An Evaluation of Program Comprehension Techniques on a Large-Scale System

Zihan Kuang
Chalmers University of Technology
zihank@student.chalmers.se

***Abstract*—Comprehending large-scale, open-source software systems like Home Assistant presents a significant challenge. This report documents an independent exploration of three comprehension techniques: static code analysis, architecture visualization, and documentation mining. Using SonarQube, pydeps, and the official developer documentation, I analyze the Home Assistant Core repository. The results demonstrate that a combination of these techniques is essential for holistic understanding. SonarQube provides a quantitative, code-level health assessment; pydeps reveals the macroscopic architectural structure; and documentation mining provides the crucial design intent and context. This evaluation compares their strengths and limitations, reflecting on how their synthesis provides a comprehensive view that connects abstract concepts like technical debt and modularity to concrete evidence.**

I. Introduction

The ability to efficiently understand existing code is a cornerstone of software engineering. For large, long-lived open-source systems such as Home Assistant, this task is particularly daunting due to the sheer volume of code and complex inter-dependencies. The goal of this assignment is to independently evaluate different program comprehension techniques and tools to tackle this complexity. This report focuses on the Home Assistant Core, applying and critically analyzing techniques across three dimensions—code, architecture, and documentation—to gain a multi-faceted understanding of the system's structure and quality.

II. Methods and Tools

To cover different dimensions of comprehension, I selected three primary techniques and their corresponding tools. The selection process also involved exploring several other tools that were ultimately abandoned.

A. Selected Techniques and Tools

- **Static Code Analysis:** This technique inspects code without executing it to find defects and quality metrics. I chose **SonarQube**, an industry-standard platform, for its comprehensive dashboards and ability to quantify abstract concepts like maintainability.
- **Architecture Visualization:** This technique focuses on recovering and visualizing the high-level structure and dependencies between modules. I selected **pydeps**, a Python-specific tool, to uncover the macro-level relationships invisible at the code level.
- **Documentation Mining:** This technique involves systematically extracting information from human-written project artifacts. For this, the "tools" were a combination of search/navigation strategies applied to the project's official developer documentation, repository files like `README.rst`, and in-code docstrings to understand the developers' intent and design philosophy.

B. Tool Exploration and Rationale for Abandonment

The tool selection was an iterative process. Initial exploration revealed that generating useful visualizations for a project of this scale is a non-trivial and computationally expensive task. Several tools were attempted and abandoned:

- **PyReveng3 & pycg:** These call graph generators proved difficult to use. PyReveng3 required a complex setup and custom scripting, while 'pycg' suffered from environment compatibility issues with a modern Python version (3.13).
- **CodeGraphy (VS Code Plugin):** This tool was found to be unsuitable as it primarily targets JavaScript/TypeScript projects and failed to produce meaningful output for the Python-based Home Assistant Core.

This preliminary exploration led to a key insight: for massive projects, a "brute-force" analysis approach is impractical. A more effective strategy is to first form hypotheses about the system's core components and then use targeted tool configurations to validate them.

III. Results: Applying Tools to Home Assistant Core

A. Static Analysis with SonarQube

A successful scan of the Home Assistant Core repository was performed using SonarQube (see Appendix Fig. 7). The 'Overview' dashboard (Fig. 1) acts as a project "health report," immediately indicating that the project passed its quality gate. To understand the details, the 'Issues' tab allows for micro-level investigation. Fig. 2 shows a specific "Code Smell" in the 'auth' module, demonstrating SonarQube's utility in guiding developers towards

concrete code improvements. The ‘Measures‘ dashboard (Fig. 3) further visualizes modules across multiple risk dimensions, allowing for a data-driven approach to identifying high-risk areas.

