Comparative Empirical Evaluation of Software Test Automation using Artificial Intelligence and Manual Test Case Design

Jesús José Bone Caicedo, José Luis Carvajal Carvajal, and Victor Xavier Quiñonez Ku

***Abstract*—This study presents an empirical comparison between manually designed test cases and those automatically generated by generative artificial intelligence tools—ChatGPT and Diffblue Cover—applied to the Spring PetClinic system, developed in Java and Spring Boot. The fully automated experiment comprised 2,480 runs distributed across 12 test classes (6 human and 6 AI-generated), with 40 iterations per class and a total duration of 6.18 hours. Four main metrics—instruction coverage, branch coverage, mutation score, and execution time—were evaluated using descriptive and inferential analysis (Student's t, Welch, and Mann-Whitney U). At the aggregate level (N = 12), no significant differences were found ($p > 0.05$; $d < 0.30$), while at the complete level (N = 2,480), small effects ($r < 0.15$) were observed in all metrics, indicating marginal differences between the two approaches. The results confirm that generative AI can achieve quantitative performance comparable to that of human testing, albeit with more limited functional reasoning. This work provides empirical evidence on the current capabilities and limitations of AI in automated testing, highlighting its potential to accelerate repetitive tasks and improve productivity without replacing human analytical judgment.**

***Index Terms*—automated testing, Diffblue Cover, generative AI, software quality, test case generation**

I. Introduction

In the context of digital transformation and the growing incorporation of artificial intelligence in all stages of the software life cycle, test automation has taken center stage in agile development processes due to its potential to increase coverage, reduce time, and improve product reliability. This technical advance not only responds to a need for efficiency in engineering teams, but also to social demand for more reliable, secure, and highly available digital services, whose quality has a direct impact on sensitive sectors such as health, education, and finance. The emergence of generative AI—with large language models (LLMs) and associated tools—has renewed this field by promising faster and assisted test case generation, albeit with mixed and context-dependent results [1], [2], [3], [4]. In ecosystems dominated by Java and Spring Boot, it is pertinent to subject these solutions to rigorous evaluation to establish their true scope and limitations compared to traditional approaches [1], [2].

Despite recent advances (e.g., improvements in LLM-assisted unit testing in industrial contexts) [5], the literature shows that the landscape is still fragmented. Research such as that by Schäfer et al. [6] and Yang et al. [7] has demonstrated the ability of LLMs to generate automatic unit tests with promising results, but without establishing direct comparisons with human tests. Complementarily, Bhatia et al. [8] empirically evaluated the performance of ChatGPT against traditional tools such as Pynguin, showing comparable or superior coverage, but also a significant proportion of incorrect assertions and dependence on human supervision. Similarly, Kanth et al. [9] contrast conventional methodologies and AI-based approaches, focusing their analysis on efficiency and coverage, although without strict control over a common system under test (SUT). For their part, Kirinuki and Tanno [10] evaluated the interaction between ChatGPT and human testers in black-box testing, highlighting qualitative differences in reasoning and understanding of the system. However, recent systematic and grey literature reviews, such as that by Ricca et al. [11], agree that there is still a lack of empirical evidence combining standardized quantitative metrics—such as mutation score or coverage—with functional observations in the same experimental environment. While recent works have explored advanced metric-driven approaches, such as mutation-guided test generation [12], these efforts remain largely focused on quantitative effectiveness rather than functional interpretation.

In this context, an empirical gap persists: there is a lack of systematic and quantitative comparisons between AI-generated tests and manually designed tests under the same SUT and replicable evaluation framework [2]. Although the literature reports benefits—faster generation and automated exploration—it also warns of limitations associated with variable quality, false positives, and dependence on human supervision [13], [14], [15]. This combination of advances and gaps justifies the need for controlled studies that contrast the actual effectiveness of both approaches under equivalent conditions.

In addition to traditional quantitative indicators, this study incorporates a complementary observation related to functional understanding, understood in empirical terms as the differences observed in the way the compared approaches—generative AI and manual design—address the logic and functionality of the system under test. This dimension does not seek a formal cognitive analysis, but rather to contextualize the numerical results (coverage, mutation score, efficiency) based on how each approach interprets and structures the test cases [2], [4], [14], [15]. Its inclusion enriches the reading of the findings, offering a more complete perspective on the performance and functional quality of the automated testing

J. J. Bone Caicedo, J. L. Carvajal Carvajal, and V. X. Quiñonez Ku are with the Systems and Software Department, Pontifical Catholic University of Ecuador Esmeraldas Campus, Esmeraldas Ecuador (e-mail: jjbone@pucese.edu.ec; jose.carvajal@pucese.edu.ec; xavier.quinonez@pucese.edu.ec).

process.

This study aims to evaluate, in a controlled environment, the effectiveness of test cases automatically generated by generative AI compared to those manually created in Java projects, using a representative Java/Spring Boot project as the system under test (SUT). The overall objective of the study is to evaluate the effectiveness and functional understanding, from an empirical approach, of AI-generated test cases compared to those designed by humans, combining quantitative metrics—coverage (instruction/branch), mutation score, and operational efficiency—with empirical observations on their ability to adequately represent the functionality of the system [16].

The expected contribution integrates comparative evidence with replicable metrics and an analytical component that considers the empirical perspective of functional understanding, offering useful inputs for decision-making in educational and professional settings [1], [5]. In line with this technical and social relevance, the research adopts a comparative experimental approach, based on objective metrics and empirical observations, which allow for the examination of both the quantitative performance and the interpretive quality of AI-generated tests compared to manual tests in a controlled environment. Finally, the article is structured as follows: section 2 describes the materials and methods; section 3 presents the results and discussion; and section 4 develops the conclusions and future projections.

II. Materials and Methods

A. Research Design

The study was developed under a mixed-method empirical-comparative design, aimed at evaluating the effectiveness and functional understanding of test cases generated by generative artificial intelligence versus those developed manually. An applied approach was adopted, focused on measuring the actual performance of automatic verification techniques in a software quality assurance context. The experiment was conducted between May and December 2025 in a controlled academic environment, using the open-source Spring PetClinic project as the system under test (SUT) due to its modular structure, stability, and widespread use in empirical research [2]. Two test suites were generated from this SUT: one using generative AI tools (ChatGPT Plus and Diffblue Cover CLI) and another manually created by the researcher. Both were evaluated under equivalent conditions using standardized quantitative metrics (code coverage, mutation score, and execution time) and a qualitative analysis aimed at understanding the functional aspects of the system.

The methodological design followed contemporary guidelines for empirical studies with language models, which recommend documenting versions, configurations, prompts, and interactions to ensure transparency, reproducibility, and traceability [17]. Recent guidelines on the role of LLMs in software engineering research were also incorporated, emphasizing the importance of maintaining human oversight, explicitly declaring the use of AI-based tools, and mitigating the methodological risks associated with their use [18]. These principles made it possible to control external variables, strengthen the internal validity of the study, and ensure that the results were verifiable and replicable through the use of professional tools in controlled and documented versions.

