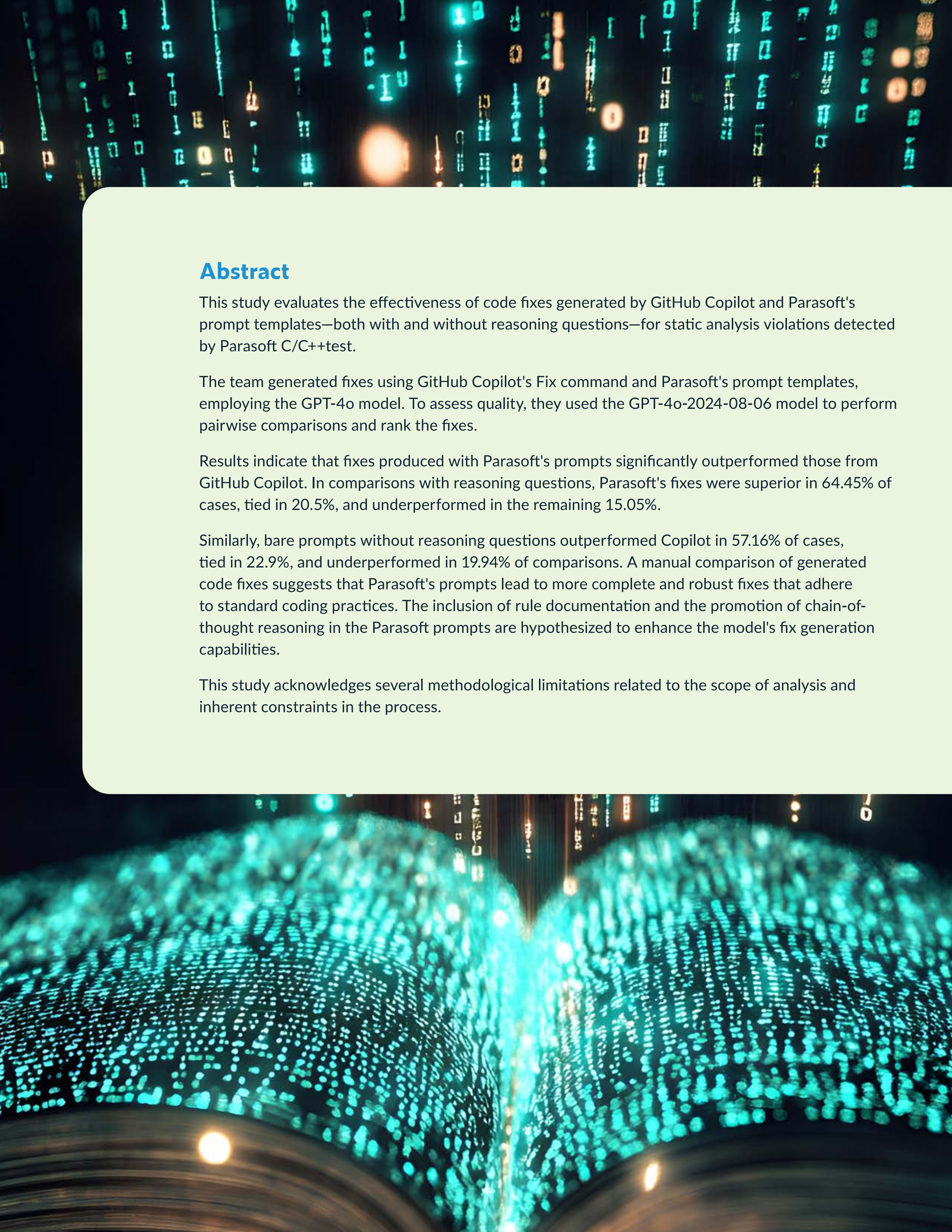
TECHNICAL WHITEPAPER

A Comparison of Static Analysis Violation Fixes

Parasoft vs GitHub Copilot



Abstract

This study evaluates the effectiveness of code fixes generated by GitHub Copilot and Parasoft's prompt templates—both with and without reasoning questions—for static analysis violations detected by Parasoft C/C++test.

The team generated fixes using GitHub Copilot's Fix command and Parasoft's prompt templates, employing the GPT-4o model. To assess quality, they used the GPT-4o-2024-08-06 model to perform pairwise comparisons and rank the fixes.

