International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Hanzoly: An AI-Powered Software Handover and Understanding Platform

Rishuraj Sunilkumar Pandey, Ashish Jitendra Mohite, Shrinivas Maroti Kadam, Aditya Bharat Shinde
Department of Artificial Intelligence and Machine Learning, Zeal Polytechnic, Pune, Maharashtra, India
Academic Year 2026–2027 | Project Guide: **Vaishnavi D. Sutar**

Abstract— Software handover can be difficult when project knowledge is held mainly by the outgoing developer or scattered across source files, documentation, and tools. HANZOLY is proposed as an
AI-powered software handover and codebase-understanding platform that scans an existing project locally, extracts technical structure, and organizes the findings into a handover report. For supported Python files, Python's Abstract Syntax Tree (AST) can identify functions, classes, and import statements. Dependency analysis helps describe relationships between modules, while a locally hosted large language model (LLM) can turn extracted technical information into readable explanations. As a supporting capability, HANZOLY is designed to detect code changes using Git Diff, identify potentially affected components, and compare test outcomes using Pytest. Optional security checks may flag exposed secrets and potentially vulnerable dependencies. This paper presents the proposed workflow, architecture, and evaluation approach. It does not claim measured performance; effectiveness must be established through implementation and testing.

Keywords— software handover, codebase understanding, local AI, Python AST, dependency analysis, change impact analysis, regression testing.

I. INTRODUCTION

Software projects are frequently transferred between developers or teams. The incoming developer must understand the directory structure, source code, dependencies, tests, and operational assumptions before making safe changes. When this knowledge is incomplete or spread across informal conversations and separate tools, onboarding and maintenance can take additional effort.

HANZOLY focuses first on software handover. It is intended to inspect an existing project in a local or private environment, collect useful technical facts, and produce a structured report for the next developer. Change impact analysis is a supporting feature that helps investigate what might be affected when code changes.

The proposed approach combines deterministic extraction—such as parsing Python source with AST—with dependency information, test results, and optional localLLM explanations. Separating extracted facts from generated explanations is important: AI-generated text should support understanding, while source files, dependency records, Git changes, and test outputs remain the evidence.

II. PROBLEM STATEMENT AND OBJECTIVES

A handover may rely on manual documentation and explanations from the previous developer. If these are incomplete, the incoming developer must reconstruct the project's structure and relationships independently. Separate tools may help with code navigation, version comparison, testing, or security checks, but the resulting information still needs to be organized into a useful handover view.

A. Objectives

The objectives of HANZOLY are to: (1) scan a selected project directory locally; (2) extract supported source-code structure; (3) organize module and dependency information; (4) use a local LLM to explain extracted findings; (5) generate a structured handover report; (6) detect modified files and sections through Git Diff; (7) highlight potentially affected components; (8) compare test outcomes before and after changes; and (9) provide optional findings from secret and dependency scanners.

B. Scope

The synopsis identifies Python AST as the initial codestructure analysis method. Therefore, the proposed ASTbased extraction is Python-focused; support for other languages would require additional parsers. Dependency and impact findings are indicators for developer review, not proof that every runtime relationship or regression has been detected.

III. EXISTING AND PROPOSED SYSTEM A. Existing System

Common handover practices include meetings, written documentation, manual source-code inspection, and separate code-quality or testing tools. These practices can be useful but may leave important knowledge undocumented or require the incoming developer to consolidate information manually. Dependency discovery and change verification can also require separate investigation.

B. Proposed System

HANZOLY proposes a connected workflow: select a project, scan its files, extract code structure, analyze module relationships, request explanations from a local LLM, and generate a handover report. When a change is made, Git Diff provides the change set; dependency information helps identify possible impact; and Pytest results before and after the change help surface potential regressions. Optional security scanners can contribute findings for further review.

C. Expected Benefits

The intended benefits are a more organized handover, reduced dependence on informal explanations, faster orientation within unfamiliar code, and clearer evidence for maintenance decisions. These are design goals, not measured outcomes at this stage.

IV. METHODOLOGY

A. Project Selection and Local Scanning

The developer selects an existing project directory. The scanner inventories folders, supported source files, and available tests.

B. Code Structure Extraction

Python AST is used to parse supported Python files and identify definitions such as functions and classes, along with import statements. Parsing results are stored

as structured data so that later stages can use the same extracted facts consistently.

C. Dependency Analysis

Import and module relationships are organized into a dependency view. These relationships can help explain how components connect and can support impact investigation. Static relationships may not capture dynamic imports, reflection, configuration-driven loading, or all runtime calls; the report should communicate such limitations.

D. Local AI Explanation

A local LLM, for example one served through Ollama, can explain extracted project facts in simpler language and summarize module roles. To improve traceability, explanations should refer to files and symbols from the extracted data, and uncertain statements should be labelled for manual verification.

E. Handover Report Generation

The report brings together project structure, important code elements, dependency information, available tests, and AI-generated explanations. The report should distinguish automatically extracted facts from model-generated interpretation and list unsupported or skipped files where relevant.