B. Materials and Experimental Environment

The study was conducted entirely on the Spring PetClinic system (Java 17, Spring Boot 3.5.6, 3.5.0-SNAPSHOT), selected as the sole system under test (SUT) due to its adoption in recent research, its modular architecture, and its compatibility with automation frameworks in Java [2]. Since the research did not involve human subjects, the researcher's intervention was limited to the design and technical execution of manual tests, while the artifacts analyzed corresponded exclusively to test cases generated by artificial intelligence and manually designed tests under controlled conditions. The manual tests were developed using functional exploration with Postman and coded in JUnit 5 and MockMvc, while the AI-generated cases were obtained using ChatGPT (GPT-5 model) and Diffblue Cover CLI (Teams Edition), the latter executed without human intervention to ensure neutrality in bytecode-based generation. Additionally, automation scripts in PowerShell were used to sequentially and reproducibly execute the functional, unit, and combined test suites, avoiding manual intervention during execution.

Quantitative metrics were collected using JaCoCo 0.8.12 (instruction and branch coverage), PITest 1.16.0 (mutation score), and Maven Surefire 3.2.5 (total execution time), with automatic export of results in XML, CSV, and HTML formats. Python 3.12.6 was used for statistical analysis with the pandas, numpy, scipy, matplotlib, seaborn, and openpyxl libraries, allowing the calculation of descriptive measures, normality and variance contrasts, hypothesis tests (Shapiro-Wilk, Levene, Student's t-test, Mann-Whitney U) and effect sizes (Cohen's d and r). All runs were performed on a computer with an Intel Core i7-12700KF processor, 32 GB DDR4 RAM, and 1 TB NVMe SSD, running Windows 11 Pro 23H2, without parallel loads and with fixed random seeds to ensure determinism and reproducibility, in accordance with recent guidelines for AI-assisted empirical studies in software engineering [11], [17].

The experimental sample consisted of test suites produced using both approaches: functional and unit tests generated by AI (ChatGPT and Diffblue Cover) and equivalent tests designed manually on the same SUT components. This configuration allowed for an objective comparison of coverage, mutation, and operational efficiency metrics, complemented by qualitative observations on the functional consistency of the cases. All configurations, parameters, versions, and resulting artifacts were documented in a supplementary technical record to facilitate future replication of the experiment.

C. Experimental Protocol

The experimental protocol was developed in five consecutive phases, with the aim of ensuring methodological consistency, variable control, and reproducibility. In the planning phase, an initial literature review was conducted on artificial intelligence-assisted test automation techniques, and the study's comparison criteria were defined based on the selected

metrics: instruction coverage, branch coverage, mutation score, and execution time. At the same time, the system under test (Spring PetClinic) was selected as the base environment for the empirical evaluation of manual and AI-generated approaches.

During the preparation phase of the experimental environment, the SUT was configured and the necessary measurement instruments for collecting metrics (PITest and Surefire) were installed, while JaCoCo was already part of the original project configuration. Subsequently, test cases were generated for the three approaches analyzed: (i) tests generated by ChatGPT GPT-5 using neutral prompts and providing only the source code of the method; (ii) unit tests automatically generated by Diffblue Cover CLI, without human intervention; and (iii) manual tests designed by the researcher with JUnit 5 and MockMvc, derived from the inspection of the source code and the behavior observed during the previous functional exploration with Postman.

The automated execution phase of the experiment was implemented using three PowerShell scripts, whose internal names were kept unchanged, supplemented with a brief description for clarity. The run_pitest_isolated_complete.ps1 script (responsible for executing all unit tests for both approaches) performed 40 consecutive iterations, recording the metrics from JaCoCo, PITest, and Surefire in each one. Similarly, the run-test-metrics.ps1 script (responsible for functional testing) ran 40 additional iterations under the same measurement conditions. Finally, the run_all_metrics_simple.ps1 script acted as an orchestrator, coordinating the sequential execution of the two previous scripts and recording the overall time of the process. In all cases, three preliminary runs (warm-up runs) were included to stabilize the JVM and mitigate the variability associated with the initial startup.

During the recording and organization of results phase, the outputs generated by each iteration—including XML files, CSV files, and HTML reports—were structured into separate directories by test type (functional or unit) and by approach (AI or manual), maintaining consistent names and fixed paths to ensure traceability. No human intervention was allowed during automated execution, and execution parameters, seeds, paths, and configurations were kept constant in order to minimize experimental noise and reinforce the internal validity of the study.

In the final phase of statistical analysis, the collected data were processed in Python using the pandas, numpy, scipy, seaborn, and matplotlib libraries. Descriptive measures (mean, median, standard deviation) were calculated, and statistical assumptions were verified using the Shapiro-Wilk (normality) and Levene (homogeneity of variances) tests. Depending on the fulfillment of these assumptions, parametric (Student's t-test or Welch's correction) or nonparametric (Mann-Whitney U) significance tests were applied. Effect sizes (Cohen's d and r) were also calculated, and graphical representations were generated to support the results obtained and facilitate their comparative interpretation.

The complete flow of the experimental protocol is presented in Fig. 1.

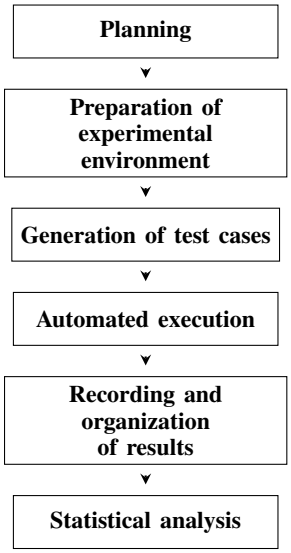

Fig. 1. Workflow representing the experimental protocol followed during the study.

D. Variables and Metrics

The study defined a set of independent and dependent variables aimed at comparing the behavior of test cases generated using artificial intelligence with those designed manually. The independent variable (IV) corresponded to the test generation approach, established with two levels: (i) manual generation and (ii) generation assisted by generative AI. The dependent variables (DV) were the results obtained from the quantitative and qualitative metrics used to evaluate the performance of both approaches, according to criteria identified in recent evaluations of AI-assisted testing tools [2], [11].

Among the quantitative variables, three main metrics were considered. Code coverage was evaluated using the indicators instruction coverage and branch coverage, in order to measure the percentage of instructions and paths executed by each test suite. The robustness of the test suite was measured using the mutation score, a widely used indicator for estimating the ability of tests to identify intentional modifications in the source code [2]. Finally, operational efficiency was defined as the total execution time of each test suite under controlled conditions, allowing for a comparison of the time load associated with each approach.

