Software Product Assurance Conference 2025

@ESA ESTEC, The Netherlands

September 25th, 2025

Automated AI-Driven Code Repair

Tool Experience Report

Presented by Pedro Nunes and Nuno Silva @ Critical Software S.A.

Developed by Rafael Fonseca, Computer Engineering Master's degree Trainee @ISEC

Disclaimer & Research Intent

This tool is *not assumed to be compliant* by default!

- Exploratory prototype
- Not used in production
- Designed to generate lessons learned
- Identify future research topics for certifiability and compliance of AI-aided SW development techniques
- Tool Experience Report from a Master's degree work (Rafael Fonseca from ISEC - Polytechnic Institute of Coimbra's Higher Engineering Institute)

Will ECSS NextGen address these emerging paradigms?

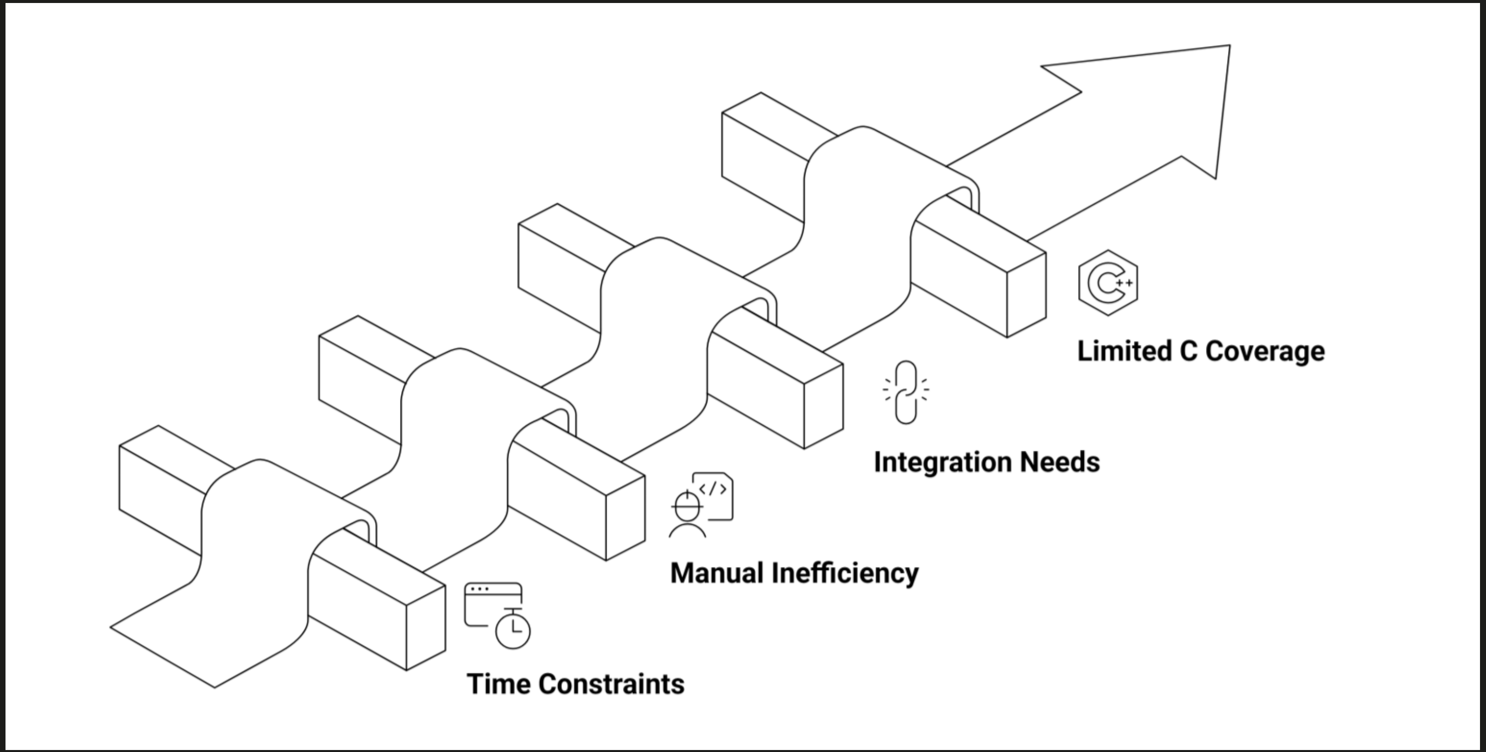
Agenda

- Why Automated Code Repair?
- Benchmark Study
- Integrated Toolchain
- Workflow
- Prompt Context Ensemble
- Key Activities
- Dashboard & Reporting
- Results
- Key Benefits & Conclusion
- Demonstration
- Lessons Learned
- Top 10 Challenges
- Q&A

Why Automated Code Repair?

Increasing pressure to maintain both speed and quality due to:

- **Time Constraints:**
  Rapid development cycles demand fast and reliable code fixes without slowing delivery
- **Manual Inefficiency:**
  Traditional rework is slow, inconsistent and error-prone, adding cognitive load to developers
- **Integration Needs:**
  Solutions must fit seamlessly into existing CI/CD pipelines with minimal disruption
- **C language support in similar tools:**
  SonarQube now offers a tool, AI Code Fix, but it does not cover C language



Benchmark study

Several Models have been tested

- Initial state: 33 issues, 242 Code Smells, 48 security hotspots and 96% UT coverage.
