

Trustworthy Generative AI for Enterprise Software Quality Engineering: Evaluation, Traceability, and Human Oversight

Brinda Pradeep

Senior Software Engineer Rockwell Automation

ARTICLE INFO

Received: 03 Sept 2024

Revised: 17 Oct 2024

Accepted: 28 Oct 2024

ABSTRACT

A variety of software quality engineering tasks such as test design, defect analysis and technical documentation are currently aided by generative artificial intelligence. Outputs produced usually will appear right on the surface but may also have gaps, incorrect logic, or material that may not be able to be referenced to actual requirements. The paper has created a model within which businesses can implement the usage of generative AI in the quality engineering practice in a regulated and reliable way. Within the framework, an evaluation model generates scores that are given to generated artifacts as correct, complete, traceable, reproducible and confident. It introduced provenance mechanisms to relate the artifact produced to its requirement in the source and its supporting evidence. The output can be reviewed on a risk-based review process, when a human is required to check the output before it is used. The paper presents the findings of an experimental test on generated test artifacts in a number of software modules. The results confirm the argument that combined with structured assessment, traceability, and human control, generative AI can be considered reliable.

Keywords: Trustworthy AI, Human oversight, Requirement traceability, Software quality engineering, Generative artificial intelligence.

1. INTRODUCTION

Large language models have transformed the method of software teams, in terms of testing and quality work. The tools created based on these models have the potential to write test cases and produce defect summaries, draft documentation, and answer engineering questions in a few seconds. Even a language model trained on code repositories of code written in a natural programming language such as Codex, already demonstrates that a language model can solve a significant fraction of programming tasks when sample sizes are allowed (Chen et al., 2021). CodeBERT has shown this ability on tasks that are a combination of natural language and programming language like code-search and documentation generation (Feng et al., 2020).

These profits come to give rise to a new dilemma to the enterprise quality engineering teams. A test case generated may appear complete and not have a critical path. A defect report generated can read and still falsely state the cause of the defect. Its output is fluent but being fluent is not being correct. Software that is generated cannot be taken at face value by enterprises (which use such models) and which are regulated or safety relevant. They require a means of involving themselves with whatever the model came out with, how to relate it to the requirement that it is purported to solve, and when an individual is supposed to intervene.

This need is addressed in this paper. It proposes a system according to which generative AIs should rather be viewed as a capacity that is worthy of trust, rather than convenient. The argument of trustworthy AI research is based on the principle of beneficence, harm avoidance, respect of autonomy, fairness and explainability of a decision (Thiebes et al., 2020). The software testing practical environment takes direct application of these principles as described in the framework of this paper.

Section 2 discusses the related work on code generation, AI based software testing as well as requirement tracing. Section 3 outlines the proposed framework and its architecture. In Section 4 the assessment model is defined. The provenance and traceability layer is discussed in Section 5. Section 6 gives the human oversight process. Section 7

shows the results of the experiments. In section 8, there is a discussion on governance issues. Section 9 ends with a debate on limitations and the paper is summed up in Section 10.

2. RELATED WORK

AI based systems have been investigated by researchers in software engineering in a number of perspectives. In a systematic mapping study of software engineering to the use of AI based systems 248 studies that were published between 2010 and 2020 were reviewed and revealed that the majority of related research was initiated after 2018 with dependability and safety being the most popular properties (Martinez-Fernandez et al., 2021). This development demonstrates that the sector is yet to establish its general guidelines, and quality engineering in generative AI is within the uncharted space.

There is a body of literature on testing of machine learning systems. A mapping study on testing methods of machine learning based systems on 70 primary studies discovered that most research intention has been directed at automated input generation and test oracle generation whereas realistic input generation is still an issue that remains unsolved (Riccio et al., 2020). This concept was implemented in DeepTest with regard to autonomous driving software. It created test input conditions equivalent to those commonly found in the real world like rain, fog and changing light, and it discovered thousands of scenarios where deep neural network models made wrong decisions, some of which might cause deadly accidents (Tian et al., 2018).