In addition to these quantitative metrics, a qualitative variable called functional comprehension was incorporated, related to the degree of consistency with which each approach represents the internal logic of the system under test. This dimension made it possible to contrast the structural and semantic adequacy of the test cases generated, taking into account recent findings on inconsistencies, omissions, or functional deviations observed in tests generated using language models [10], [13]. The conceptual structure linking the independent variable with the dependent variables, both quantitative and qualitative, is summarized in Fig. 2.

E. Data Analysis Methods

The data was analyzed using a quantitative and inferential approach, aimed at comparing the behavior of test cases gen-

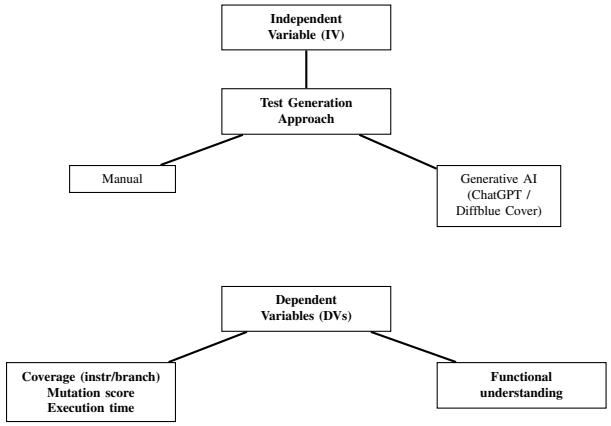

Fig. 2. Conceptual diagram showing the relationships between the independent variable (test generation approach) and the dependent variables (quantitative and qualitative metrics).

erated by artificial intelligence with those designed manually. The metrics obtained—instruction coverage, branch coverage, mutation score, and execution time—were processed and organized into comparative tables, accompanied by statistical visualizations such as bar charts, box plots, and violin plots. These analyses were developed using Python and the pandas, numpy, matplotlib, and seaborn libraries.

First, descriptive statistics were applied to calculate measures of central tendency (mean and median) and dispersion (standard deviation) for each metric evaluated. Subsequently, an inferential analysis was performed to identify statistically significant differences between the test generation approaches. To this end, the assumption of normality was verified using the Shapiro-Wilk test and the homogeneity of variances using the Levene test.

The selection of statistical tests depended on the fulfillment of these assumptions. When the metrics met normality and presented homogeneous variances, the classic Student's t-test for independent samples was applied. In cases where normality existed but variances were not homogeneous, Welch's t-test was used. When the metrics did not meet the normality assumption in any of the groups, the nonparametric Mann-Whitney U test was used. In all cases, a significance value of $p < 0.05$ was adopted, accompanied by 95% confidence intervals.

Finally, effect sizes were calculated to complement the statistical interpretation. Cohen's d was used in comparisons made with Student's t-test, and the r measure was used for comparisons based on the Mann-Whitney U test. The combination of descriptive statistics, parametric and nonparametric tests, and effect size calculations ensured a robust, reproducible analysis consistent with best methodological practices in empirical software engineering studies [2], [11], [15]. The decision flowchart used to select the appropriate statistical test is shown in Fig. 3.

F. Validity and Reliability Control

Internal validity was ensured by conducting the experiment in a controlled environment, keeping hardware, software, and

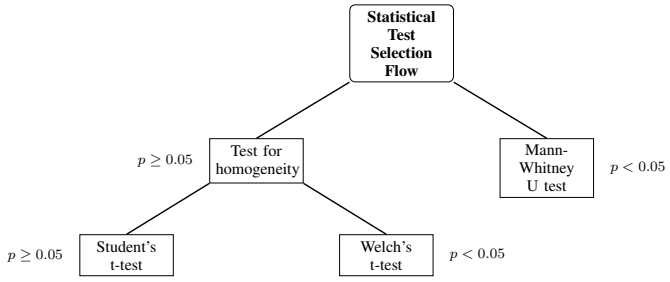

Fig. 3. Decision flowchart for selecting the appropriate statistical test based on normality and variance homogeneity assumptions.

system configuration conditions constant. The measurement tools—JaCoCo, PITest, and Surefire—operated autonomously with respect to the origin of the test cases, which made it possible to avoid instrumental biases and ensure a partially blind evaluation. The source code of the system under test was not modified, and the original tests of the project were excluded, preserving the experimental integrity. To reinforce construct validity, widely accepted metrics in software engineering—instruction coverage, branch coverage, and mutation score—were used, identified as consistent indicators for evaluating the quality and effectiveness of AI-assisted testing in recent studies [2], [11], [15], and consistent with contemporary methodological guidelines for empirical research with generative models [17].

In terms of reliability, controls were implemented to minimize researcher influence. The prompts used in ChatGPT were formulated in a neutral manner, without examples or guidelines, and the tests generated by Diffblue Cover and those designed manually were developed completely independently, avoiding code transfers or adaptations between approaches. Errors or failures observed during execution were recorded without subsequent corrections, ensuring traceability and consistency. External validity was supported by the use of Spring PetClinic, a representative and widely adopted system in empirical evaluations of testing tools for Java/Spring Boot environments, facilitating replication of the study and reinforcing the comparability of the results obtained.

G. Reproducibility and Ethics

The research was conducted under principles of transparency and traceability, ensuring that the experiment could be replicated in its entirety. All scripts, configurations, execution parameters, and intermediate results were documented and stored in a repository derived from the Spring PetClinic project, maintained under version control and structured to allow complete reproduction of the experimental flow. The system under test was used in accordance with its open source license (Apache 2.0) and without modifications to its source code, ensuring the consistency of the environment and the stability of the evaluation conditions. Likewise, the tools used—including ChatGPT and Diffblue Cover—were executed in accordance with their respective terms of use, avoiding any manipulation or intentional bias in the results.

From an ethical perspective, the study did not involve human participants or the processing of personal data, and

therefore did not require informed consent procedures or review by an ethics committee. However, a strict commitment to academic integrity, transparent communication of findings, and the absence of selective exclusion of data was maintained. Generative artificial intelligence was used with neutral prompts and without interventions intended to influence the results, so that the test sets produced reflected the genuine behavior of each tool.

To ensure complete reproducibility of the study, the repository used during the experiment—containing all scripts, configurations, and results—is publicly available at: https://github.com/solveighty/spring-petclinic-test/

III. Results

A. Experiment Execution and Dataset Structure

The experimental execution was carried out using a fully automated process, designed to ensure consistency and eliminate any human intervention during data collection. Forty iterations were performed for each of the 12 test classes considered in the study—six designed manually and six generated using artificial intelligence tools (ChatGPT and Diffblue Cover)—yielding a total of 480 base measurement cycles. The entire execution process took 6.18 hours, considering the valid iterations and preliminary executions used to stabilize the JVM. Each cycle generated two separate records, corresponding to the compilation and execution phases, resulting in a final set of 2,480 raw records. For each record, four metrics were automatically stored: instruction coverage, branch coverage, mutation score, and execution time, extracted directly from the XML, CSV, and HTML artifacts generated by JaCoCo, PITest, and Surefire. Table I provides a summary of the experimental dataset structure.