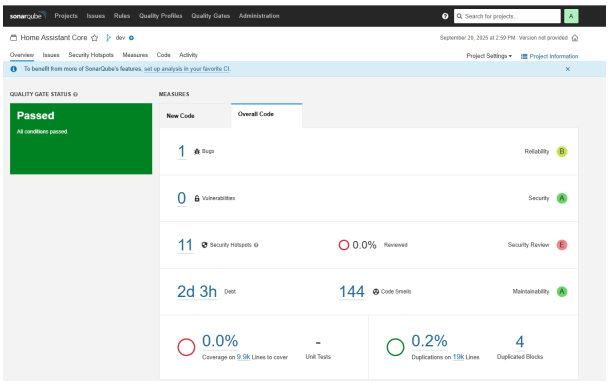

Fig. 1. SonarQube’s main ‘Overview‘ dashboard for Home Assistant Core.

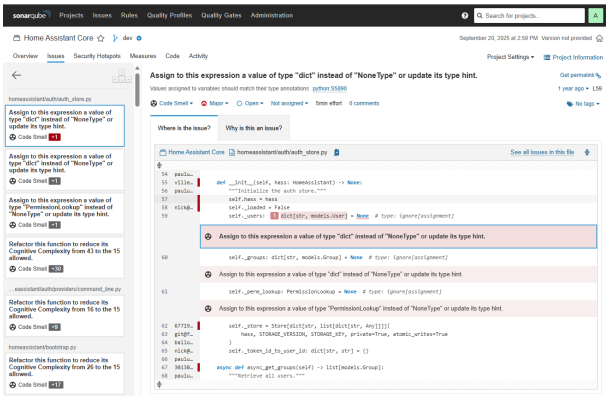

Fig. 2. A specific "Code Smell" example in `auth_store.py`.

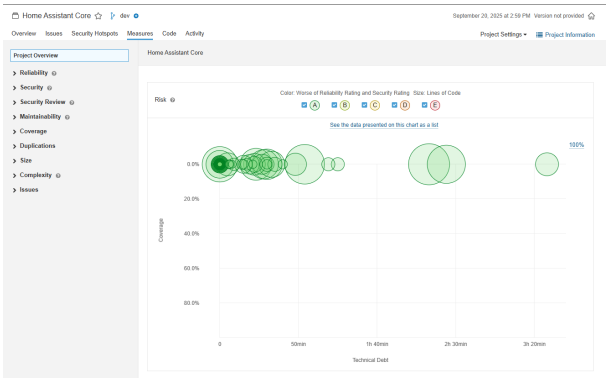

Fig. 3. The ‘Measures‘ bubble chart for multi-metric risk assessment.

B. Architecture Visualization with pydeps

To create a meaningful visualization, an iterative refinement of the ‘pydeps‘ command was necessary to avoid a massive, unreadable graph (Listing 1). The resulting graph (Fig. 4) reveals the core architectural structure of the system, successfully visualizing its backbone and identifying central "hub" modules like ‘core‘ and ‘helpers‘.

```
:~/homeassistant-core$ pydeps --cluster -o
    ha_core_deps_focused.svg \
    -T svg \
    --max-module-depth 2 \
    -x "homeassistant.components.*" \
    -x "homeassistant.generated.*" \
    -x "script.*" \
    -x ".github.*" \
    -x "venv.*" \
    -x "tests.*" \
    -x "aiohttp" \
    -x "requests" \
    -x "voluptuous" \
    -x "jinja2" \
    -x "yaml" \
    -x "urllib3" \
    -- \
    homeassistant
```

Listing 1. Refined pydeps command.

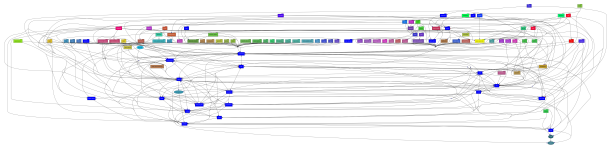

Fig. 4. Dependency graph of Home Assistant’s core modules.

C. Documentation Mining

To complement the automated analysis, documentation mining was performed to understand the intended architecture. The official developer documentation provided crucial high-level context. The core architecture diagram (Fig. 5) explains the system’s foundational pillars—the Event Bus, State Machine, and Service Registry. This conceptual knowledge is essential for correctly interpreting the dependency graphs from ‘pydeps‘. The documentation further explains how this core is extended via integrations (Fig. 6), clarifying the project’s philosophy of modularity.