Tantithamthavorn and Jiarpakdee demonstrate that AI-based defect prediction models can be precise but not feasible as they are difficult to interpret and difficult to follow up (Tantithamthavorn & Jiarpakdee, 2021). In their effort to bridge a similar gap in the general machine learning field, Mitchell et al. introduce a kind of individual cards called model cards, concise documents containing the information on how a model performs in various conditions and revealing the purpose of its usage (Mitchell et al., 2019).

The third research underlying this paper is traceability research. Hubner and Paech developed an interaction log-based system that matches requirements in an issue tracker with changes to source code, and found a precision of over 90 percent (along with a 80 percent recall) (Hübner & Paech, 2020). Ali et. examined the impact of deleting grammatical categories in text on the accuracy of traceability and introduced a new method, ConPOS which resolves more precisely, recalls better and has a higher average of proximal accuracy than the current models of the text, either the vector space or probabilistic models (Ali et al., 2018). Pokharel and Reza created a tool, which connects both the functional and non-functional requirements with the quality attribute scenarios and a traceability matrix (Pokharel & Reza, 2019). The traceability model proposed by Philip et al., takes into account the feature and requirement-related dependencies and marks potential conflicts when they are at the implementation stage (Philip et al., 2017). Murtazina and Avdeenko approached traceability within agile project by using an ontology to provide support with a coherent description of the types of requirements and requirements relationships (Murtazina & Avdeenko, 2019). These works indicate that traceability can be executed within a very high level of precision, but needs extra planning.

The additional exploration of the framework in this paper is through test generation research. Rojas et al. discovered that a complete set of tests created at once to reach all coverage targets is more likely to achieve higher overall coverage than creating one goal at a time; however, the few goals still benefit from the traditional approach (Rojas et al., 2016). Extending search-based test generation to resource aware sampling and mutation strategies to RESTful web services and finding a relative test coverage improvement to 130.7 to baseline methods, Zhang et al. validated the utility of search-based test generation methods (Rojas et al., 2016). An Improved Combined Fitness function suggested by Sahee and Ray was employed with adaptive particle swarm optimization to enhance coverage of important paths that most of the fitness functions may overlook (Sahoo & Ray, 2019).

3. Proposed Framework

The framework consists of a generative AI system as a part of larger engineering pipeline. The architecture is given in Figure 1.

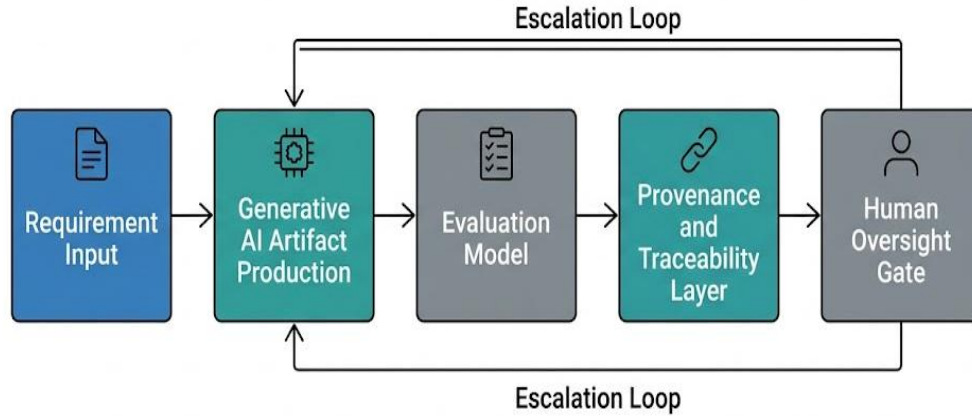


Figure 1: Architecture diagram illustrating the process of processing requirement input, through the generative AI artifact creation, into the evaluation model, provenance layer, and human oversight gate, back to the approved or escalated artifact.