TABLE I
SUMMARY OF THE EXPERIMENTAL DATASET COLLECTED DURING THE STUDY.

Component	Cant.	Details
Total test classes	12	6 Manual + 6 AI-Generated
Iterations per class	40	Includes warm-up (3 initial iterations)
Total raw records	2,480	1,600 manuals and 880 AI
Metrics per record	4	instruction%, branch%, mutation%, time(s)

During the dataset integrity check, it was confirmed that the 2,480 expected measurements were generated correctly, with no missing records, duplicates, or format inconsistencies. The identification labels for each approach (manual and AI) remained consistent across all iterations, and the exported files retained the expected structure for statistical processing. Mutation score values equal to 0 were observed in some cases; however, these correspond to valid runs in which the test sets failed to kill any mutations generated by PITest. All values were retained in their entirety due to their analytical relevance, ensuring the completeness, consistency, and reliability of the dataset used in the following stages of the study.

B. Preparing the Dataset

The raw records generated during the experimental run—originally stored in twelve CSV files corresponding to the manual and AI-generated test classes—were consolidated into a single master file, following the procedure described in the experimental protocol. This integration was performed using automated Python scripts (pandas), which included format normalization, column name standardization, and data type verification, fully preserving the metrics collected in each iteration and ensuring the structural consistency of the dataset.

For the subsequent stages of the analysis, two levels of aggregation were defined. Table II illustrates the structure and distribution of records at both aggregation levels.

TABLE II
TOTAL RECORDS FOR EACH TEST TEAM.

Analysis level	Manual	AI	Description
N = 2,480	1,600	880	Individual observations per iteration
N = 12	6	6	Independent averages by test class

The complete level (N = 2,480) retained all individual records, allowing the intrinsic variability of the measurements to be captured. The aggregated level (N = 12) was obtained by averaging the 40 iterations associated with each test class, generating a balanced and statistically independent set at the class level.

C. Descriptive statistics of the metrics

Descriptive statistics were used to characterize the overall behavior of the metrics evaluated for both approaches (manual and artificial intelligence), both at the aggregate level (N = 12) and at the complete level of individual records (N = 2,480). For each metric, mean, median, and standard deviation values were calculated to describe the central tendency and variability associated with each test group.

Table III presents the descriptive measures at the aggregate level, where each value represents the average of 40 iterations per test class.

TABLE III
AVERAGE OF 40 ITERATIONS PER TEST CLASS (MANUAL N = 6, AI N = 6).

Metrics	Group	Mean	Median	St Dev.
Instr Coverage	Manual	18.25	15.95	12.50
Instr Coverage	AI	17.67	16.13	11.36
Branch Coverage	Manual	14.58	11.88	12.62
Branch Coverage	AI	12.05	9.89	9.49
Mutation Score	Manual	18.52	15.28	17.71
Mutation Score	AI	14.76	15.11	11.61
Time Execution	Manual	0.082	0.071	0.069
Time Execution	AI	0.194	0.144	0.193

Table IV illustrates the descriptive measures at the complete level, reflecting the raw distribution across all 2,480 individual records collected during the experiment.

The differences between means and medians observed in some metrics indicate that the data do not follow a completely

TABLE IV
INDIVIDUAL RECORDS PER ITERATION (MANUAL N = 1600, AI N = 880).

Metrics	Group	Mean	Median	St Dev.
Instr Coverage	Manual	19.94	21.85	12.11
Instr Coverage	AI	14.20	10.69	8.71
Branch Coverage	Manual	17.69	12.50	12.39
Branch Coverage	AI	13.50	16.25	6.75
Mutation Score	Manual	22.92	16.67	17.19
Mutation Score	AI	16.63	19.44	8.16
Time Execution	Manual	0.079	0.012	0.178
Time Execution	AI	0.114	0.003	0.232

symmetrical distribution. This behavior is reflected at both levels of analysis and is reported for consideration in subsequent statistical stages.

D. Verification of statistical assumptions

Statistical assumptions were validated in order to determine the relevance of applying parametric or nonparametric tests in inferential analyses. Two fundamental conditions were evaluated: normality of distributions and homogeneity of variances, both considered standard requirements for the use of Student's t-test and its variants.

1) Normality (Shapiro-Wilk): The normality of the metrics was assessed using the Shapiro-Wilk test, applied both at the complete level of individual records (N = 2,480) and at the aggregate level by test class (N = 12). At the complete level, none of the distributions evaluated had p-values greater than 0.05, indicating an absence of normality. In contrast, at the aggregate level, all metrics showed p-values greater than 0.05 for both approaches, fulfilling the assumption of normality.

Table V presents the Shapiro-Wilk test results at the complete level (N = 2,480), showing normality assessment for all metrics across both manual and AI approaches.

TABLE V
SHAPIRO-WILK NORMALITY TEST (MANUAL N = 1600, AI N = 880; FULL LEVEL, N = 2,480).

Metrics	Group	W	p-value	Normality
Instr Coverage	Manual	0.8160	2.34e-39	No
Instr Coverage	AI	0.7524	2.66e-34	No
Branch Coverage	Manual	0.8178	3.21e-39	No
Branch Coverage	AI	0.8697	1.96e-26	No
Mutation Score	Manual	0.8404	2.46e-37	No
Mutation Score	AI	0.8313	1.82e-29	No
Time Execution	Manual	0.4797	8.07e-56	No
Time Execution	AI	0.5354	4.67e-43	No

Table VI shows the Shapiro-Wilk test results at the aggregate level (N = 12), demonstrating that all metrics meet the normality assumption at this level of analysis.

These results indicate that only the aggregate level meets the normality assumption, while the complete level requires the use of nonparametric tests as it does not conform to normal distributions.

TABLE VI
SHAPIRO-WILK NORMALITY TEST (MANUAL N = 6, AI N = 6; AGGREGATE LEVEL, N = 12).

Metrics	Group	W	p-value	Normality
Instr Coverage	Manual	0.9087	0.4281	Yes
Instr Coverage	AI	0.9038	0.3971	Yes
Branch Coverage	Manual	0.8807	0.2722	Yes
Branch Coverage	AI	0.9127	0.4541	Yes
Mutation Score	Manual	0.9132	0.4581	Yes
Mutation Score	AI	0.9720	0.9056	Yes
Time Execution	Manual	0.9296	0.5771	Yes
Time Execution	AI	0.8507	0.1596	Yes

2) Homogeneity of variances (Levene): The homogeneity of variances between the manual and AI groups was evaluated using Levene's test. In three of the four metrics analyzed—instruction coverage, branch coverage, and mutation score—values of $p > 0.05$ were observed, indicating equality of variances between groups. In contrast, execution time presented $p < 0.05$, which shows heterogeneity of variances and requires the application of Welch's correction in the parametric analysis.

