obliged to ensure compliance with specific quality-secure AI systems based on fairness-preserving algorithms.

AI-based techniques for software quality assurance, together with their many advantages, introduce as many challenges to practical implementation, among which are ethical and legal matters. The use of AI in automated testing specifically raises several ethical concerns. AI algorithms that decide on software quality may lack fairness, transparency, and accountability often absent in software development processes. [19][20]

9.1. Ensuring Fairness and Transparency in AI Algorithms

Different methods, such as training a linear model to find related features or using residual analysis, can reveal the gender and ethnic biases present in a chosen algorithm. Building a causal model and using a number of different methods to calculate quantities of interest to find places where discrimination is present in racial categories like missed opportunities, distributive fairness, and treatment equality are other ways to find the bias that is already there in a set of data. Another way to construct a causal model is to utilize the counterfactual model assisted by the same quantity theorized previously to allow for equal opportunities and equality. Different ethical and societal principles need to be accounted for when designing Explainable AI (XAI). These principles comprise utility, responsibility, transparency, auditability, predictability, acceptability, fairness, harm avoidance, trust, and reliability. These issues are often part of the ethics analysis used as a way to check the behavior of AI-based systems. The underlying rationales, principles, and norms related to ethics are also described in these checklists. Moreover, several principles and ancillary guidelines are also accounted for when designing XAI. The principles just described have recently been used to review multiple existing XAI literature and discern it from these principles. In providing an appropriate ethical argument, there are several defenses that can be used, such as respect for individual beliefs. Another defense is to show that the overall good will be achieved despite the constraint of an ethical principle. For example, pharmaceutical companies can argue that even though insulin is expensive, the end result of their work is the medical treatment of millions of diabetics worldwide. [21]

Respected data scientists and AI researchers argue that it is time to ensure that AI-based techniques affect us fairly and transparently. We can either pursue them in combination with our case studies to ensure our AI technique not only assists companies and researchers to improve software quality but also helps to achieve it transparently and justly. We are in the process of constructing a generalized framework based on the above principles and designing algorithms to pursue it in the future. The ethics of AI-based software engineering is an emerging domain and has already started to play a crucial role in designing AI-based techniques for solving software engineering problems.

X. CASE STUDIES AND REAL-WORLD APPLICATIONS

Janzen provides a thorough overview of AISQ implementations and experiences in the context of the case study partner Quinyx to start this section. Of special interest here therefore are the framing and the structuring of the practice papers provided in this section, offering a wide variety of context and domain perspectives. We have organized the papers based on the aforementioned AISQ application aspects that were identified during the workshop. In some cases, the application domains or systems under consideration are similar, but the different AISQ techniques or AI models applied have led to different insights. In other cases, a single AI technique was leveraged with a variety of research and practice perspectives.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

In collaboration with our case study partners LowCode Agents, Keller Group, msg systems AG, and Quinyx, we have identified a number of real-world applications highlighting the successful impact of AI on ensuring software quality. Feasible positive outcomes with respect to IS are, for instance, the optimization of the test selection processes to enable better regression coverage, and superior test case and requirement prioritization based on learning historical issue information. Beyond the case study applications, the articles in this book provide insights into how AI can be used to efficiently generate concrete recommendations for improved testing strategies, or learn to repair not only possible software faults in integration flows but also the underlying functional and/or performance issues that are violating requirements. A common aspect among the case studies mentioned here is their translational angle towards real-world implementations, which are discussed and reflected with practice and research outputs. [22]

10.1. Successful Implementations of AI in Software Quality Assurance

1. Examples of such effective implementations of AI techniques in the Software Quality Assurance cycle.
2. Highlight case studies on how artificial intelligence enhances software quality and reliability, which could be in different domains. Some of them may include: - Education and Training Systems - Algorithm development - Learning Analytics - Clinical Decision Support System - Financial forecasting - Security Testing - Cloud Resource Allocation - Medical Decision Support System - Cybersecurity - Business Process Modeling - Bug Prediction - Software Defect Prediction - Solution to Assessment Campaign [1]
3. A case study on the role of Recurrent neural networks (RNNs) to catch vulnerabilities in a software system. The areas that could be target domain the case targets in; the software available/industry available that are making use of it authorities currently using AI-based practices.