- GPT-4o-MINI: 8 issues left, 153 Code Smells, 31 security hotspots and 89% UT coverage.
- Some models seem to outperform GPT-4o-MINI but they simply removed the code → no pass.
- Example project with > 200 KLOC of C code.

Models	Reliability	Maintainability	Security	Security Review	Coverage	Duplications	Quality Gate	Time Needed (sec)	Time Needed (min)
Pre-Fix	33 BUGS	242 CS	0	49SH	96%	5.3%	PASSED	*	*
GPT4	8 BUGS	153 CS	0	31 SH	89%	5.3%	PASSED	1099	18,31666667
GPT3.5	19 BUGS	275 CS	0	65 SH	91,60%	5.7%	FAILED	317	5,283333333
GPT-4O-MINI	8 BUGS	153 CS	0	31 SH	89%	5.3%	PASSED	681	11,35
O3-MINI HIGH	15 BUGS	267 CS	0	65 SH	91.5%	5.7%	FAILED	1829	30,48333333
malibu-poolside	4 BUGS	87 CS	0	11 SH	91.6%	5.4%	FAILED	1912	31,86666667
malibu-Karvel-v1	4 BUGS	87 CS	0	11 SH	91.6%	5.4%	FAILED	1880	31,33333333
deepseek-r1:7b	16 BUGS	271 CS	0	65 SH	91,60%	5.7%	FAILED	17578	292,9666667
starcoder2:7b	16 BUGS	268 CS	0	65 SH	91,60%	5.7%	FAILED	2530	42,16666667
qwen2.5-coder:7b	16 BUGS	267 CS	0	65 SH	91,60%	5.7%	FAILED	3007	50,11666667
qwen2.5:7b	17 BUGS	273 CS	0	65 SH	91,60%	5.7%	FAILED	5787	96,45
codestral:22b	15 BUGS	271 CS	0	65 SH	91.6%	5.7%	FAILED	18092	301,5333333
codegemma:7b	1 BUGS	78 CS	0	1 SH	92%	5.7%	FAILED	6089	101,4833333
codellama:7b	18 BUGS	271 CS	0	65 SH	91.6%	5.7%	FAILED	7505	125,0833333

Integrated Toolchain

Language

Python: integration of Jenkins/SonarQube, Bitbucket, Azure OpenAI, Tree Sitter, and HTTP Requests

Static Analysis Tool

SonarQube: SAT that identifies code quality issues and vulnerabilities according to quality profiles (e.g. MISRA C)

AI Model Provider

Azure cloud GPUs and local GPUs: inference computation power provision for AI Models

AST Parser

Tree Sitter: language parsing for precise code context extraction and manipulation

Version Control Sys.