Table VII displays the results of Levene's test for homogeneity of variances at the aggregate level (N = 12).

TABLE VII
LEVENE'S TEST FOR HOMOGENEITY OF VARIANCES (N = 12).

Metrics	F. Stat	p-value	Homogeneity
Instr Coverage	0.1006	0.7577	Yes
Branch Coverage	0.1262	0.7298	Yes
Mutation Score	0.3930	0.5448	Yes
Time Execution	6.3679	0.0302	No

E. Parametric analysis (Student's t and Welch)

Parametric contrasts were applied only at the aggregate level (N = 12), given that this level met the assumption of normality and, for three of the metrics, the assumption of homogeneity of variances. For metrics with homogeneous variances, the standard Student's t-test was used, while for the Time (s) metric, which did not meet homogeneity, the Welch correction variant was applied.

Table VIII presents the results of the parametric tests (Student's t and Welch) applied at the aggregate level.

TABLE VIII
RESULTS OF PARAMETRIC TESTS (T-STUDENT AND WELCH).

Metrics	p-value	Cohen's d	Sig.
Instr Coverage	0.9350	0.048	No
Branch Coverage	0.7022	0.227	No
Mutation Score	0.6727	0.251	No
Time Execution	0.2293	-0.769	No

Figure 4 associated with parametric contrasts presents box plots illustrating the distribution of each metric for the two approaches evaluated. Its purpose is solely to provide a descriptive visualization of the central behavior and dispersion of the aggregate values (N = 12). This representation allows us to

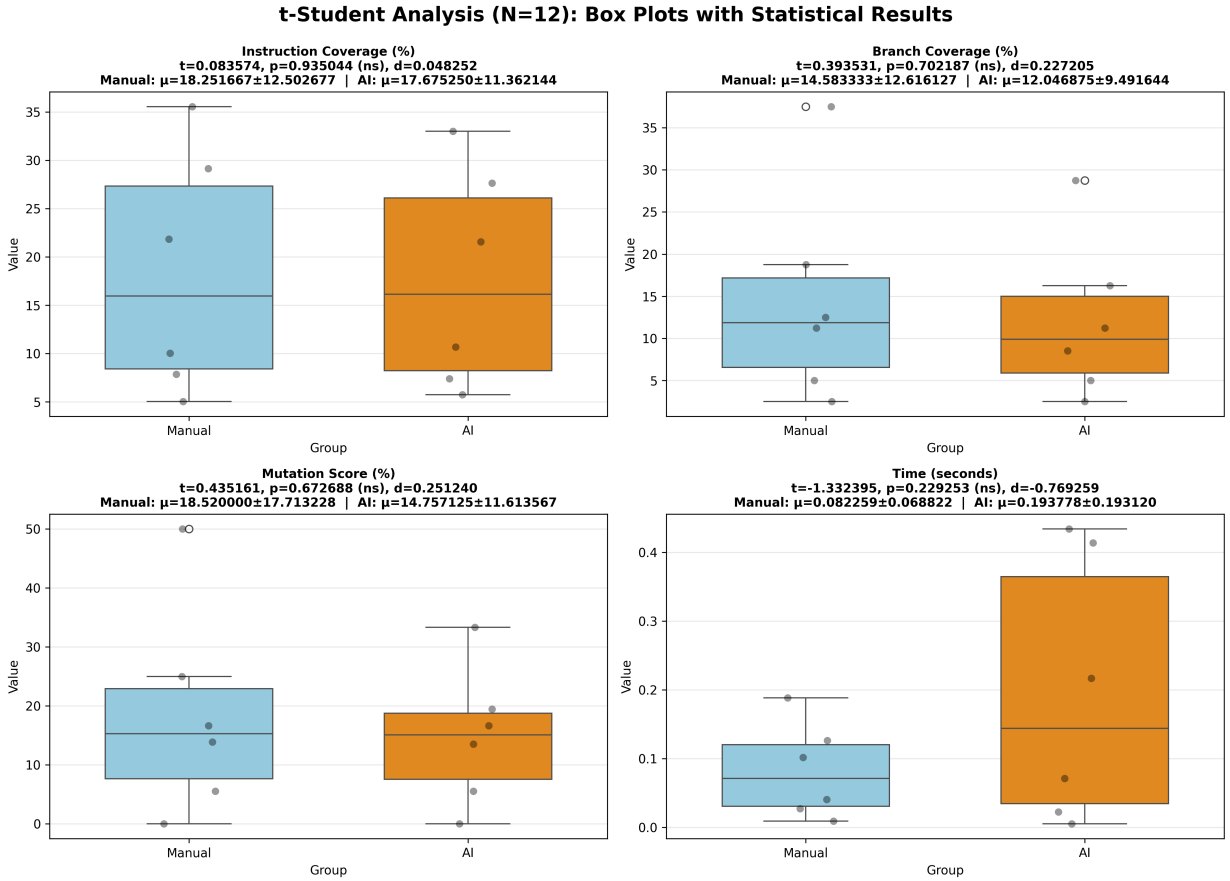

Fig. 4. Box plots from parametric analysis (t-Student and Welch) showing the distribution of metrics for both test approaches (Manual vs. AI) at the aggregated level (N = 12).

observe the overlap between groups, but it does not constitute additional evidence beyond the reported inferential results.

F. Non-parametric analysis (Mann-Whitney U)

Since the entire dataset (N = 2,480) did not meet the normality assumption, the nonparametric Mann-Whitney U test was applied to compare the distributions between the two approaches (manual and AI) for each metric. The contrasts were performed using the individual values per iteration, with the p-value and effect size r.

Table IX presents the results of the Mann-Whitney U test applied at the complete level (N = 2,480).

TABLE IX
RESULTS OF NON-PARAMETRIC TESTS (MANN-WHITNEY U).

Metrics	p-value	r	Sig.
Instr Coverage	3.20e-13	0.145	Yes
Branch Coverage	5.73e-10	0.123	Yes
Mutation Score	3.82e-03	0.057	Yes
Time Execution	3.74e-05	0.082	Yes

Figure 5 corresponding to the nonparametric analysis includes violin plots showing the complete shape of the distribution for each metric in both approaches (N = 2,480). This visualization is used solely for descriptive purposes to illustrate the density, variability, and possible asymmetries in the data. No additional analytical conclusions are drawn from these figures, as the inferential evidence is based solely on the results of the Mann-Whitney U test.

In summary, the results obtained show that, although both approaches—manual and AI-driven—produced comparable metrics across several indicators, subtle differences can be observed in both the distribution and consistency of the values. These variations suggest that automatic generation methods achieve similar levels of performance to manual methods in quantitative terms, but with different behavior patterns, which forms the basis for the interpretive analysis developed in the following section.