A pipeline starts with a request at the point of a requirement or change request into the system. The generative AI sub-architecture of a component creates a candidate artifact, whether it is a test case, a summary of a defect or some documentation, or it is an engineering answer. This candidate is not sent out to the developer or tester. It initially goes through the evaluation model in Section 4 that scores it on various dimensions. Then it traverses the provenance layer defined in Section 5 that assigns the artifact to this source requirement, the corresponding component and any provenance. Lastly, the human oversight gate denoted in Section 6 would determine the possibility of the artifact to move on its own, or that it would have to be approved by a qualified reviewer to proceed.

This design is based on the same reasoning as in AI based systems research, where testing and quality assurance are the most dynamic sphere of interest of all the activities in software engineering related to AI (Martinez-Fernandez et al., 2021). It is also an expression of the concept that a model output must be complemented by a documentation of its performance and intended use, just like the function of model cards (Mitchell et al., 2019).

4. EVALUATION MODEL

All the artifacts generated by the generative AI component have to be scored in five dimensions before it can be accepted. The five dimensions are correctness, completeness, traceability, reproducibility and confidence. Correctness is a check which a generated logic is behaving correctly. Check of completeness verifies that the artifact has addressed all the requirement requirements. Traceability is used to determine whether or not the artifact can be traced back to a particular requirement and component. Reproducibility also ascertains consistency of output of the same input when the process is repeated. Confidence shows confidence signal of a model as well as accuracy in past in similar tasks.

Equation 1 is used to come up with the overall trust score of an artifact.

$$T = w_1C + w_2C_m + w_3T_r + w_4R_p + w_5C_f \quad (1)$$

T is the sum of trust scores, C is the correctness score, C_m is the completeness score, T_r is the traceability score, R_p is the reproducibility score, C_f is the confidence score, w_1 , w_2 , w_3 , w_4 and w_5 are the weights such that they all add up to 1. The artifact types can be weighted differently, as the weight of a defect report generated by the system could be biased towards correctness and the weight of a test case generated by the system may be biased towards reproducibility.

The mean scores during the assessment exercise as outlined in Section 7 have been tabulated in Table 1, by artifact type. The same data are in the form of a chart in figure 2.

Table 1. An average score on a scale of 0 to 100 on four forms of created quality engineering artifact derived by evaluating the artifact based on the measures of correctness, completeness, traceability and reproducibility.

Artifact Type	Correctness	Completeness	Traceability	Reproducibility
Test cases	82	76	71	88
Defect summaries	74	69	65	79
Technical documentation	88	83	60	91
Engineering answers	70	62	55	74