Results indicate that fixes produced with Parasoft's prompts significantly outperformed those from GitHub Copilot. In comparisons with reasoning questions, Parasoft's fixes were superior in 64.45% of cases, tied in 20.5%, and underperformed in the remaining 15.05%.

Similarly, bare prompts without reasoning questions outperformed Copilot in 57.16% of cases, tied in 22.9%, and underperformed in 19.94% of comparisons. A manual comparison of generated code fixes suggests that Parasoft's prompts lead to more complete and robust fixes that adhere to standard coding practices. The inclusion of rule documentation and the promotion of chain-of-thought reasoning in the Parasoft prompts are hypothesized to enhance the model's fix generation capabilities.

This study acknowledges several methodological limitations related to the scope of analysis and inherent constraints in the process.

Methodology

This study employed a systematic approach to evaluate and compare the effectiveness of different automated code violation fix generation methods.

Input Data

For input data, we constructed a CWE Violations dataset by analyzing 1,856 open-source projects with Parasoft C/C++test's static analysis engine. We used the CWE Top 25 + On the Cusp 2023 test configuration and identified severity 1 through 4 violations, finding a total of 330,906 violations located within function bodies.

Each violation includes the following:

- Associated C/C++test rule ID, for example, CODSTA-199
- Precise location within the associated source file
- Violation message, such as, "File not closed: uhook_fdin, new_fd"
- Rule description
- Code flow traces for flow analysis violations

Sample Selection

To sample a somewhat representative set of violations, we applied the following procedure.

1. Calculated a rule frequency distribution in the entire dataset.
2. Calculated a rule frequency distribution in each project.
3. Selected a total of five projects with a rule distribution most similar to the rule distribution within the entire dataset.
4. Selected a total of 20 of the most common rules in the entire dataset. Those 20 rules are responsible for over 97% of all discovered violations.
5. Applied samplings to the five selected projects, selecting 30 samples for each of the 20 most common rules in the entire dataset.
6. Used a total of 600 samples for further experiments.

Experiment Design

We obtained fixes by invoking the GitHub Copilot Fix command within VS Code and by applying Parasoft's prompt templates, with and without Reasoning Questions (referred to further as bare and reasoning, respectively).

The comparison used the GitHub Copilot plugin dated Nov 8, 2024, with accompanying GitHub Copilot Chat v0.22.

Data Collection

Due to technical limitations in the programmatic interface with Visual Studio Code, we successfully extracted fixes for 432 out of the 600 violations from the dataset using Github Copilot. The success rate across rules averaged approximately 72% with a singular notable outlier: Rule PB-66_a successfully obtained a fix for only a single sample, significantly deviating from this pattern.

Both VS Code and Parasoft created fixes using the GPT-4o-2024-08-06 model.

Evaluation Process

We employed the GPT-4o-2024-08-06 model to rank the quality of the outputs. The model was asked with Ranking_Prompts to analyze two solutions to the same task of fixing a static analysis violation and return a response using the Ranking_Prompt_Response_Schema that proclaimed which solution was a winner or whether there was a tie. To control for potential order bias, we compared each input sample pair twice, alternating the presentation order of the analyzed solutions in the evaluation prompts.

Results

GitHub Copilot vs C++test With Reasoning

	Win Rate	Tie Rate	Lose Rate
GitHub Copilot	0.150895	0.204604	0.644501
C++test with reasoning questions	0.644501	0.204604	0.150895

