Milestone3

ANDRE GNANDT* and BINGZHENG JIN

ACM Reference Format:
Andre Gnandt and Bingzheng Jin. 2025. Milestone3. 1, 1 (December 2025), 7 pages.

1 GenAI Disclosure

The use of Gen AI tools (ChatGPT) was used in the code production of RQ3.

2 Introduction and Research Questions

2.1 RQ1 - AI code test contributions - Andre Gnandt and Bingzheng Jin

We want to investigate the frequency in which AI agents contribute tests to their respective projects and repo's. Considering that AI agents pose a risk of contributing code that contains errors, it is important for some form of a safeguard or error check to take place on their code. We would like to explore the rate at which AI agents contribute tests in relation to changes made to the codebase. You can call this the test-to-code churn ratio. If we have extra time, then more details will be explored, such as the metrics for the different types of tests (unit, integration etc.) contributed.

2.2 RQ2 - AI code issues - Andre Gnandt

We want to investigate how often code errors or code quality issues appear in the AI agents pull requests. More specifically, we want to explore what kind of errors or issues are most common amongs the AI agents. If we have extra time, we will explore the possible reasons as to why such code issues occur frequently in AI agents.

2.3 RQ3 - AI code contribution efficiency - Bingzheng Jin

We want to explore the rate at which AI agents produce code contributions. It would be interesting to examine just how efficient at producing correct and reliable code these agents are, and the overall time it takes for an agent to complete its task and for its PR to be approved.

3 Complete Data Wrangling Methodology

3.1 RQ1 - Andre Gnandt and Bingzheng Jin

Data Frame indexing, sub-setting and filtering was frequently used in the exploration of this RQ. It is shown in the code of 'RQ1.jpynb' in several cases: obtaining all PR's with test contributions from sub-setting the all PR's table with an indexed and filtered all prs df based on text content, getting all PR commits with test contributions from sub-setting the pr commits table with an indexed and filtered series from the 'message' column of the pr commits table based on matching texts (from the indexed 'message' column). These methods were applied in order to obtain all of the records from the pull requests table which contained a title or body field related to code test contributions. Similarly, this was performed on the pr commits table and 'message' column to obtain commit records related to test code contributions. Only pr commits with a PR Id not already identified from the pull requests table were selected, in order to obtain unique records for all PR's related to code test contributions.

*Both authors contributed equally to this research.

Authors' Contact Information: Andre Gnandt; Bingzheng Jin.

2025. ACM XXXX-XXXX/2025/12-ART
https://doi.org/

Data Frame aggregation was performed to combine the Pull requests and PR commits (with disjoint PR IDs) in order to obtain a data frame consisting of all unique pull requests with code test contributions, and their text related content in the new and transformed 'text' column. This can be seen in the code as both the code test PR's and code test PR commits data frames are initially restructured to match each others column names and types, and later concatenated (using .concat()) in order to obtain all distinct PR's related to code test contributions and their text content. This also involves using a group by PR ID and aggregation of text content on the pr commits.

Text processing and Regex were used to match records to code-test-like regex patterns in the 'message', 'title', and 'body' columns of the tables and data frames described above. A regex pattern for code-test phrases was constructed and applied to find matches in any of these fields in the PR's and PR commits tables, and only records with matches were collected in order to form our full collection of code test PR's. A dictionary of regex patterns for identifying different types of tests (like unit test or integration tests) was created. Each test type regex in this test type dictionary was used to locate test type matches to each pull request in our total test-contributive pull requests data frame (the data frame described above) in order to identify and count how many PR's contributed such tests for each different type of test.

3.2 RQ2 - Andre Gnandt

Data Frame indexing, sub-setting and filtering was frequently used in the exploration of this RQ. In the code, you will see this used in several cases. Including: Getting count of PR's that have PR reviews, getting count of PR's that have PR reviews in a commented state, getting count of PR's that have PR reviews in a changes requested state, getting count of PR's that are imperfect (not initially approved or dismissed) and much more. All of these involved the indexing, filtering, sub-setting, and querying (.query() method) operations between the pr_reviews and pr_review comment tables to identify PR's with code issues of different severity levels (significant issues, minor issues, possible issues, and "all imperfect PR's" - which includes have any PR review in a state of commented or changes requested).