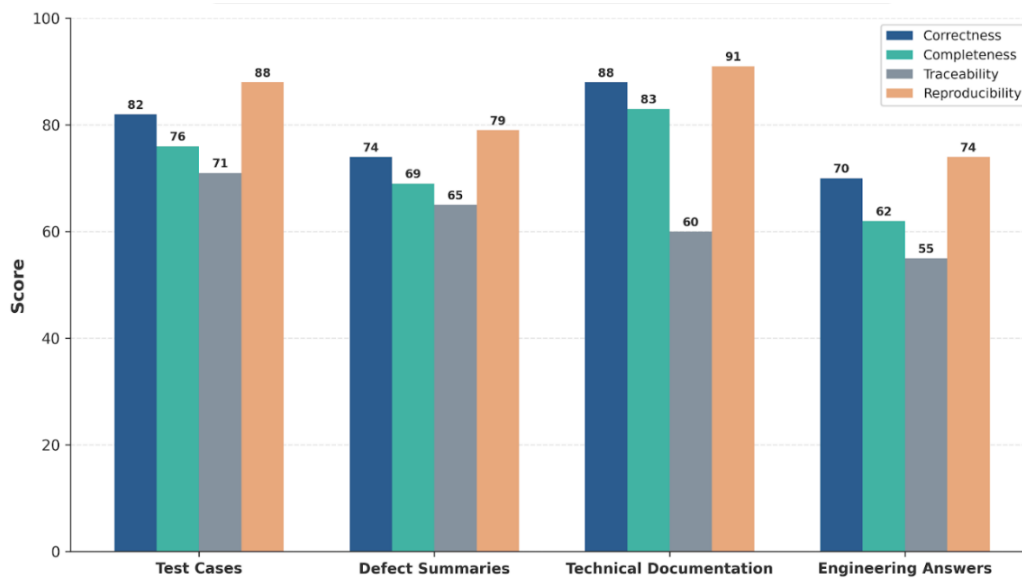


Figure 2: Comparison of the scores of the correctness, completeness, traceability and reproducibility of the four types of artifacts generated.

Table 1 has a trend that is in accordance with previous studies. Correctness and reproducibility (documentation) have the highest score since it utilizes stable code structure thereby confirming the prior observation that language models trained on code-natural language pairs are good at documentation generation (Feng et al., 2020). The test cases and engineering responses have lower grades on traceability which is what is intended to be bridged by the provenance layer proposed in this paper.

5. Requirement Traceability Mechanisms

A score in evaluation will not provide an evaluator with the source of an artifact. This is solved by the provenance layer which has four links to each generated artifact. These links relate the artifact to the requirement which caused it, the component of the application it interacts with, a previous implementation of a similar artifact, and the validation outcome when the artifact is run or read. This requires a design that closely resembles the interaction log-based approach, in that it also correlates unstructured requirements in an issue tracker to source code changes with accuracy greater than 90 percent (Hübner & Paech, 2020).

The precision and recall are used to determine the quality of traceability indicated in Equation 2.

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN} \quad (2)$$

In which P is precision, R is recall, TP is the correctly identified links in the trace, FP is the incorrect links in the trace identified and FN is the ones that the system did not identify. This loss of accuracy could happen with a traceability approach that eliminates grammatical categories in requirement text prior to matching and thus maintaining the entire requirement text intact during matching instead of filtering it (Ali et al., 2018).

A list of four of the traceability techniques that were taken into account in the provenance layer design, and with their strengths and limitations is given in Table 2.

Table 2. Four requirement traceability methods are discussed as the ones to be considered to use in the provenance layer and their essence/method and general limitation are qualitatively compared.

Technique	Core Method	Main Limitation
Interaction log linking	Logs the activity of the track's developer	Needs discipline of commitment within the teams
Constraint based pruning	Using pruning rules, applies a part of speech filtering	Would be sensitive to style of writing
Quality attribute scenario linking	Makes use of non-functional requirements structured templates	Poor coverage of functional requirements not of a business character
Ontology based linking	Formally depicts the types of requirements and relationships	Needs requires in-advance modeling to be used

The structure of this paper builds on a combination of an interaction log linking and ontology-based linking because requirements model generation process reveals that a modelled relationship model assists in alerting conflicts among the requirements prior to carrying out the implementation process (Philip et al., 2017). Ontology-based approach is also suitable to agile environment, since it learns about the kinds of requirements as the project progresses as opposed to depending on specific set of documents initially (Murtazina & Avdeenko, 2019).

The precision of traceability and recall varied with five review iterations with the pilot study as demonstrated in Figure 3.