Bitbucket VCS on-prem: to enable controlled code changes with rollback capabilities

Automation Server

Jenkins on-prem: CI/CD automation server for triggering, executing and verifying repair workflows

Workflow

1 Configuration and Environment Setup - Load environment variables and configure commands.

2 Static Analysis Execution - Scan codebase for violations using SonarQube or similar tool.

3 Issue Parsing and Categorization - Normalize and map issues by rule ID, augment with compliance metadata.

4 Static AI Fix Generation - Batch violations by rule, extract context, and generate fixes using LLMs.

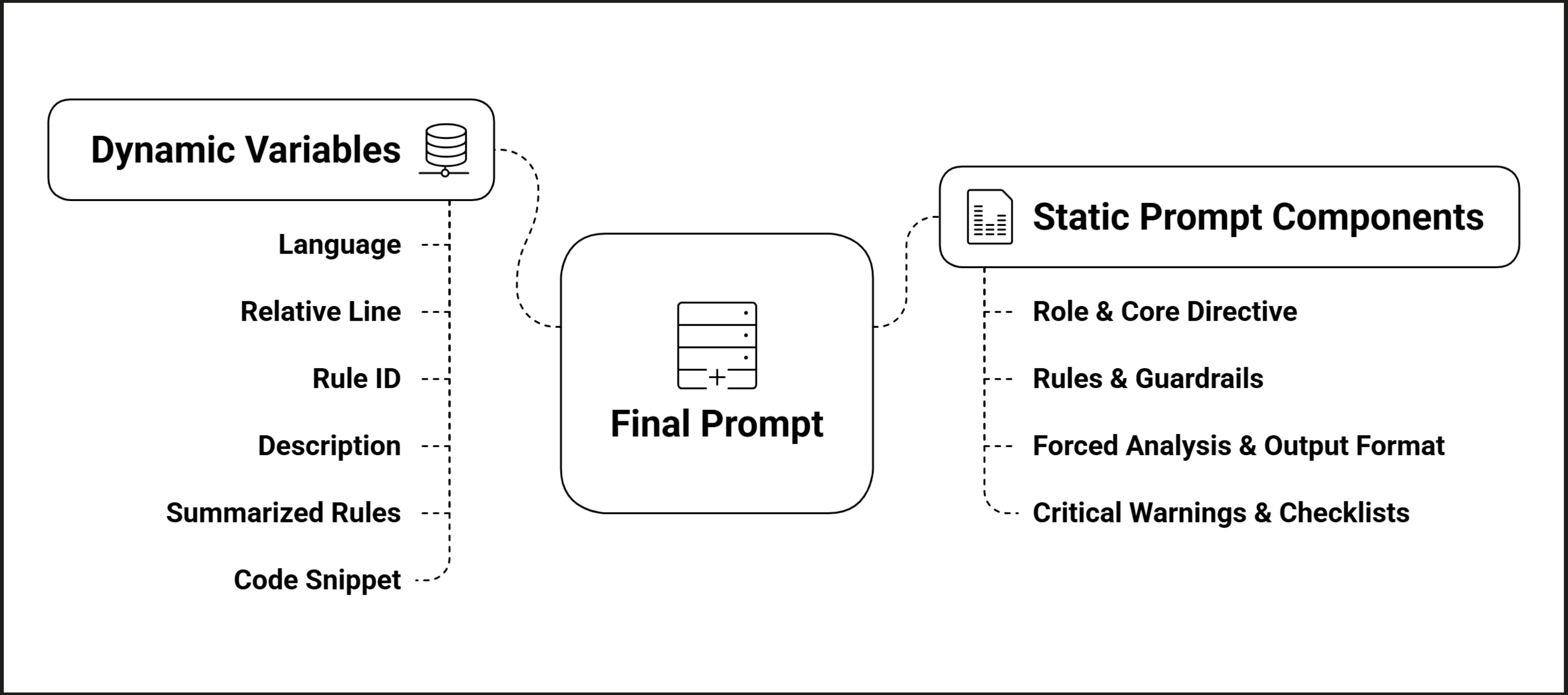
5 Fix Application with Offset Tracking - Apply patches with line offset, commit changes, and log actions.