IV. DISCUSSION

A. General interpretation of the inferential results

Inferential analyses reveal different behavior depending on the level of aggregation considered. At the aggregate level (N = 12), corresponding to the execution and coverage averages for each of the test cases evaluated, parametric tests (Student's t-test and Welch's t-test) did not identify statistically significant differences between the manual and AI approaches for any of the metrics. This absence of difference suggests that, when results are summarized at a macro level, both approaches produce comparable performance in terms of coverage, effectiveness against mutations, and execution times. Likewise, the

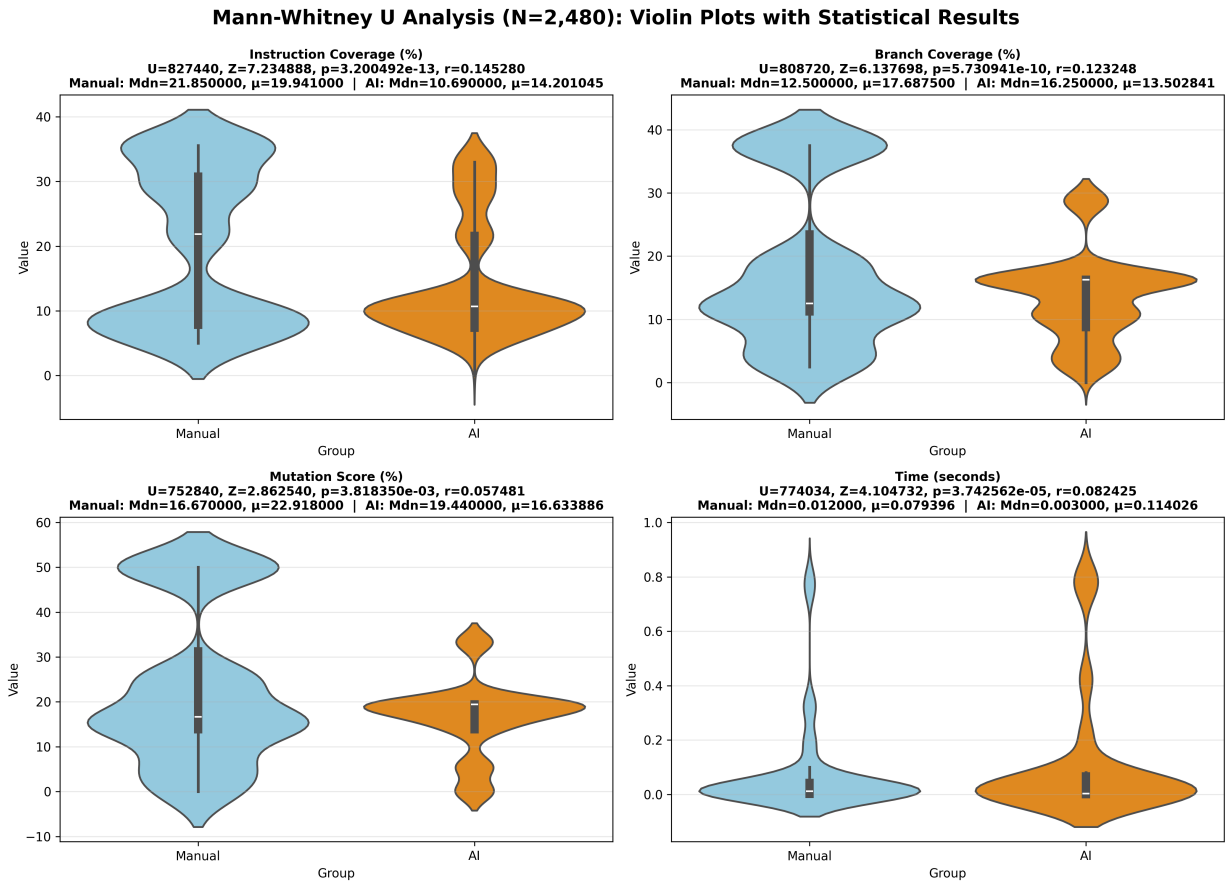

Fig. 5. Violin plots from non-parametric analysis (Mann-Whitney U) showing the distribution shape and density for each metric across both test approaches (Manual vs. AI) at the full individual level (N = 2,480).

effect sizes obtained (Cohen's d) were small, reinforcing the conclusion that, at the aggregate level, the behavior of both approaches is essentially equivalent.

In contrast, analysis of the entire dataset (N = 2,480), based on individual values per iteration, showed a different pattern. Application of the nonparametric Mann-Whitney U test revealed statistically significant differences between the approaches across all metrics. However, despite statistical significance, effect sizes (r) were consistently small or close to zero, indicating that these differences, while detectable in large samples, have limited practical magnitude. In other words, the variations observed between manual and AI-generated tests exist, but they are subtle and do not imply sustained advantages of one approach over the other in terms of coverage, mutation detection capability, or time efficiency.

Taken together, the results suggest that the two approaches perform similarly overall, with differences only noticeable when analyzing complete distributions with a large number of observations. This indicates that the differences found may be associated with the internal variability of each approach rather than a systematic and robust contrast between them. Thus, the joint analysis shows that, although the generative approach introduces its own patterns in the creation of test cases, its final performance does not differ substantially from that obtained through expert manual testing.

B. Differential behavior of testing approaches

Beyond quantitative analysis, evaluation of the test case generation and review process revealed significant qualitative differences in how the two approaches interpret and construct test cases. These observations stem from the functional understanding gained during manual implementation and AI evaluation. In the case of ChatGPT, generation was found to be strongly conditioned by the formulation of the prompt: the use of neutral instructions limited to the method fragment allowed for the evaluation of its autonomous ability to derive functional scenarios, although it tended to produce a reduced number of tests focused on main code paths. This observation is consistent with prior findings showing that LLM-based test generation is often path-insensitive when operating with limited contextual information, which restricts its ability to reason about deep or alternative execution paths [19]. Diffblue Cover exhibited more systematic behavior, guided by its bytecode analysis, generating consistent tests with less user dependency. Both tools stood out for their speed: the AI was able to produce complete tests in a matter of seconds, in contrast to the manual process, which required successive iterations and detailed case-by-case construction.

The manual approach, however, demonstrated greater analytical breadth and depth. Human reasoning made it possible to identify boundary inputs, alternative scenarios, and behaviors

not explicit in the code, which were not always inferred by the AI without additional guidance. This capability came at the cost of considerably greater effort, both in terms of time and cognitive load. Finally, errors caused by unmanaged exceptions in the SUT were observed in both approaches, especially in negative scenarios; such incidents correspond to limitations of the system itself and not to a difference attributable to the testing strategies, so they are only mentioned in general terms.

C. Comparison with previous studies

The results obtained show a pattern consistent with recent research on the use of generative models in software test automation. In particular, the qualitative findings coincide with Garousi et al. [2] and Ricca et al. [11], who report that AI-based tools tend to produce syntactically correct but functionally incomplete tests, especially in scenarios that require contextual reasoning or the identification of alternative paths. Similarly, Schäfer et al. [6] and Yang et al. [7] observed that automatic generation tends to cover only main code paths, which is consistent with the small number of cases generated by ChatGPT in this study and its dependence on prompt content.