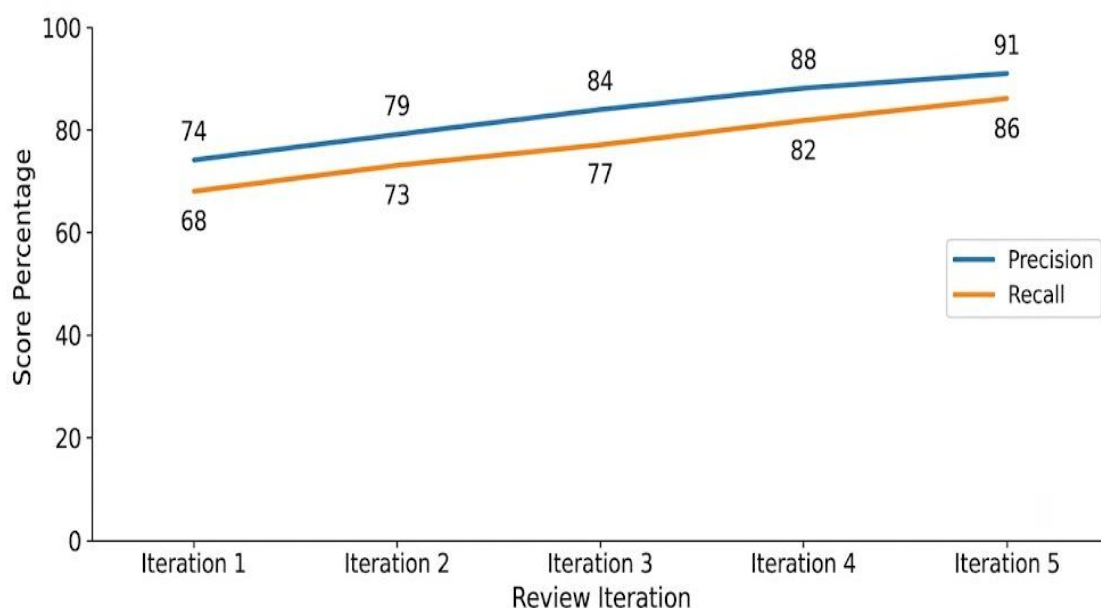


Figure 3: Line chart depicting traceability accuracy and recall values of 5 review iterations done in a row of the pilot evaluation.

6. Human Oversight and Risk Based Review

All the created artifacts do not have the same significance of risk. A test case that is generated in order to test a low priority reporting screen does not require such scrutiny as those of a generated test case in order to test a payment processing module. All the artifacts are assigned a risk score by the framework with the help of Equation 3.

$$R = \frac{I \cdot U}{C_f} \quad (3)$$

The risk score is denoted by R, business impact rating of the part of the business affected is denoted by I, and uncertainty rating is denoted by U (depending on the confidence score in Equation 1) where C_f is the confidence score. An increased impact and uncertainty increase the risk score upwards and confidence score decreases it respectively.

The extent of the decision on the review is based on a set of three risk bands. When the risk score is low on the artifact, the artifact gets automatically approved and records a periodical audit. Artifacts with a medium score on the risk score are automatically approved and embedded in the next planned human review cycle. Those artifacts that have a high-risk scoring are placed in custody until the approval by a qualified reviewer. This hierarchy shows the issue in the state of research in defect prediction when accurate but not explainable nor actionable predictions are not trusted by practitioners (Tantithamthavorn & Jiarpakdee, 2021).

Table 3 presents the results of the distribution of the generated artifacts by these three risk bands during the pilot study, and the outcome of the review.

Table 3. Generation of distribution plotting of generated artifacts in three risk bands during pilot study indicating the outcome of review and the time to disposition (average).

Risk Band	Artifact Count	Review Outcome	Average Time to Disposition (hours)
Low	312	Auto approved	0.1
Medium	164	Auto approved, flagged for audit	0.4
High	58	Held for expert review	6.2

This same distribution is in the form of a chart as presented in Figure 4.