6 Multi-Level Verification - Trigger Jenkins pipeline for build/test.

7 Error Recovery - Revert failed fixes

8 Reporting and Metrics Generation - Generate HTML/Markdown reports, diffs, and metrics.

Prompt Context Ensemble



Key Activities

1 Static Analysis Report Handling:

- Reads the SonarQube report from SONARQUBE_REPORT_FILE.
- Extracts:
    - File Path
    - Start & End Line
    - Issue Type & Description
- Groups issues by type to generate structured fix reports.

2 Code Extraction for Fixing

- If the file has ≤ 300 lines, it extracts the full file for better AI understanding.
- Otherwise:
    - Finds the enclosing function where the issue is located.
    - If no function is found, extracts ±10 lines around the issue

3 Generating AI Fixes:

- Uses Azure OpenAI API to generate fixes.
- Enforces structured response format.
- Ensures AI only returns necessary fixes instead of rewriting the entire file.

4 Storing Fixes in JSON:

- Fixes are saved into separate JSON files by issue type:
    - fixes_bug_<timestamp>.json
    - fixes_code_smell_<timestamp>.json
    - fixes_vulnerability_<timestamp>.json.

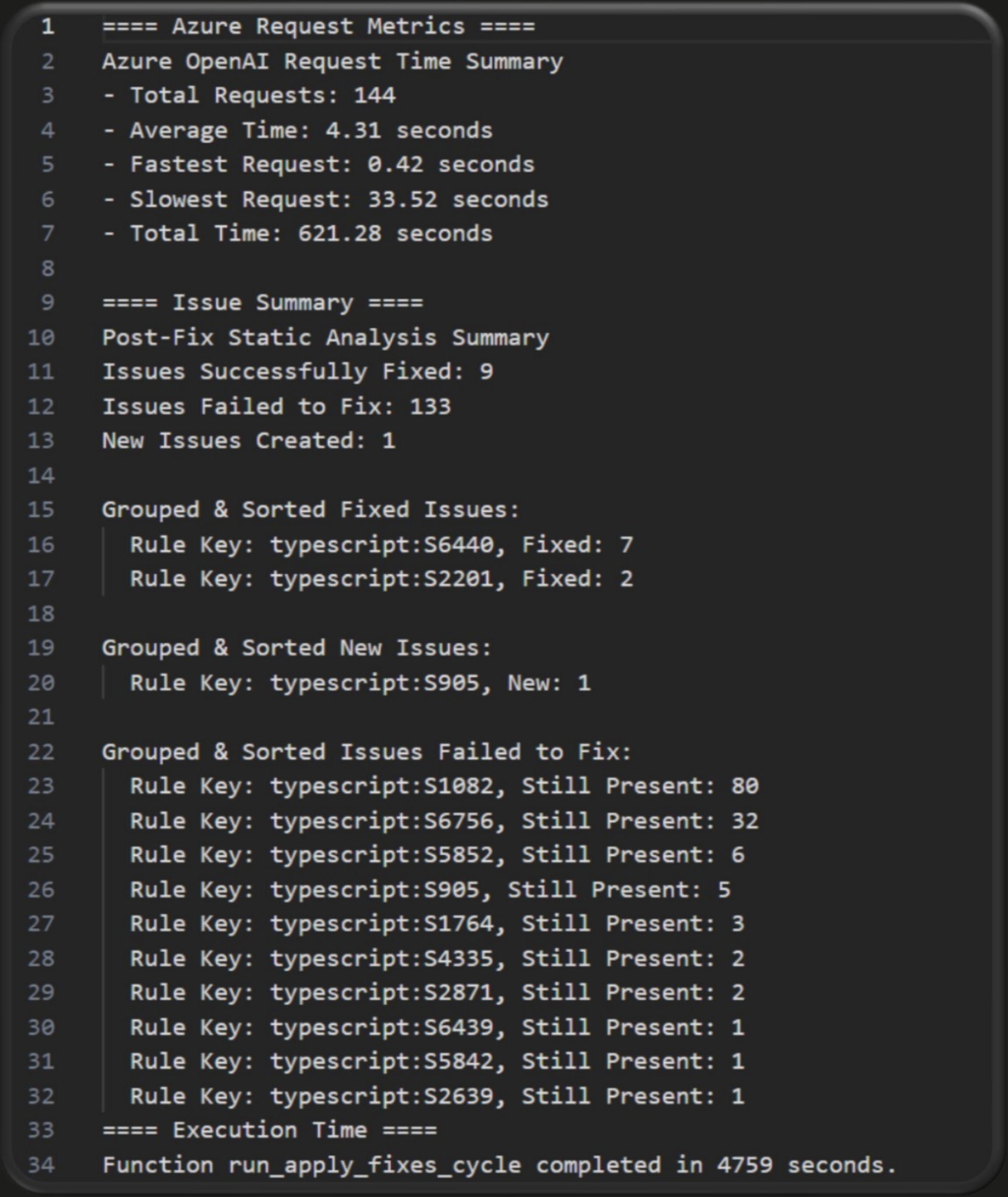
Dashboard & Reporting

Observability and Explainability

- The system generates both human-readable and machine-parsable JSON reports for different audiences.
- Full AI prompt logs available for transparency and auditing
- The logs also include the solution and an explanation

HTML Report Features:

- Fix success rates by issue type and code module
- Remaining violations (old and added)
- Performance statistics and execution times



```