In terms of quantitative effectiveness, the results are also consistent with the findings of Bhatia et al. [8] and Li et al. [4], who reported improvements in productivity and generation time, but no substantial differences in metrics such as coverage or mutation. This is consistent with the parametric contrasts in this work, which showed no significant differences at the aggregate level, and with the nonparametric contrasts, whose effect sizes were small despite statistical significance. On the other hand, large-scale studies such as that by Alshahwan et al. [5] in Meta also report that LLMs can incrementally improve existing tests but cannot replace human strategies in identifying edge cases, which is consistent with the greater analytical breadth observed in the manual approach of this study.

D. Practical implications

The study's findings have direct implications for software development organizations and educational institutions. In the business world, the ability of generative AI tools—such as ChatGPT or Diffblue Cover—to produce test cases in a matter of seconds can significantly reduce the time spent on repetitive tasks and speed up the initial stages of quality assurance. However, since the generated tests tend to focus on main code paths and have limited functional coverage, their use should be framed within hybrid strategies where automation acts as a starting point and human judgment ensures comprehensiveness, robustness, and semantic validity.

An important consideration for professional practice is the appropriate selection of tools. The results suggest that, to integrate AI more effectively into the testing cycle, agent-based solutions—such as GitHub Copilot or similar alternatives—may offer advantages over isolated models, as they can analyze the entire project and generate more contextualized test cases [20]. Although these systems do not replace human evaluator reasoning, they do provide assistance that is more aligned with the actual structure of the code, comparable to the bytecode-level analysis performed by Diffblue Cover. For enterprise teams, this means prioritizing tools that can leverage the global context of the repository, always complemented by expert review.

In the field of education, the findings reinforce the need to train professionals capable of combining generative tools with critical thinking and functional understanding. AI can serve as a teaching resource to explore testing alternatives and encourage active learning; however, the ability to interpret system behaviors, anticipate edge cases, and validate the relevance of tests still depends on human reasoning. Overall, the study indicates that AI does not replace traditional testing processes, but rather expands their capabilities when used with technical judgment, human oversight, and sound software engineering practices.

E. Limitations

The experimental design of this study has several limitations that should be considered when interpreting the results. First, the evaluation was performed on a single system under test (Spring PetClinic), which limits the generalization of the findings to other domains, architectures, or design styles. Furthermore, although the total number of observations is large (N = 2,480), the number of classes compared at the aggregate level was small (N = 12), which limits the statistical stability of the parametric contrasts applied at that level.

A second limitation is associated with the partial access that AI-based tools had to the project context. In the case of ChatGPT, the models only received specific code snippets through prompts, without access to the complete repository or relevant artifacts such as configuration, dependencies, or interactions between components. This may have reduced their ability to derive complex scenarios or identify non-obvious execution paths from the provided fragment. Relatedly, the variability inherent in AI-generated testing was not evaluated, as cases were generated only once per method, without exploring the dispersion across multiple model executions.

Additionally, the study did not examine relevant qualitative dimensions such as maintainability, readability, or structural quality of the test code generated by each approach. Nor was the semantic completeness of the assertions or the consistency of the coding style evaluated. Finally, the scope focused on unit and functional tests, without including end-to-end scenarios or integrated user flows, which could reveal additional behaviors not observed in this work. Taken together, these limitations do not invalidate the results obtained, but they do narrow their scope and indicate areas where future research could complement or expand on the findings presented here.

F. Threats to validity

This study presents several threats to validity that must be considered when interpreting the results, following the classification of Wohlin et al. [21] and the recent recommendations of Sjøberg and Bergersen [22]. In terms of internal validity, the results may have been influenced by uncontrolled

factors, such as fluctuations in execution times, variability inherent in AI-generated tests—given that models can produce different outputs for small variations in the prompt—and errors originating from unhandled exceptions in the system under test itself.

Regarding external validity, the conclusions are limited by the use of a single system (Spring PetClinic) and a specific technological ecosystem (Java and Spring Boot), as well as by the comparison being restricted to two representative tools (ChatGPT and Diffblue Cover). These methodological decisions reduce the ability to generalize the results to larger projects, industrial domains, or alternative generative models. Regarding construct validity, although the metrics used—coverage, mutation score, and execution time—capture relevant technical dimensions, they do not cover aspects such as maintainability, readability, or structural quality of the test code, which limits the conceptual breadth of the constructs evaluated.

Finally, the validity of the conclusion is conditioned by the dual nature of the experimental design: while the aggregate level has a small sample size (N = 12), which reduces statistical power, the complete level (N = 2,480) can generate significance with small effect sizes, increasing the risk of false positives. Furthermore, multiple independent generations of tests were not performed by the AI, which prevents the evaluation of statistical stability between runs. Although these threats do not invalidate the findings, they do limit the scope of possible inferences from the study.

G. Future work

The results obtained open up various opportunities for future research aimed at gaining a more complete understanding of the role of AI in automated test generation. First, it would be valuable to replicate this study on larger systems under test or those belonging to more complex industrial domains, in order to assess whether the patterns observed hold true under different architectures, design styles, and workloads. Likewise, a natural extension would be to compare the performance of agent-based tools—such as GitHub Copilot Agent, Claude Code Agent, or equivalent platforms—whose iterative mechanism and self-correction capability could enable more comprehensive and contextualized test generation than that obtained using isolated models.

Another relevant line of work involves incorporating types of tests not addressed in this study, particularly end-to-end tests and integrated user flows, where AI agents could exhibit different behavior due to their ability to reason about sequential actions. Furthermore, future research could evaluate the maintainability, readability, and structural quality of the test code generated by these agents, as well as the stability of the results across multiple runs, given that these systems often adjust their output based on intermediate feedback or incremental analysis of the repository.

Finally, considering recent advances in agentic architectures [20], it would be pertinent to explore configurations where LLMs have expanded access to the project context, integrating static analysis, dynamic exploration, or automated verification mechanisms. Such approaches would allow us to investigate whether the combination of iterative reasoning, short-term memory, and access to the entire repository improves functional coverage, the identification of complex execution paths, or the generation of boundary scenarios more effectively than the tools evaluated in this study.

V. Conclusions

This study shows that generative AI is capable of approximating the quantitative performance of human-designed tests, but does not fully reproduce the depth of their functional reasoning. Rather than confirming the superiority of one approach over the other, the results reveal a structural tension: while generative models optimize speed and syntactic consistency, human engineers contribute contextual judgment, business understanding, and the ability to anticipate scenarios that are not trivial from code inspection alone. In this sense, the work contributes to nuancing the narrative of total replacement, proposing a more balanced reading of the role of AI in software testing.