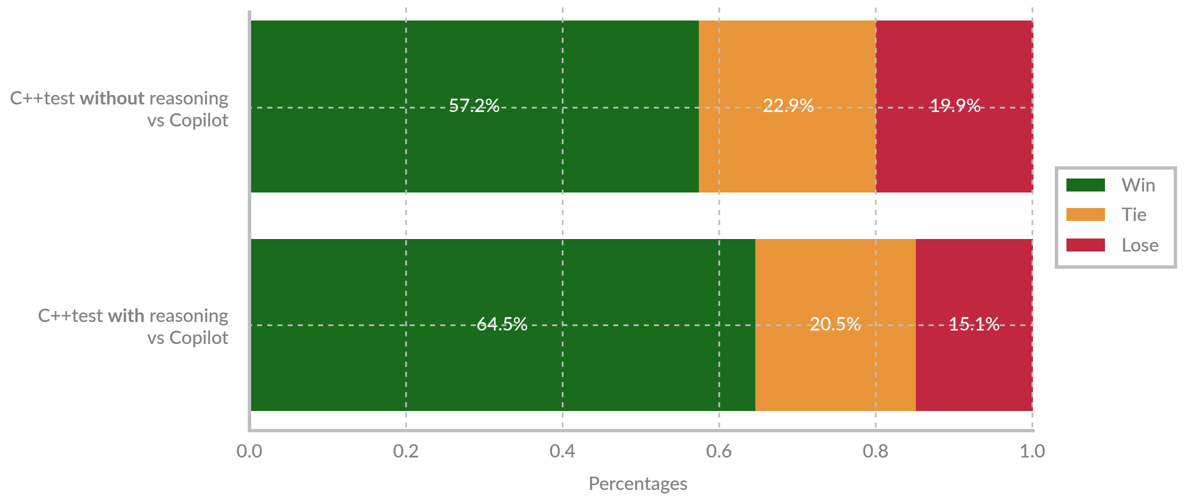
GitHub Copilot vs C++test Without Reasoning

	Win Rate	Tie Rate	Lose Rate
GitHub Copilot	0.199488	0.2289	0.571611
C++test without reasoning questions	0.571611	0.2289	0.199488

Pairwise Win Rates

	GitHub Copilot	C++test With Reasoning Questions	C++test Without Reasoning Questions
GitHub Copilot		0.150895	0.199488
C++test with reasoning questions	0.644501		0.313433
C++test without reasoning questions	0.571611	0.186567	

Win, Tie, Lose Percentages: C++test vs Copilot



Conclusion

This analysis shows that fixes obtained with Parasoft's prompts consistently rank better than those obtained from GitHub Copilot. The better performance is observed both for bare and reasoning prompt variants, with reasoning prompts performing slightly better.

Manual inspection of Sample Data revealed that fixes generated with Parasoft's prompts are often more complete (such as fixing all instances of an issue in adjacent lines), robust (code has better error handling), and conform to standard practices. We hypothesize that the inclusion of rule documentation and enforcement of chain-of-thought reasoning in the prompts (either in the form of reasoning questions or the prompt structure of bare prompts) stimulate the model to produce better fixes.

Research Limitations

1. The analysis was limited to violations produced by Parasoft C/C++test and only violations appearing within function bodies were analyzed. Violations that appeared outside of function bodies were not included in this analysis.
2. Since the ranking prompts only included the function body and not other relevant code, it has access to the same limited information as Parasoft's fixing prompt, whereas Copilot uses full (or almost full) file context. In rare cases, this might create a bias against Copilot fixes. For example, when the solution returned by Parasoft's prompt templates indicates that a violation is a false positive or when the model lacks information to fix the violation, the ranking model, having access to the same limited data, might be inclined to agree with Parasoft's assessment.

TAKE THE NEXT STEP

Contact us to learn more about Parasoft's static analysis solutions.

About Parasoft

Parasoft helps organizations continuously deliver high-quality software with its AI-powered software testing platform and automated test solutions. Supporting the embedded, enterprise, and IoT markets, Parasoft's proven technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award-winning reporting and analytics dashboard provides a centralized view of quality, enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives—security, safety-critical, Agile, DevOps, and continuous testing.

Appendices: Prompts Templates

Ranking Prompts

System Prompt Template

```
You will be presented with a programming task enclosed within
<TASK></TASK> tags, and two solutions: Solution_A and Solution_B,
within their respective <SOLUTION_A></SOLUTION_A> and <SOLUTION_
B></SOLUTION_B> tags.
Your job is to analyze both solutions with respect to the task
description, then compare them and deliver a final verdict.
If both solutions are quite similar without any significant
differences, choose 'tie' as the answer.
```

User Prompt Template

```
<TASK>
{task}
</TASK>

<SOLUTION_A>
{solution_A}
</SOLUTION_A>

<SOLUTION_B>
{solution_B}
</SOLUTION_B>
```

Ranking Prompt Response Schema

```
class ABRankedSolutions(BaseModel):
    solution_A_analysis: str
    solution_B_analysis: str
    comparison: str
    final_verdict: str
    winning_solution: Literal["A", "B", "tie"]
```