==== Azure Request Metrics ====
Azure OpenAI Request Time Summary
- Total Requests: 144
- Average Time: 4.31 seconds
- Fastest Request: 0.42 seconds
- Slowest Request: 33.52 seconds
- Total Time: 621.28 seconds

==== Issue Summary ====
Post-Fix Static Analysis Summary
Issues Successfully Fixed: 9
Issues Failed to Fix: 133
New Issues Created: 1

Grouped & Sorted Fixed Issues:
  Rule Key: typescript:S6440, Fixed: 7
  Rule Key: typescript:S2201, Fixed: 2

Grouped & Sorted New Issues:
  Rule Key: typescript:S905, New: 1

Grouped & Sorted Issues Failed to Fix:
  Rule Key: typescript:S1082, Still Present: 80
  Rule Key: typescript:S6756, Still Present: 32
  Rule Key: typescript:S5852, Still Present: 6
  Rule Key: typescript:S905, Still Present: 5
  Rule Key: typescript:S1764, Still Present: 3
  Rule Key: typescript:S4335, Still Present: 2
  Rule Key: typescript:S2871, Still Present: 2
  Rule Key: typescript:S6439, Still Present: 1
  Rule Key: typescript:S5842, Still Present: 1
  Rule Key: typescript:S2639, Still Present: 1
==== Execution Time ====
Function run_apply_fixes_cycle completed in 4759 seconds.
```

Results

Four experimental projects were used to evaluate the results of the tool:

- 1: **Space Domain**, C, Software-on-Board (SoB) platform for spacecraft
- 2: **Space Domain**, C , Module for concurrent mission experiments using tech such as 6G, laser comms, IoT
- 3: **Energy Domain**, C#, Core module of a web-based system for energy, building, and maintenance management
- 4: **Energy Domain**, JS/TS, User portal client module for interaction and visualization of the state of all objects received