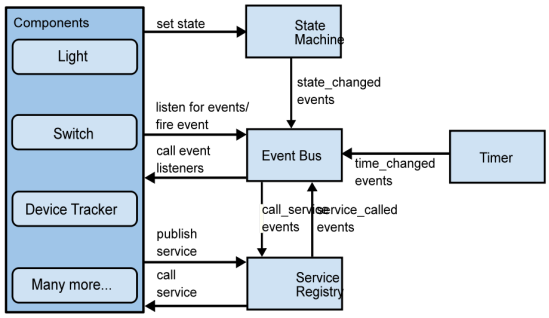

Fig. 5. The core architecture diagram from the developer documentation, explaining the system’s foundational pillars.

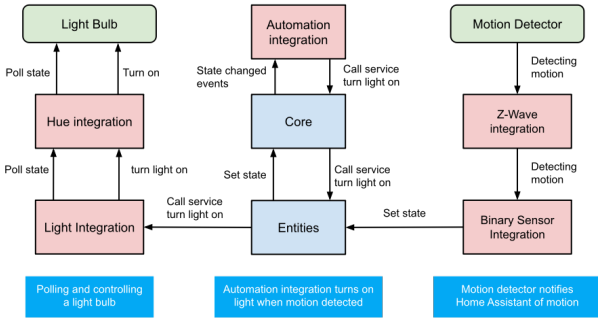

Fig. 6. The integration architecture diagram, showing how the core is extended with new functionality like the Hue integration.

IV. Comparison and Analysis of Techniques

The three techniques provide different and complementary perspectives on the system.

- **Information Type:** SonarQube provides quantitative code quality data. Pydeps provides qualitative structural dependency data. Documentation Mining provides human-authored design intent and conceptual models.
- **Usefulness:** Pydeps was most helpful for gaining an initial high-level map of the system. The documentation was crucial for understanding the *meaning* behind the structure revealed by pydeps. SonarQube was best for drilling down into the implementation quality of specific modules once they were identified as architecturally significant.
- **Limitations:** SonarQube lacks architectural context. Pydeps shows connections but not their purpose. Documentation's primary limitation is that it can become outdated; the outputs from the other tools can serve as a "ground truth" to check if the documentation accurately reflects the current code.

In summary, 'pydeps' provides the blueprint, SonarQube provides the inspection report for each room, and the documentation provides the architect's notes explaining why the building was designed that way.

V. Reflection

This exploration provided several key insights. First, program comprehension is an iterative, hypothesis-driven process. The initial failure to generate a useful 'pydeps' graph highlighted the need to first form a mental model of the system—often from documentation—before a tool can effectively validate or refine it.

Second, the findings provide tangible examples of established software engineering concepts. The technical debt metrics from SonarQube are a practical quantification of the abstract concept of technical debt. Similarly, the dependency graph from pydeps can be viewed as a map of the project's socio-technical structure, an observation that aligns with the principle that a system's architecture often mirrors the communication structure of the organization that built it.

The most significant lesson is that no single tool is a silver bullet. A holistic understanding only emerges from synthesizing the microscopic view of code quality (SonarQube), the macroscopic view of system architecture (pydeps), and the crucial human-authored context provided by the documentation.

Appendix

This section contains supplementary materials that support the analysis.

```
INFO: 11/171 source files have been analyzed
INFO: 18/171 source files have been analyzed
INFO: 29/171 source files have been analyzed
INFO: 38/171 source files have been analyzed
INFO: 50/171 source files have been analyzed
INFO: 64/171 source files have been analyzed
INFO: 80/171 source files have been analyzed
INFO: 92/171 source files have been analyzed
INFO: 104/171 source files have been analyzed
INFO: 114/171 source files have been analyzed
INFO: 128/171 source files have been analyzed
INFO: 140/171 source files have been analyzed
INFO: 152/171 source files have been analyzed
INFO: 164/171 source files have been analyzed
INFO: SCM Publisher 171/171 source files have been analyzed (done) | time=143811ms
INFO: CPD Executor 13 files had no CPD blocks
INFO: CPD Executor Calculating CPD for 99 files
INFO: CPD Executor CPD calculation finished (done) | time=95ms
INFO: Analysis report generated in 145ms, dir size=4.7 MB
INFO: Analysis report compressed in 604ms, zip size=2.2 MB
INFO: Analysis report uploaded in 119ms
INFO: ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=homeassistant-core
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=AZlnOAH1Bpt4Vj68muHc
INFO: Analysis total time: 2:55.302 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 2:56.886s
INFO: Final Memory: 39M/148M
INFO: ------------------------------------------------------------------------
(venv) azureuser@DAT266:~/homeassistant-core$
```