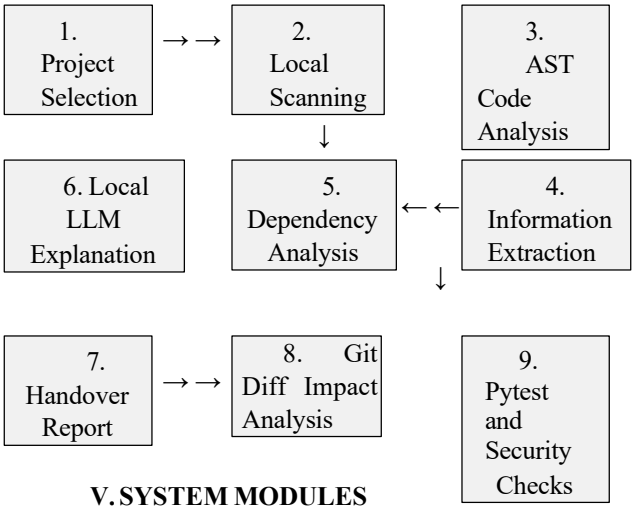
F. Supporting Change Impact Analysis

Git Diff identifies changed files and code sections between selected revisions. HANZOLY then uses available dependency information to highlight potentially affected modules, functions, and tests. This is a prioritization aid: developers must review the proposed impact list before relying on it.

G. Testing and Security Checks

Pytest can run the project's available tests before and after a change. The system compares pass/fail outcomes and reports newly failing tests as possible regression signals. Optional secret and dependency scanners may identify exposed credentials or known vulnerable packages; each finding should include its source and require validation

E. Overall workflow of the proposed Hanzoly platform.



V. SYSTEM MODULES

A. Project Scanner and Code Analyzer

The scanner inventories the project, locates supported Python files and test files, and passes source files to the AST analyzer. The analyzer extracts functions, classes, and imports into structured records.

B. Technical Information and Dependency Module

This module organizes the extracted records and maps available relationships between modules. The resulting information supports both the handover report and the supporting impact analysis.

C. Local AI Explanation and Report Module

The local AI component generates readable descriptions from extracted facts. The report module assembles these descriptions with source-derived information, dependency details, test status, and known limitations.

D. Change, Test, and Security Modules

The change detector reads Git Diff output; the impact module maps changes to potentially related components; the test module compares Pytest results; and optional security integrations record secret or dependency findings. The precise security checks depend on the tools integrated in the implementation.

HANZOLY MODULES AND TECHNOLOGIES

Module	Technology	Output
Project Scanner	Python	File and test inventory
Code Analysis	Python ast [4]	Functions, classes, imports
Dependency Analysis	Python	Module relationship map
AI Explanation	Local LLM via Ollama [17]	Plain-language summaries
Report Generation	FastAPI [18], HTML/CSS/JS, JSON, SQLite	Handover report
Change Detection	Git Diff [15]	Changed files and sections
Impact and Testing	Pytest [16]	Before/after test comparison

VI. TOOLS AND TECHNOLOGIES

Python and AST: backend logic and structured parsing of supported Python source files. **FastAPI:** a possible API layer between the interface and analysis modules. **HTML, CSS, and JavaScript:** possible technologies for the web interface. **Git/Git Diff:** version tracking and change identification. **Pytest:** execution and comparison of available tests. **Local LLM/Ollama:** optional locally hosted explanations. **JSON/SQLite:** possible formats for storing extracted data and analysis records. **Security scanners:** optional tools for secret and dependency checks. These technologies describe the proposed design; final integrations depend on the implemented version.

VII. EVALUATION PLAN AND LIMITATIONS

Evaluation should be conducted after implementation using representative Python projects with known structure, dependencies, tests, and controlled changes. Useful measures include: (1) whether extracted functions, classes, and imports match manual inspection; (2) whether dependency links are correctly identified; (3) whether report sections are complete and traceable; (4) whether known

changed files and likely dependent modules appear in the impact output; (5) whether test-result differences are reported correctly; and (6) whether optional security findings can be reproduced and validated.

Results should be recorded with the project used, test environment, number of files inspected, expected and observed outputs, and limitations. No benchmark values are included in this paper because the supplied project synopsis does not provide completed experimental results. Static analysis may miss runtime behaviour, and LLM explanations can be incomplete or incorrect; human review remains necessary.

VIII. CONCLUSION

HANZOLY is proposed as an AI-powered software handover and understanding platform. Its primary workflow scans an existing project locally, extracts code structure, organizes dependencies, uses a local LLM to explain technical findings, and produces a structured handover report. Change impact analysis, Git Diff, Pytest comparison, and optional security checks provide supporting maintenance information. The approach aims to organize knowledge transfer while keeping analysis in a local or private environment when configured accordingly. Future work should implement the modules, evaluate them on representative projects, measure extraction and impactanalysis quality, and document privacy and security boundaries.

REFERENCES

[1] Python Software Foundation, “ast — AbstractSyntax Trees,” Python Documentation. Available: https://docs.python.org/3/library/ast.html

[2] pytest Development Team, “pytest documentation.” Available: https://docs.pytest.org/

[3] Git Project, “git-diff — Show changes betweencommits, commit and working tree, etc.” Available: https://git-scm.com/docs/git-diff

[4] FastAPI, “FastAPI Documentation.” Available: https://fastapi.tiangolo.com/

[5] Ollama, “Ollama Documentation.” Available: https://docs.ollama.com/

[6] Python Software Foundation, “json — JSONencoder and decoder,” Python Documentation. Available: https://docs.python.org/3/library/json.html