Projects	Size (KLOC)	I. Issues	I. Maintainability	I. Security HS	I. Coverage (%)	I. Duplicated Lines (%)
1	~170	33	243	49	96,0%	5,3%
2	~28	18	425	275	74,1%	3,7%
3	~7	1	93	14	7,0%	0,8%
4	~30	138	529	6	0,0%	9,2%

Projects	F. Issues	F. Maintainability	F. Security HS	F. Coverage (%)	F. Duplicated Lines (%)
1	8	153	31	86,5%	5,4%
2	5	107	15	74,6%	4,0%
3	1	93	14	7,0%	0,8%
4	54	460	6	0,0%	7,7%

Key Benefits & Conclusion

Technical Benefits

- Portable script-based solution configurable via .env and CLI
- Transparent AI prompt logging for auditability
- Standards-aware fixes ensuring compliance
- Git prevents uncontrolled AI behavior

Business Impact

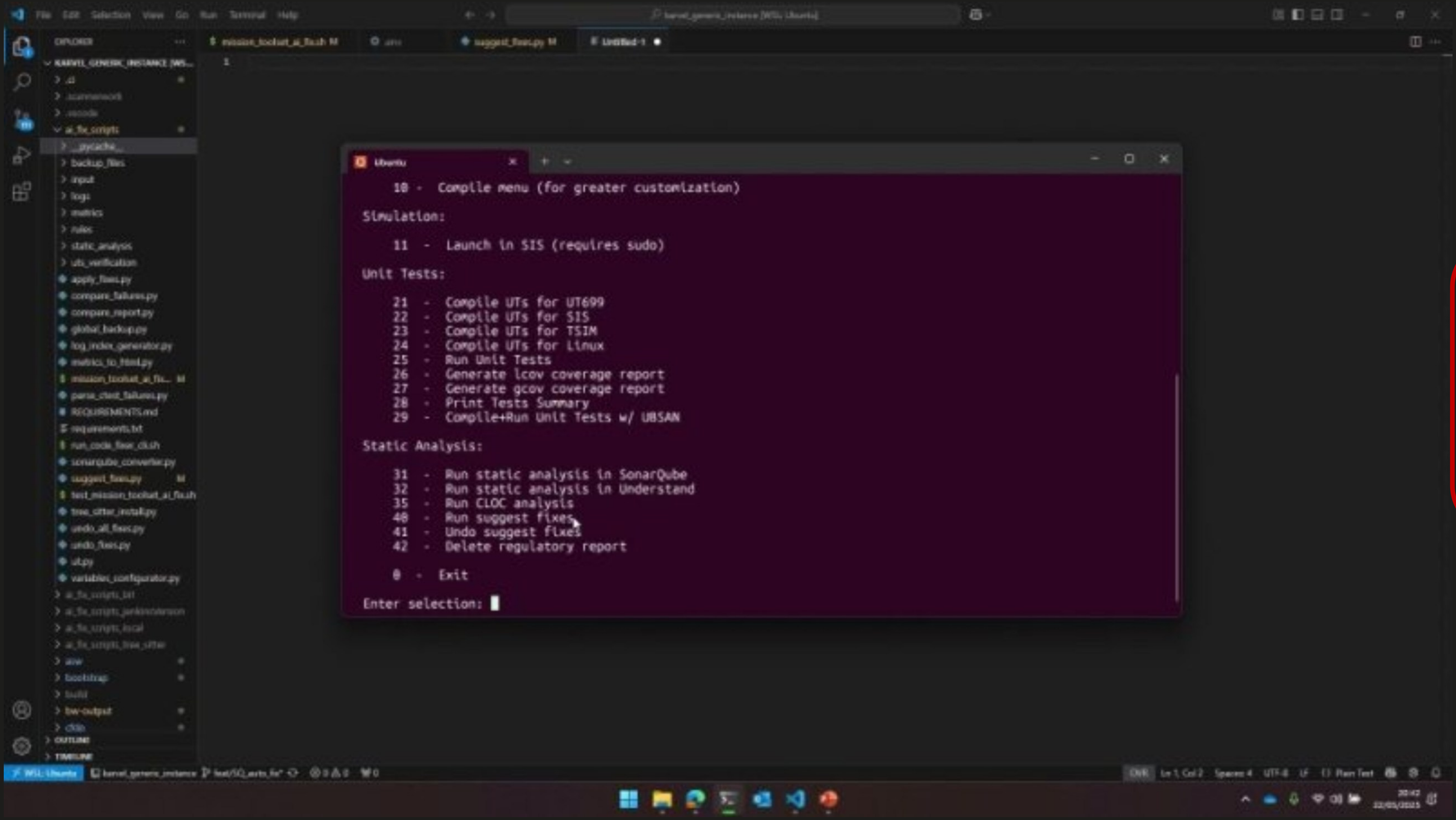
- Reduced triage time when # of violations is high
- Enhanced developer productivity and focus
- Adaptable integration with existing workflows
- Extensible architecture supporting multiple languages