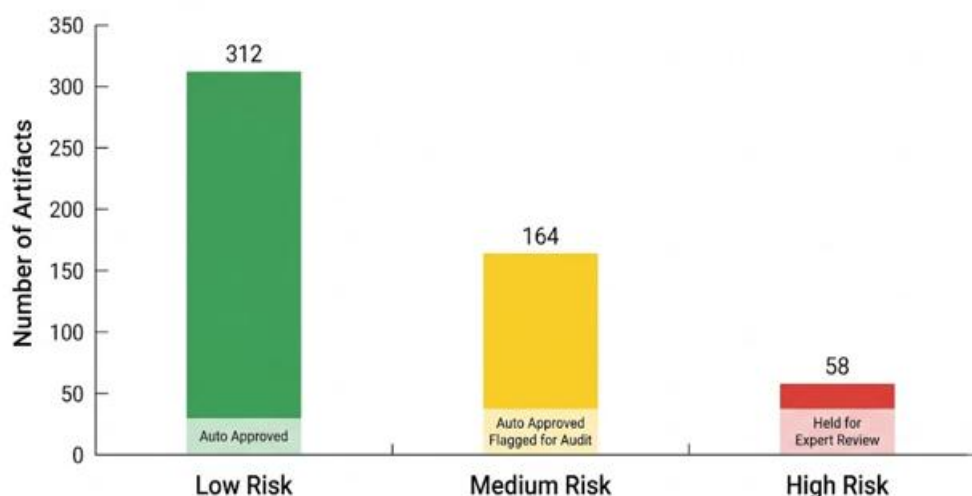


Figure 4: Bar chart that indicates the number of generated artifacts per risk band and the output (review) that was displayed at the pilot study.

7. EXPERIMENTAL SETUP AND RESULTS

In order to test the framework, some generated test artifacts were created of four modules within a sample enterprise application, which comprised of authentication, processing of orders, management of inventory and reporting. The contents of each of the modules were the generate AI element, which generated test cases against the requirements of each of the modules and the test cases were run against the application build. This arrangement is similar to the search-based test generation methodology of RESTful web services that modeled test actions as sequences of calls sent to it, and claimed high coverage improvement compared to baseline techniques (Rojas et al., 2016).

A summary of the results of each of the modules (numbers of generated test cases, pass rate and the number of defects detected) is given in table 4.

Table 4. Four application modules with results of an experiment that express the number of generated test cases, pass rate and the number of defects revealed by the generated test cases.

Module	Generated Test Cases	Execution Pass Rate (%)	Defects Found
Authentication	145	91	9
Order processing	210	85	17
Inventory management	168	88	11
Reporting	96	94	4

The least pass rate, and highest number of defects was found in order processing. This result is also expected as the highest business impact rating in the risk model was on this module and therefore, it does not contradict its business impact. This module would be better covered by a weighted search strategy as it also allows optimization concurrently over all goals, as opposed to one goal at a time as worked by the whole test suite generation approach (Rojas et al., 2016). Another approach that may result in increased defect rate during high-risk modules like order processing would be a fitness function that is tuned by using Improved Combined Fitness approach (Sahoo & Ray, 2019).

Figure 5 provides a comparison of the defect rate obtained by the test cases generated against the defect rate obtained during interventional tests of the modules in previous release cycle which is used as a manual baseline.

