Automating Bug Analysis in Regulatory Information Requirements Using AI Agents

Wendell Marques
Sidia Institute of Technology
Manaus, Brazil
wendell.marques@sidia.com

Caina Oliveira
Sidia Institute of Technology
Manaus, Brazil
caina.oliveira@sidia.com

Edluce Veras
Sidia Institute of Technology
Manaus, Brazil
edluce.leitao@sidia.com

Eduardo Oliveira
Sidia Institute of Technology
Manaus, Brazil
eduardo.oliveira@sidia.com

Thiago Fragata
Sidia Institute of Technology
Manaus, Brazil
thiago.fragata@sidia.com

Abstract

This paper proposes an agent-based system for the automated validation of homologation compliance in Android software builds. To address the inefficiencies of manual verification, the proposed system employs an autonomous agent equipped with specialized tools to interface with internal systems, enabling the systematic analysis and cross-referencing of embedded assets and device parameters against regulatory specifications. Consequently, the system eliminates manual intervention in the verification of regulatory information requirements. The findings suggest that the delegation of compliance tasks to specialized AI agents enhances operational reliability and optimizes the quality assurance lifecycle in large-scale mobile development.

Keywords

Homologation, Mobile Device Compliance, Autonomous AI Agents, React Paradigm, Langchain, Large Language Model Orchestration

1 Introduction

The expansion of mobile ecosystems requires manufacturers to comply with a complex matrix of regulatory mandates across diverse jurisdictions and carriers. In many markets, formal homologation is mandatory, requiring the embedding of a specific image containing the product name and a unique homologation code into the software build. Non-compliance can lead to severe legal sanctions and financial losses. Currently, validation relies on manual, multi-step workflows where testers cross-reference spreadsheet data with source repositories through visual inspection. This process is inherently error-prone and resource-intensive, creating a critical need for an automated and reliable verification solution.

In recent years, Large Language Models (LLMs) have significantly impacted Software Engineering (SE) by automating previously manual, labor-intensive tasks[**?**][2][5]. However, the utility of standalone LLMs is often constrained by finite context and a lack of real-time interaction with external environments[3]. To overcome these constraints, the paradigm of AI Agents has emerged, augmenting the reasoning capabilities of LLMs with the ability to orchestrate development tools, access external knowledge bases, and leverage domain-specific resources. This agentic approach enables the delivery of comprehensive, end-to-end solutions for complex, real-world SE challenges[6]. These agents operate as autonomous systems that leverage LLMs to iteratively reason through complex problems, invoking specialized tools to interface with external data sources and dynamically adapting their execution paths based on the resulting outputs[1].

To address the challenges mentioned in homologation validation, this research proposes RICH (Regulatory Info Checker), an autonomous AI-agent system designed to automate the validation of homologation compliance. By implementing the ReAct (Reasoning and Acting) paradigm, RICH orchestrates a suite of specialized tools that allows it to perform internal system queries, binary extraction, and automated image analysis. This enables the system to accurately validate the codes embedded in the software binary, reducing manual overhead and increasing the reliability of the compliance process.

2 Contextualization

This research addresses the operational challenges of regulatory compliance within the Android technology sector, specifically regarding regulatory compliance in specific regional markets requiring homologation. Compliance in these markets requires the programmatic embedding of homologation assets into the device software. This requirement dictates that each software build must incorporate an image following visual specifications (model name, homologation code, proper positioning) to ensure legal market entry.

Ensuring the accuracy of this embedded data is imperative, as releasing software without the proper embedded image or with an incorrect homologation code can lead to significant legal and financial repercussions for the manufacturer. Consequently, rigorous validation testing is conducted during the implementation phase as part of the standard verification workflow to ensure compliance before device commercialization is authorized.

Historically, the validation of this requirement relies entirely on manual intervention by testers. The existing workflow necessitates the manual orchestration of some data sources: Extract ticket info (model, SKU, and OS version), query a internal spreedsheet to extract homologation code from device under test, and navigating source code repositories to locate the corresponding image. The final validation phase requires a human operator to perform a visual cross-reference between the retrieved metadata and the rendered image, introducing subjectivity into a process that demands absolute precision.