Artificial intelligence (AI) holds the potential to change the landscape for all business and service sectors, including the development of high-quality software applications. This section puts forth a case study and/or an example of a successful implementation of AI in software quality assurance. These case studies can be of general or industry-specific enterprises related to AI-based testing (AItes), AI-based static and dynamic program analysis, AI-based software quality metrics, and what can be identified from mining software repositories using AI techniques, etc. These cases can help the reader know how industry professionals are leveraging these AI techniques and approaches in their organizations. In which domains and projects are they mostly adapted? What are the challenges faced and benefits achieved and realized? Recommendations are given for the practitioner augmenting AI techniques to achieve software development/testing and verification filtering.

XI. FUTURE TRENDS AND EMERGING TECHNOLOGIES IN AI-DRIVEN SOFTWARE QUALITY ASSURANCE

Ambiguous or Controversial Results in Software Quality Betterment Starting the development of a productive socio-technical AI ethics, with consequences for international HRM, will only result if taking into account the Czech technical and economical specifics of Hi-Tech importance. The public AI we are going to connect to is more of an AI advisor for the business processes optimizing with the knowledge of a consciousness born in social-governance AI design. Knowledge and the quality of knowledge acquisition are (probably) one of the people's mind processes transferable to socio-technical AI. Given that the capability exists and is closely coupled with learning, it is time to view the AI-society collaboration from an additional and disciplinary angle, which could only provide a new viewpoint, not an answer today. The next intellectual and interdisciplinary step should be in the research into: how to predict the knowledge evolution? In the process, the learning units, the connection or network process itself could be an area to be predictive. For example, today, extra unit connections could mean extra proposed but still unrealized solution.

Interdisciplinary Methods and Techniques in AI-Driven Software Quality Assurance Software development and software quality are areas that have originated in 1940–1960 with influences from different streams of knowledge. The cognitive approach sheds more light on these participants involved in the software quality process, and the enterprise systems upgrade and adaptation towards emerging technology implementation are the key challenges of tomorrow. We believe that work described in Section 11.2 could inspire the AI research into software quality upgrading, using an alternative methodology embedding knowledge. This could be

particularly useful given the ambiguity of the results obtained by Kotulski regarding the software quality development trajectory. The legal offer of the emerging technologies can then be placed in the legislative footsteps and hardened from the international Human Resources perspective.

Open-Source Tools for AI-Based Tools Ecosystem While artificial intelligence (AI) teams are trying to develop the most advanced ecosystems, programming skills for basing technologies are required. In our opinion, the future of AI in quality assurance will have an echo in the open-source and open-accessed community-driven ecosystem. AI-based tools application will become easier and the responsibility for software errors caused by deep learning results and reinforced learning training will be submitted to the open AI community. This step will, on the one hand, allow the creators of management solutions to be freed from the need to learn sophisticated AI programming languages. On the other hand, it will release the creators of AI-based algorithms/programs to concentrate on knowledge acquisition and programming of the specific process behind the software. The future of AI research in quality assurance is linked with the transfer and learning AI techniques, prediction, clustering, and search mechanisms [23][24].

XII. CONCLUSION

AI Ethics and Transparency, which regulate that developers cannot just build any AI but are required to do that in an ethical and transparent way. It has been discussed before that a software that decides on its own how to test presents a major issue. Reviving Self-Learning Systems, like Cruise Cells, that not only produce the tests and test sequences completely on their own, but also set the system under test to certain values in coordination with other cells. Adaptive Quality Assurance, that goes beyond compliance checks. The goal is to interact with subjects, to collect the knowledge from them, learning what is crucial and what is just "usual" system behavior and adapt those parts where currently most effort in testing is needed. Continuous Quality drafts that allow for preliminaries on possibly suboptimal approaches and time for feedback, which are not in the focus in our current study. Testing and high-assurance AI in-a-Loop, participating in the puzzle solver competition, that is let an AI create only high-quality tests. Automated or assisted model generation for automated testing or formal verification. Always remember, every project funded by the German state needs to consider this (that happens less often than one might think).

There are multiple extensions and innovations possible in AI-driven software quality-assurance systems. Many of these are spin-offs that affect other fields as well.