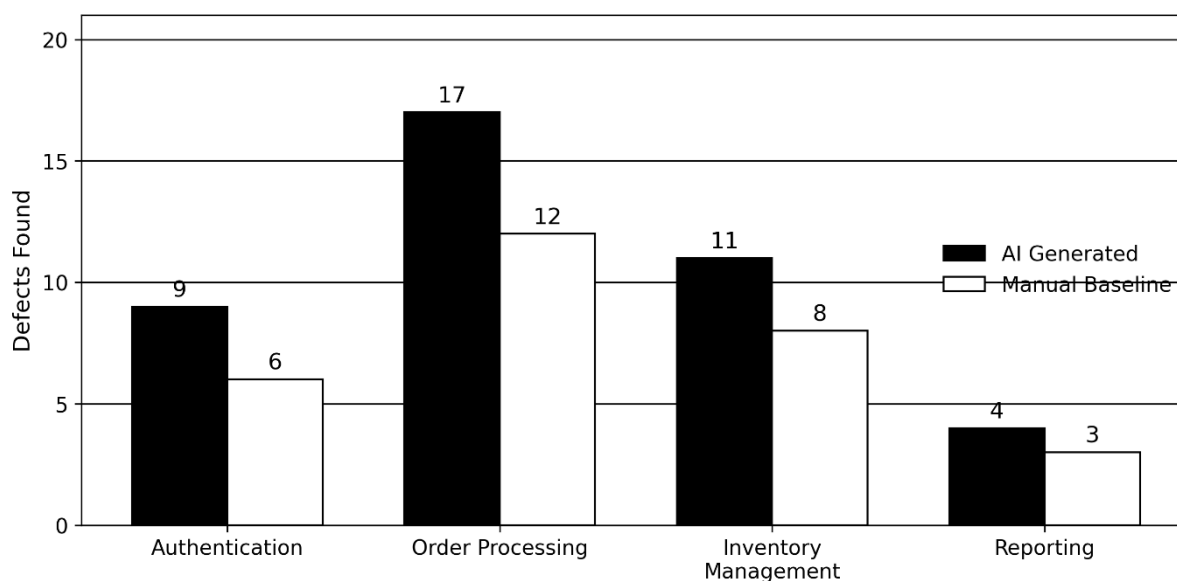


Figure 5: The rate of defect detection using the aid of AI generated test cases as compared to a manual testing baseline in the four modules under evaluation.

The general trend of this experiment is in line with previous results of generating codes. An ensemble sampling of a language model, instead of a single iteration of sampling with an attempt to generate a piece of code, increases the proportion of successful samples significantly during code synthesis tasks (Chen et al., 2021). The framework utilises the same principle by creating various candidate test cases in relation to the requirement and only successful candidate passing through the evaluation model.

8. GOVERNANCE CONSIDERATIONS

The Generative AI model uncertainty has to be follow-up over time, as the same model, doing well on a release, may suck on a subsequent release. Lack of consistency in the repeated runs undermines reproducibility, and reduces the trust score of Equation 1. The created artifacts are also prone to sensitive enterprise data in case requirement text or code snippets get through external model services without the necessary controls. Another open question is accountability, as even a defect that is not detected by one of the generated test cases has to have an explicit embodiment of a failure owner.

The qualitative variant of these governance themes is shown in Table 5 that outlines each of the concerns and optimal control.

Table 5. Thematic summarization of governance related to the adoption of generative AI in quality engineering of enterprises based on the chosen themes with a recommended control recommended around each theme.

Governance Theme	Description	Recommended Control
Model uncertainty	The accuracy of the models may change according to releases/domains.	Periodical recheck of a set benchmark.
Output inconsistency	The same input should not be repeated as this may not always give the same results.	Scoring reproducibility within the evaluation model.
Sensitive information exposure	Code-Text can cross enterprise boundaries.	Data masking
Accountability for decisions	A defect that was not detected is because of a lack of clarity in ownership when an artifact is generated.	Name of review sign off dependent in provenance layer.

9. DISCUSSION

The outcomes of this paper demonstrate that generative AI has the potential to improve the work of the quality engineering at a significant scale, yet only in a case when its results undergo some organization checks. Evaluation model provides a numeric basis on which one can compare the types of artifacts. Provenance layer provides an answer to the question of the provenance of an artifact and cannot be determined only by the generated text block. The review process, which is risk based, ensures that the human attention is concentrated on those artifacts that are the most important and examine the artifact evenly on both low risk and high-risk artifacts.

Section 7 had four modules in controlled sample; therefore, the results must be interpreted as an example of the framework as opposed to general performance. True enterprise applications come with an increased number of modules, greater amounts of legacy code and less consistent requirements writing styles that will reduce the accuracy of traceability than what is found in this report. Nonetheless, the direction taken by the findings is consistent with an increasing amount of research on AI based systems where testing and quality are the most pursued research questions whereas issues related to maintenance continue to be less popular in research (Martinez-Fernandez et al., 2021).

10. CONCLUSION

This article provided a roadmap to incorporating generative AI within the process of enterprise software quality engineering in a manner that helps build trust (as opposed to believe in it). The model is a blend of a framework which has built in assessment type model, provenance layer and traceability layer, as well as, a risk oriented human monitoring model. Each generated artifact is assigned a certain score by the evaluation model in terms of correctness, completeness, traceability, and reproducibility, as well as, confidence. The provenance layer links up all artifacts to the need and evidence.