While traditional automation could address linear data retrieval, the heterogeneity of the sources and the need for semantic validation of visual assets require a more flexible, autonomous approach. This research proposes an automated approach leveraging Artificial Intelligence (AI) agents to orchestrate the verification process. The proposal introduces a system capable of using tools dynamically to interface with external systems and databases to retrieve necessary information, cross-reference data integrity, and generate analysis reports. By transforming a manual verification chain into an autonomous agentic workflow, this approach aims to eliminate human-induced bottlenecks, enhance the reliability of regulatory compliance, and ensure scalable validation.

3 Proposal

This research proposes the development of an autonomous AI Agent architecture designed to orchestrate a suite of specialized tools for the automated validation of critical regulatory requirements. In our current test processes, manual homologation code verification represents a significant operational bottleneck; therefore, this system is designed to execute verification tasks asynchronously and in parallel with standard testing procedures.

The proposed system is triggered via an event-driven mechanism: upon the initiation of a test execution task, a background request transmits the unique Ticket Identifier (ticket ID) to the Agent. Utilizing this identifier, the Agent employs a reasoning loop to dynamically invoke a set of functional extensions (tools) to query internal repositories and validate compliance. The Agent is responsible for validating the presence of the required homologation metadata within the binary, ensuring both its correct embedding and its adherence to regulatory structural and visual specifications.

Unlike rigid, sequential workflows, the architecture treats these tools as atomic capabilities accessible through the Agent's reasoning process. This design allows the system to autonomously determine the optimal sequence of tool invocation to retrieve and synthesize the information required for a final compliance determination. By decoupling the regulatory verification from the manual testing pipeline, the proposed solution mitigates human latency and enhances the throughput of the homologation cycle. The specialized toolset integrated into the Agent's environment will be detailed in section 4.

4 Methodology

This section details the architecture, tools and evaluation metrics used.

4.1 System Architecture

The system was implemented using Python 3.12, leveraging the LangChain and LangGraph frameworks to implement an agent architecture based on the ReAct (Reasoning and Acting) paradigm. This approach enables a cognitive loop where the agent can iteratively decompose complex queries, execute targeted actions, and refine its reasoning based on the observed output of those actions.

The agent's cognitive core is powered by the gpt-oss-120b model from OpenAI, deployed locally using the vLLM serving engine. Local deployment was strategically chosen to optimize inference throughput and ensure strict data governance, preventing the exposure of sensitive regulatory information to external APIs.

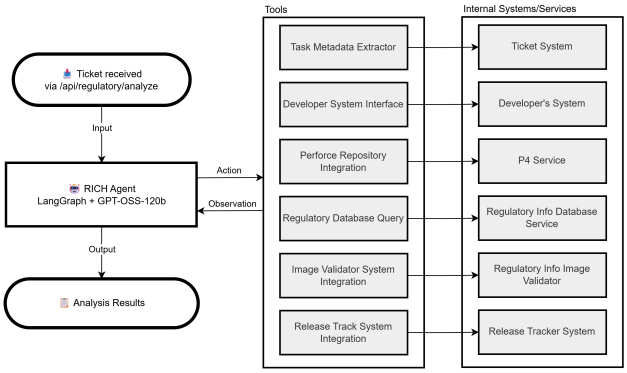

Figure 1: RICH Agent Architecture

To ensure modularity and scalability, each tool was designed as an autonomous module, specifically engineered to interface with external systems and retrieve the precise data required for the verification of regulatory information requirements.

The system's entry point is a REST API endpoint implemented via Flask, which triggers the execution pipeline upon receiving a specific ticket ID, the architecture diagram is detailed in figure 1.

4.2 Tool Details

(1) **Task Metadata Extractor:** Interfaces with a task tracker (e.g., Jira) to retrieve device model, carrier (SKU), Android version, and release identifiers.
(2) **Developer System Interface:** Accesses development environment metadata via REST API to retrieve the Perforce (P4) depot path associated with the project configuration.
(3) **Perforce Repository Integration:** Interfaces with internal Perforce service via REST API to locate and retrieve regulatory images from source code repositories.
(4) **Regulatory Database Query:** Retrieves device-specific homologation codes from the Regulatory Info Service database.
(5) **Image Validator System Integration:** Encodes retrieved images into Data URI format and submits them to an internal validation system [4] to verify presence of required homologation codes.
(6) **Release Tracker System Integration:** Retrieves software build specifications and commit history to validate intended fixes are embedded in the binary.