XIII. REFERENCES

[1] S. Martínez-Fernández, J. Bogner, X. Franch, et al., "Software engineering for AI-based systems: a survey," in *Automated Software Engineering Journal*, 2022, dl.acm.org. [PDF]

[2] R. Bibyan, S. Anand, A. Jaiswal, and A. G. Aggarwal, "Bug severity prediction using LDA and sentiment scores: A CNN approach," Expert Systems, 2024. [HTML]

[3] M. Yang, P. Kumar, J. Bhola, M. Shabaz, "Development of image recognition software based on artificial intelligence algorithm for the efficient sorting of apple fruit," in Journal of System Assurance Engineering and Management, Springer, 2022. [HTML]

[4] M. Pandit, D. Gupta, D. Anand, N. Goyal, H.M. Aljahdali et al., "Towards design and feasibility analysis of DePaaS: AI based global unified software defect prediction framework," in Applied Sciences, 2022, mdpi.com. mdpi.com

[5] Z. Bilgin, M.A. Ersoy, E.U. Soykan, E. Tomur, et al., "Vulnerability prediction from source code using machine learning," in IEEE, 2020. ieee.org

[6] MK Thota, FH Shajin, P Rajesh, "Survey on software defect prediction techniques," International Journal of Applied, 2020, gigvvy.com. gigvvy.com

[7] R. M. Samant, M. R. Bachute, S. Gite, and K. Kotecha, "Framework for deep learning-based language models using multi-task learning in natural language understanding: A systematic literature review and future directions," IEEE Access, 2022. ieee.org

[8] Y. Yang, X. Xia, D. Lo, and J. Grundy, "A survey on deep learning for software engineering," ACM Computing Surveys (CSUR), 2022. [PDF]

[9] G. Cai, Q. Su, and Z. Hu, "Automated test case generation for path coverage by using grey prediction evolution algorithm with improved scatter search strategy," Engineering Applications of Artificial Intelligence, 2021. [HTML]

[10] C. Tam, E. J. da Costa Moura, T. Oliveira, et al., "The factors influencing the success of on-going agile software development projects," International Journal of ..., vol. 2020, Elsevier, 2020. [HTML]

[11] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, et al., "Large language models for software engineering: A systematic literature review," arXiv preprint arXiv, 2023. [PDF]

[12] S. Ansaripour and T. J. Gandomani, "Comparison of the Classification Methods in Software Development Effort Estimation," in Intelligent Multimedia Processing Conference, Zanjan, 2022. iau.ir

[13] H. Rastegari, A. N. Najafabadi, T. J. Gandomani, "Dynamic PScore: A Dynamic Method to Prioritize User Reviews," 2023, researchsquare.com. researchsquare.com

[14] C. López-Vázquez, M. E. Gonzalez-Campos, et al., "Building a gold standard dataset to identify articles about geographic information science," in IEEE, 2022. ieee.org

[15] Y. Wang, C. Huang, M. Zhao, J. Hou et al., "Mapping the population density in mainland China using NPP/VIIRS and points-of-interest data based on a random forests model," Remote Sensing, 2020. mdpi.com

[16] M. Yılmaz and F. Terzi, "Measuring the patterns of urban spatial growth of coastal cities in developing countries by geospatial metrics," Land Use Policy, 2021. [HTML]

[17] A. Groce, I. Ahmed, J. Feist, G. Grieco, "Evaluating and improving static analysis tools via differential mutation analysis," 2021 IEEE 21st International Conference, 2021. nsf.gov

[18] W. Deng, J. Xu, Y. Song, and H. Zhao, "Differential evolution algorithm with wavelet basis function and optimal mutation strategy for complex optimization problem," Applied Soft Computing, 2021. [HTML]

[19] O. Akinrinola, C. C. Okoye, O. C. Ofodile, et al., "Navigating and reviewing ethical dilemmas in AI development: Strategies for transparency, fairness, and accountability," GSC Advanced Research and Reviews, vol. 2024, gsconlinepress.com, 2024. gsconlinepress.com

[20] ID Raji, A Smart, RN White, M Mitchell, "Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing," in Proc. fairness, accountability, 2020, dl.acm.org. acm.org

[21] AB Arrieta, N Díaz-Rodríguez, J Del Ser, A Bennetot, et al., "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," Information Fusion, Elsevier, 2020. [PDF]

[22] J. Yin, K. Y. Ngiam, and H. H. Teo, "Role of artificial intelligence applications in real-life clinical practice: systematic review," Journal of medical Internet research, 2021. jmir.org

[23] A. Whiteley, J. Pollack, P. Matous, "The origins of agile and iterative methods," The Journal of Modern Project Management, 2021. journalmodernpm.com

[24] G. Höhne, C. Weber, and S. Husung, "Ilmenau's contributions to Design Science," Design Science, 2024. cambridge.org

[25] V. Garousi, A. Rainer, P. Lauvås Jr, A. Arcuri, "Software-testing education: A systematic literature mapping," Journal of Systems and Software, Elsevier, 2020. [PDF]