Text Processing using dictionaries was utilized along with aggregation and transformation (using .apply() method) to match all PR's with possible code issues (as described earlier) to certain categories of code issues based on matching keywords in the texts of their body fields. This was also used to identify PR reviews in a "commented state" that have PR review comments or bodies with texts that match code issues identifying keywords, to better identify PR reviews in commented state which actually have code issues.

These techniques were used to obtain the values for: count of all PR's that are imperfect (either had a review with commented or changes requested state), count of all PR's with possible code issues (had a PR review with a commented state), count of all PR's with minor code issues (had a PR review with a commented state and text matched comments or body that identify code issues), count of all PR's with significant code issues (had a PR review with a "changes requested" state, and the number of PR's that have code issues belonging to a certain "code-issue" category".

These 4 counts of issue-identifying PR's were compared to the total number of PR's that have reviews to obtain the metrics on the frequency of code issues in AI Agent PR's and severity of such, and plotted in bar charts for analysis. The number of PR's that have code issues belonging to a certain category was plotted on a pie chart (for each category) for analysis.

3.3 RQ3 - Bingzheng Jin

The analysis uses the AIDev-pop dataset, publicly hosted on Hugging Face. Four data tables are relevant:

(1) pull_request.parquet — main PR metadata including creation time, closure time, merge state, and agent attribution.
(2) issue.parquet — repository issue metadata, including initial creation time.
(3) related_issue.parquet — mapping between PRs and their corresponding issues.
(4) pr_commit_details.parquet — commit-level code changes including additions and deletions.

We restrict our analysis to:

- PRs that are closed
- PRs with non-null agent fields

This ensures that only valid agent-generated software contributions are studied.

Temporal Metrics

For each PR, we compute:

- Issue → PR creation time
- Issue → PR closure time
- Issue → PR merge time (if available)

Time durations are converted into days using timestamp differences. These intervals help quantify how long it takes for issues to be addressed and PRs to be reviewed and completed.

Code-Change Metrics

We measure modification complexity by aggregating commit-level changes:

- Total additions
- Total deletions
- Total changes = additions + deletions

Aggregating changes across all commits linked to each PR helps determine whether agent-generated edits tend to be small or large in scale.

We calculate descriptive statistics for both temporal and code metrics, and visualize distributions using histograms and boxplots. The objective is to observe skewness, spread, and presence of extreme values.

For all computed metrics, we generated:

- Summary statistics
- Histograms for time-based distributions
- Boxplots for code change distributions
- Comparisons between merged and non-merged PRs (optional)

These visualizations facilitate interpretation of trends across thousands of PRs.

4 Analysis and Results

4.1 RQ1 - Andre Gnandt and Bingzheng Jin

The size of the total test contribute pull requests df (code_test_PRs df) is compared to the size of all pull requests (all_pr_df) to obtain the frequency in which AI agents contribute tests. We find that 23.55% of the PR's included test contributions, and 76.45% did not. These findings were also plotted in a pie chart for visualization purposes.

For each specific test type, the total count of PR's with the specific test type contribution (from the regex_to_categories and test_category_counts dictionaries) was plotted in a pie chart. We were then able to analyze the percentage of all total test contributions that fit a specific test type. We

find that 69.52% of tests are Unit Tests, 19.75% are Integration Tests, 2.52% are UI Tests, 7.36% are End-to-End tests, and 0.83% are system tests.

4.2 RQ2 - Andre Gnandt

For each of the 4 code issue category types (imperfect PR's, Possible Minor Issues, Minor Issues, and Significant Issues), the size of the respective data frames was compared to the size of the data frame of all Pull Requests that have been reviewed. From this we find the different percentages of PR's that have different levels of code issues severity. We find that 12.14% have significant code issues, 33.24% have minor issues, 29.66% have possible minor issues, and 75.04% are imperfect PR's (PR's that required some form of reconsideration before merging). This was plotted in a bar chart for visualization. The first 3 severities are disjoint while the last one (all imperfect PR's) is the total of the other 3. From this we know that 12.14%+33.24% = 45.38% of PR's have some kind of code issue (mostly minor issues).