Limitations

- Current focus on MISRA rules for C/C++.
- Verification depends on the quality of the tests.
- Fixes may pass tests but change the behavior.
- Some fixes require global context (global variables, dependencies between files) that exceed prompt limits.
- LLM models may generate syntax errors or apply incorrect conventions.

The AI-driven code repair solution automates and audits code fixes, empowering development teams to maintain high quality standards without sacrificing the schedules.

Demonstration



```
    10 -  Compile menu (for greater customization)

Simulation:

    11  -  Launch in SIS (requires sudo)

Unit Tests:

    21  -  Compile UTs for UT699
    22  -  Compile UTs for SIS
    23  -  Compile UTs for TSIM
    24  -  Compile UTs for Linux
    25  -  Run Unit Tests
    26  -  Generate lcov coverage report
    27  -  Generate gcov coverage report
    28  -  Print Tests Summary
    29  -  Compile+Run Unit Tests w/ UBSAN

Static Analysis:

    31  -  Run static analysis in SonarQube
    32  -  Run static analysis in Understand
    35  -  Run CLOC analysis
    40  -  Run suggest fixes
    41  -  Undo suggest fixes
    42  -  Delete regulatory report

    0  -  Exit

Enter selection:
```

Experimental project with added issues, code smells and security hotspots.

Lessons Learned - Conclusions

Lessons Learned:

- Scope Limitation: Rule Coverage and Validation Context
- Submodules and Build Complexity
- Fix Safety and Semantic Preservation
- Context Limitations and LLM Behavior
- Language and Rule Set Expansion

Conclusions:

- Suitable for specific and repetitive fixes
- Suitable for providing solutions/alternatives to the engineers
- Improvement depends on amount of issues and automation
- Such a solution cannot substitute the standard process, AI tools can introduce wrong solutions: human verification is needed

The next slide provides a set of specific concerns and topics relevant for Space Engineering.

Top 10 Challenges

1. Tool Qualification and Determinism by using an AI model (Q80C, §6.4.9)
2. Verification, Validation, and the Sufficiency of Testing
3. Formal Compliance with ECSS Standards
4. Complexity of Fixes and Architectural Impact
5. Traceability and Design Authority
6. Security Implications and Vulnerability Injection
7. Human-in-the-Loop and Code Ownership + Skill Atrophy
8. Auditability vs. Explainability
9. Impact on Non-Functional Requirements
10. Management of the AI Model(s) as COTS Components

Contacts:

Rafael Fonseca

rafaelfonsecajp@gmail.com

https://www.linkedin.com/in/rafaelfonseca-/

Computer Engineering Master student @ ISEC

Pedro Reis Nunes

pedro.r.nunes@criticalsoftware.com

https://www.linkedin.com/in/umpedronunes/

AI Engineer @ Critical Software

Nuno Silva

nsilva@criticalsoftware.com

https://www.linkedin.com/in/nunopedrojesussilva/

Engineering Manager @ Critical Software

Q&A

Thanks for your attention!