In conceptual terms, the research provides empirical evidence that traditional quality metrics—such as coverage and mutation score—are necessary but insufficient for evaluating the true scope of AI-generated tests. Although the values obtained are comparable to those obtained by humans, functional analysis reveals a cognitive gap: generative models tend to operate as pattern optimizers on code, while human testers reason about requirements, usage flows, and domain constraints. This distinction reinforces the importance of incorporating semantic dimensions and understanding of system behavior into future evaluations of AI-assisted testing, aligning with and extending recent discussions in the literature on LLMs and test quality.

From a professional practice perspective, the findings suggest that generative AI should be integrated as a technical support component within the software verification process, rather than as a substitute for human reasoning. The tools evaluated proved particularly useful for accelerating the initial generation of test cases, reducing effort in repetitive tasks, and providing a structured basis on which human developers or evaluators can refine, expand, and validate critical scenarios. Hybrid strategies are recommended, in which AI is responsible for rapid synthesis and systematic exploration of variants, while human oversight ensures the functional relevance of cases, manages exceptions, and prioritizes the most sensitive behaviors of the system.

Finally, the study highlights a clear agenda for the research community: to move toward evaluation frameworks that capture not only surface metrics, but also the conceptual quality of AI-generated tests and their alignment with the functional objectives of the system. The gap observed between the generative speed of models and the interpretive richness of human testers indicates that the future of AI-assisted testing does not lie in replacement, but in the construction of collaborative human-AI models, in particular through agents capable of iteratively reasoning about the entire repository without losing sight of the design decisions and acceptance criteria defined by humans.

REFERENCES

[1] A. Trudova, M. Dolezel, and A. Buchalcevova, "Artificial intelligence in software test automation: A systematic literature review," pp. 181–192, 2020.

[2] V. Garousi, Z. Jafarov, A. B. Keleş, S. Değirmenci, E. Özdemir Testinium AŞ, and R. Zarringhalami, "Ai-powered software testing tools: A systematic review and empirical assessment of their features and limitations," Tech. Rep., 5 2025. [Online]. Available: https://arxiv.org/abs/2409.00411

[3] A. Singh and O. Al-Azzam, "Artificial intelligence applied to software testing." Academy and Industry Research Collaboration Center (AIRCC), 11 2023, pp. 01–12.

[4] T. Li, C. Cui, R. Huang, D. Towey, and L. Ma, "Large language models for automated web-form-test generation: An empirical study," *ACM Transactions on Software Engineering and Methodology*, 5 2025. [Online]. Available: https://doi.org/10.1145/3735553

[5] N. Alshahwan, J. Chheda, A. Finegenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Automated unit test improvement using large language models at meta," in *FSE Companion - Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. Association for Computing Machinery, Inc, 2024, pp. 185–196. [Online]. Available: https://arxiv.org/abs/2402.09171

[6] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," 12 2023. [Online]. Available: https://arxiv.org/abs/2302.06527

[7] L. Yang, C. Yang, S. Gao, W. Wang, B. Wang, Q. Zhu, X. Chu, J. Zhou, G. Liang, Q. Wang, and J. Chen, "An empirical study of unit test generation with large language models." 9 2024. [Online]. Available: https://arxiv.org/abs/2406.18181

[8] S. Bhatia, T. Gandhi, D. Kumar, and P. Jalote, "Unit test generation using generative ai : A comparative performance analysis of autogeneration tools," 2 2024. [Online]. Available: https://arxiv.org/abs/2312.10622

[9] R. Kanth, R. Guru, M. B. K, and D. Akshaya, "Ai vs. conventional testing: A comprehensive comparison of effectiveness &efficiency," *Educational Administration: Theory and Practice*, 1 2023. [Online]. Available: https://kuey.net/index.php/kuey/article/view/7495

[10] H. Kirinuki and H. Tanno, "Chatgpt and human synergy in black-box testing: A comparative analysis," 1 2024. [Online]. Available: https://arxiv.org/abs/2401.13924

[11] F. Ricca, A. Marchetto, and A. Stocco, "A multi-year grey literature review on ai-assisted test automation," 1 2025. [Online]. Available: https://arxiv.org/abs/2408.06224

[12] G. Wang, Q. Xu, L. C. Briand, and K. Liu, "Mutation-guided unit test generation with a large language model," 8 2025. [Online]. Available: https://arxiv.org/abs/2506.02954

[13] R. F. de Lima Junior, L. F. P. de Barros Presta, L. S. Borborema, V. N. da Silva, M. L. de Melo Dahia, and A. C. S. e Santos, "A case study on test case construction with large language models: Unveiling practical insights and challenges," 12 2023. [Online]. Available: https://arxiv.org/abs/2312.12598

[14] S. Rehan, B. Al-Bander, and A. A.-S. Ahmad, "Harnessing large language models for automated software testing: A leap towards scalable test case generation," *Electronics (Switzerland)*, vol. 14, 4 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/7/1463

[15] D. Huang, J. M. Zhang, M. Harman, Q. Zhang, M. Du, and S.-K. Ng, "Benchmarking llms for unit test generation from real-world functions," 8 2025. [Online]. Available: https://arxiv.org/abs/2508.00408

[16] M. Gkikopouli and B. Bataa, "Empirical comparison between conventional and ai-based automated unit test generation tools in java," p. 30, 6 2023. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1764443&dswid=4942

[17] S. Baltes, F. Angermeir, C. Arora, M. M. Barón, C. Chen, L. Böhme, F. Calefato, N. Ernst, D. Falessi, B. Fitzgerald, D. Fucci, M. Kalinowski, S. Lambiase, D. Russo, M. Lungu, L. Prechelt, P. Ralph, R. van Tonder, C. Treude, and S. Wagner, "Guidelines for empirical studies in software engineering involving large language models," 9 2025. [Online]. Available: https://arxiv.org/abs/2508.15503

[18] B. Trinkenreich, F. Calefato, G. Hanssen, K. Blincoe, M. Kalinowski, M. Pezzè, P. Tell, and M. A. Storey, "Get on the train or be left on the station: Using llms for software engineering research," in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, 7 2025, pp. 1503–1507. [Online]. Available: https://dl.acm.org/doi/10.1145/3696630.3731666

[19] D. Liao, X. Yin, S. Pan, C. Ni, Z. Xing, and X. Sun, "Navigating the labyrinth: Path-sensitive unit test generation with large language models," 10 2025. [Online]. Available: http://arxiv.org/abs/2509.23812

[20] A. Plaat, M. van Duijn, N. van Stein, M. Preuss, P. van der Putten, and K. J. Batenburg, "Agentic large language models, a survey," 11 2025. [Online]. Available: https://arxiv.org/abs/2503.23037

[21] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. [Online]. Available: https://dl.acm.org/doi/book/10.5555/2349018

[22] D. I. Sjøberg and G. R. Bergersen, "Improving the reporting of threats to construct validity," in *ACM International Conference Proceeding Series*. Association for Computing Machinery, 6 2023, pp. 205–209.