For each code issues type, the count of all PR's of the test type was plotted in a bar chart (values from the issue_types and issue_counts dictionaries). From this, we can analyze which percentages of code issues are a certain type of issue. We find that 32.8% of code issues are code style and conventions, 27.7% are code quality, structure or efficiency, and 39.4% are legitimate errors or bugs.

4.3 RQ3 - Bingzheng Jin

Issue to PR Close Time

The distribution of close times is highly right-skewed, with most PRs closing within a short duration after issue creation. As shown in Figure 1:

- A large concentration of PRs close very quickly.
- A long tail represents tasks taking hundreds to thousands of days.
- The steep peak near zero indicates rapid agent responsiveness.

Interpretation:
AI agents are generally capable of completing tasks quickly. Extreme outliers suggest that some issues remain open due to human review delays or long-term repository workflows, not necessarily due to agent inefficiency.

Total Code Changes per PR The boxplot in Figure 2 reveals substantial variation in code modification size:

- Most PRs involve small or moderate changes.
- Numerous extreme outliers reach hundreds of thousands or even over one million lines changed.
- These may represent automated refactoring, bulk changes, or repository-wide modifications.

Interpretation:
AI agents can handle both small, targeted tasks and large-scale transformations. The variability highlights the diverse nature of tasks assigned to AI systems.

Issue → PR Merge Time
The merge-time distribution (Figure 3) mirrors the close-time distribution:

- Most merged PRs are accepted shortly after issue creation.
- A dense peak near zero suggests that many agent-generated solutions are quickly accepted.
- The long tail again shows rare but extreme cases with delays of hundreds or thousands of days.

Interpretation:
Human reviewers tend to accept agent-generated contributions quickly, indicating trust or straightforwardness of tasks. Outliers likely correspond to tasks awaiting human attention rather than agent delays.

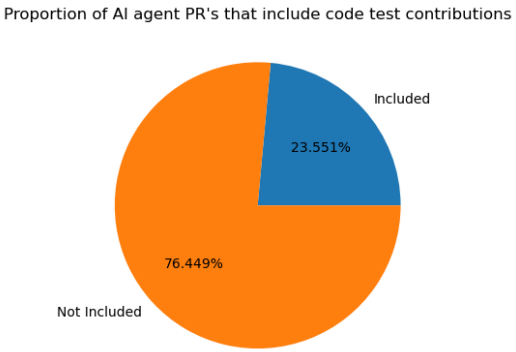
From the combined results:

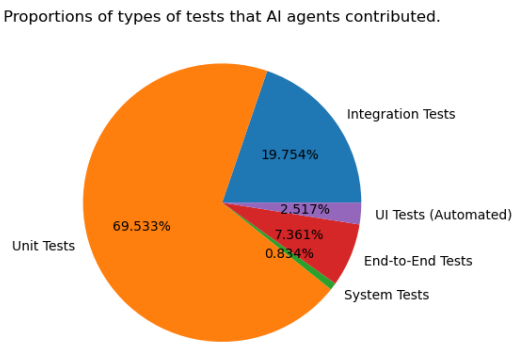
- Agents respond quickly to most tasks.
- Contribution size varies widely, indicating flexible capability across task types.
- Merged PRs follow similar timing patterns, implying that agent-generated solutions are often accepted promptly.
- Long-tail behavior reflects human review factors rather than agent-side performance.

Overall, AI agents demonstrate strong efficiency in both task turnaround time and code generation.

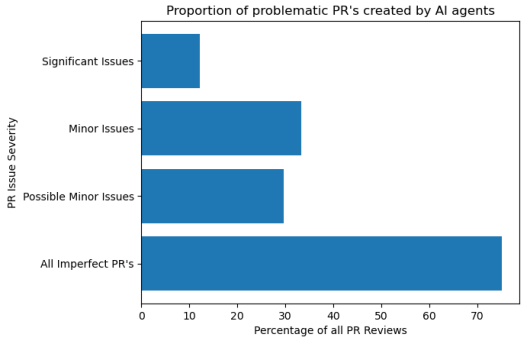
5 Visualizations and Findings

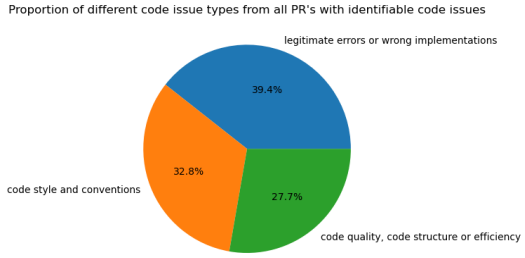
5.1 RQ1 - Andre Gnandt and Bingzheng Jin