REFERENCES

- [1] Ali, N., Cai, H., Hamou-Lhadj, A., & Hassine, J. (2018). Exploiting Parts-of-Speech for effective automated requirements traceability. *Information and Software Technology*, 106, 126–141. <https://doi.org/10.1016/j.infsof.2018.09.009>
- [2] Chen, M., Tworek, J., Jun, H., Yuan, Q., De Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., . . . Zaremba, W. (2021). Evaluating large language models trained on code. *arXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.2107.03374>
- [3] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *CodeBERT: A Pre-Trained Model for Programming and Natural Languages*, 1536–1547. <https://doi.org/10.18653/v1/2020.findings-emnlp.139>
- [4] Hübner, P., & Paech, B. (2020). Interaction-based creation and maintenance of continuously usable trace links between requirements and source code. *Empirical Software Engineering*, 25(5), 4350–4377. <https://doi.org/10.1007/s10664-020-09831-w>
- [5] Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., Vollmer, A. M., & Wagner, S. (2021). Software Engineering for AI-Based Systems: A Survey. *arXiv (Cornell University)*. <https://doi.org/10.1145/3487043>
- [6] Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). Model Cards for Model Reporting. *Model Cards for Model Reporting*, 220–229. <https://doi.org/10.1145/3287560.3287596>
- [7] Murtazina, M., & Avdeenko, T. (2019). An ontology-based approach to support for requirements traceability in agile development. *Procedia Computer Science*, 150, 628–635. <https://doi.org/10.1016/j.procs.2019.02.044>
- [8] Philip, M. M., Varma, R., & Balakrishn, V. (2017). Requirements traceability model generation for evolving software systems. *Journal of Software Engineering*, 11(3), 282–289. <https://doi.org/10.3923/jse.2017.282.289>
- [9] Pokharel, S., & Reza, H. (2019). Toward the design and implementation of Traceability Engineering tool support. *Journal of Software Engineering and Applications*, 12(06), 249–265. <https://doi.org/10.4236/jsea.2019.126015>
- [10] Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, 25(6), 5193–5254. <https://doi.org/10.1007/s10664-020-09881-0>
- [11] Rojas, J. M., Vivanti, M., Arcuri, A., & Fraser, G. (2016). A detailed investigation of the effectiveness of whole test suite generation. *Empirical Software Engineering*, 22(2), 852–893. <https://doi.org/10.1007/s10664-015-9424-2>
- [12] Sahoo, R. R., & Ray, M. (2019). PSO based test case generation for critical path using improved combined fitness function. *Journal of King Saud University - Computer and Information Sciences*, 32(4), 479–490. <https://doi.org/10.1016/j.jksuci.2019.09.010>
- [13] Tantithamthavorn, C. K., & Jiarapakdee, J. (2021). Explainable AI for software engineering. *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 1–2. <https://doi.org/10.1109/ase51524.2021.9678580>

- [14] Thiebes, S., Lins, S., & Sunyaev, A. (2020). Trustworthy artificial intelligence. *Electronic Markets*, 31(2), 447–464. <https://doi.org/10.1007/s12525-020-00441-4>
- [15] Tian, Y., University of Virginia, Pei, K., Columbia University, Jana, S., Columbia University, Ray, B., & University of Virginia. (2018). DeepTest: Automated testing of Deep-Neural-Network-driven autonomous cars. *ICSE '18: 40th International Conference on Software Engineering*, 12 pages. <https://www.cs.columbia.edu/~suman/docs/deeptest.pdf>
- [16] Zhang, M., Marculescu, B., & Arcuri, A. (2021). Resource and dependency based test case generation for RESTful Web services. *Empirical Software Engineering*, 26(4). <https://doi.org/10.1007/s10664-020-09937-1>