Fig. 7. Console log showing the successful execution of the 'sonar-scanner' command.

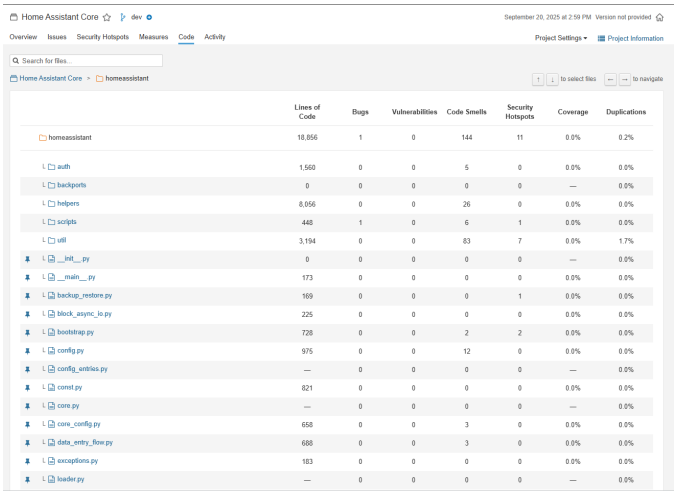

Fig. 8. Detailed breakdown of metrics per folder and file in the SonarQube 'Code' tab.

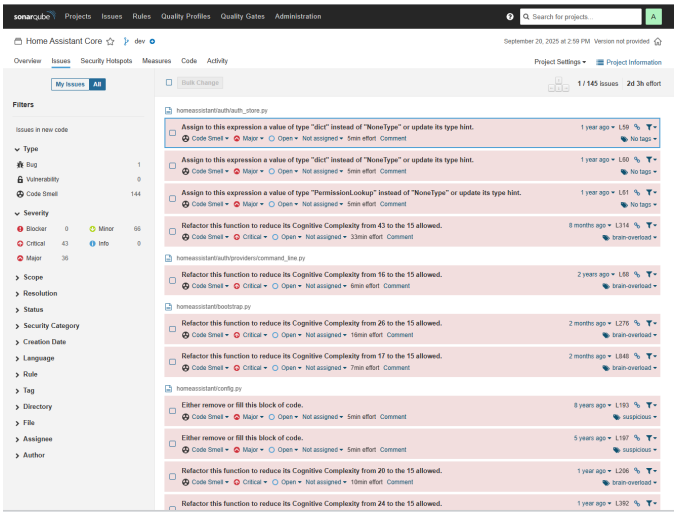

Fig. 9. The ‘Issues‘ tab showing a filterable list of all identified code smells, bugs, and vulnerabilities.

```
(venv) azureuser@DAT266:~/homeassistant-core$
pydeps --cluster -o ha_core_deps_refined.svg \
    -T svg \
    --max-module-depth 2 \
    --rmprefix homeassistant. \
    -x "homeassistant.components.*" \
    -x "homeassistant.generated.*" \
    -x "homeassistant.brands.*" \
    -x "homeassistant.scripts.*" \
    -x "homeassistant.util.*" \
    -x "tests.*" \
    -x "aiohttp" \
    -x "requests" \
    -x "voluptuous" \
    -x "jinja2" \
    -x "yaml" \
    -x "urllib3" \
    -- \
    homeassistant
(venv) azureuser@DAT266:~/homeassistant-core$
```

Fig. 10. Console log showing the execution of the refined ‘pydeps‘ command.