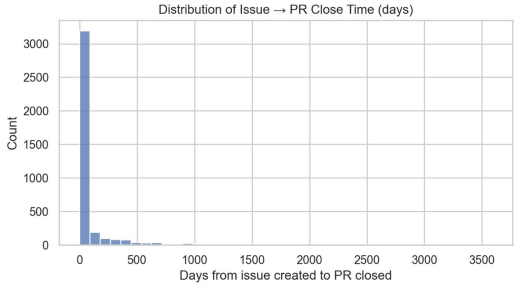


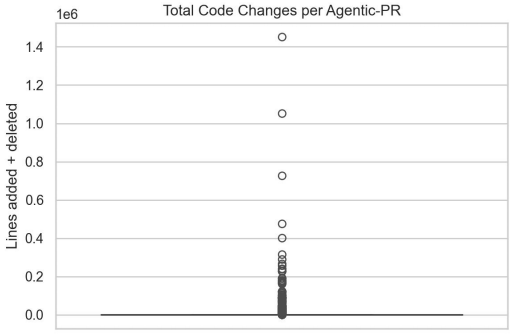
5.2 RQ2 - Andre Gnandt





5.3 RQ3 - Bingzheng Jin





6 Conclusions

6.1 RQ1 - Andre Gnandt and Bingzheng Jin

From our results, it seems that AI agents contribute test's 23.55% of the time. This demonstrates that the agents do contribute tests somewhat frequently, but may still be lacking in this category. Proper test coverage in software development usually aims for test coverage of 75%-100%, suggesting that the agents may not contribute tests frequently enough to fulfill this, and may be a cause for concern.

We can see in the pie chart that the majority of AI agent test contributions are Unit Tests (69.55%). The only other test type that is contributed a considerable amount is Integration Tests (19.75%). All other test types are contributed minimally. This is not too surprising, as these 2 types of test are most common and of greatest importance at the foundation of testing.

6.2 RQ2 - Andre Gnandt

We have found that 45.38% of AI agents PR's have some type of code issue. This suggests that AI agents contribute issues a frequent and concerning amount of the time, however, looking more closely, most of these issues are minor or insignificant issues (33.24%). This suggests that the agents don't frequently cause concern-able code issues. However, 12.14% of PR's involved significant issues, suggesting that there is at least a small concern of agents contributing code issues a minority of the time.

We see that the agents code issue categories are relatively evenly proportioned. This means that the different types of code issues in which the agents contributed were pretty evenly distributed across the categories: legit errors and bugs, code style and conventional issues, and code quality, structure or efficiency. The category with the highest frequency was 'legitimate errors and bugs', which accounts for 39.4% of all code issues. The remaining 2 categories: code style and conventions occurs 32.8% of the time, and code quality, structure or efficiency issues only 27.7% of the time. This is not too surprising, because even in human PR's, errors and bugs are by far the most common issue, but AI agents may have a harder time following code quality, efficiency, style or conventions than humans do. This results in an elevation of the frequency of these 2 less frequent categories in comparison to humans, but with bugs and legitimate errors still being the most frequent code issue for both humans and AI agents.

6.3 RQ3 - Bingzheng Jin

Our analysis shows that AI agents are capable contributors to real-world software development workflows. They typically generate solutions rapidly, often within a short period after issue creation, and their contributions vary naturally depending on task scale. The high frequency of quick merges suggests that agent-generated code is generally acceptable and integrates smoothly into existing codebases. However, long-tail patterns indicate that human review or project maintenance cycles still influence final PR outcomes. While AI agents excel at execution, full integration still depends on human oversight and repository workflows. These findings provide valuable insight into the practical performance of AI coding agents and highlight promising potential for their use in large-scale software systems.

7 Link to your GitHub repository

GitHub repository link: https://github.com/andre-gnandt/DATA542-Project

Please see the markdown file for instructions and folders RQ1, RQ2 and RQ3 for the appropriate code for each research question.