4.3 Experiment

The RICH Agent system was evaluated using 110 real-world task instances, where a unique ticket identifier serves as the input to initiate the process. Upon receiving the identifier, the system executes an orchestrated agent workflow to analyze and validate regulatory information requirements. Given the criticality of the domain, the outputs are validated via an Expert Review and classified as either valid or incorrect. These findings are subsequently analyzed using the precision and accuracy metrics detailed in section 4.4.

4.4 Evaluation Metrics

The system outputs will be validated through a formal expert review process, where each response is binarily classified as correct or incorrect based on the expert's technical analysis. To evaluate the system's performance, the overall accuracy will be calculated, representing the proportion of correct predictions relative to the total number of task instances. This metric provides a straightforward measure of the agent's reliability in executing the end-to-end validation workflow against the expert's ground-truth labels.

5 Results

The automated tests were executed on 110 tasks. Of these, 77 (70.0%) were correctly evaluated (**Yes**), 25 (22.7%) were classified as incorrect (**NO**), 4 (7.3%) were *Not run* due to missing binaries information in the ticket.

The majority of 'Incorrect' cases stem from unregistered software-version information (e.g., 'Dev did not register the SW Version during the development process') and reference inconsistencies ('required data not found in the system'). These deficiencies prevent the extraction of required regulatory identifiers, cause significant deviations in quality KPIs, and increase operational cost due to the need for manual intervention.

Not run cases focus on tickets that did not contain the necessary binary information, highlighting the importance of prior validation.

The analysis recommends the implementation of:

- **Ticket pre-analysis** – Implement a tool that can validate whether all relevant information is present in the ticket before the actual analysis begins
- **Automated fallback** – Research whether it's possible to retrieve the missing information from another internal system.
- **Programmatic enrichment of metadata** – Automatic ingestion of Country and model information from context.

6 Conclusion and Future works

The experimental results demonstrate that the RICH agent can accurately validate a significant portion of homologation tasks, achieving a 70.0% success rate. The analysis of the remaining cases revealed that the primary bottlenecks were not failures in the agent's reasoning or tool execution, but rather systemic deficiencies in the input data, such as unregistered software versions and missing metadata in task tickets. This finding highlights an unexpected benefit of the system: it acts as a diagnostic tool for the overall development process, exposing gaps in data integrity that were previously obscured by manual testing. In conclusion, the delegation of compliance tasks to specialized AI agents can enhance operational reliability and reduces human intervention. However, reproducibility is limited by the reliance on proprietary internal systems and locally deployed models.

To build upon these findings, future efforts will focus on mitigating the identified failures to increase the system's overall accuracy. Specifically, we aim to implement a pre-analysis validation layer to ensure ticket completeness before the agent begins the verification process. Additionally, we will explore automated fallback mechanisms to retrieve missing metadata from alternative internal sources, thereby reducing the dependency on manual ticket updates.

A limitation of this study is the absence of comparative time/effort metrics between the manual and automated validation processes. Future work will include time-motion studies to quantify efficiency gains.

Furthermore, we intend to broaden the agent's capabilities to encompass a wider array of requirements for validation, further minimizing manual intervention in the verification pipeline. Finally, we identify the adoption of the Model Context Protocol (MCP) as a promising avenue for enhancing the interoperability between the LLM-based agent and legacy internal systems. Standardizing these interfaces via MCP would reduce the friction of custom tool integration and allow for a more scalable and seamless consumption of organizational data.

Artifact Availability

RICH cannot be shared due to industrial restrictions and confidentiality constraints.

Acknowledgments

This is a result of the Research, Development & Innovation Project (ASTRO) performed at Sidia Institute of Science and Technology sponsored by Samsung Eletrônica da Amazônia Ltda., using resources under terms of Federal Law No.8.387/1991, by having its disclosure and publicity in accordance with art.39 of Decree No.10.521/2020. The author, Eduardo Oliveira, is a former employee and can currently be reached at the email eduardopeluchi@outlook.com.

References

[1] Islem Bouzenia and Michael Pradel. 2025. Understanding software engineering agents: A study of thought-action-result trajectories. *arXiv preprint arXiv:2506.18824* (2025).

[2] Dongming Jin, Zhi Jin, Xiaohong Chen, and Chunhui Wang. 2024. MARE: Multi-Agents Collaboration Framework for Requirements Engineering. CoRR abs/2405.03256 (2024). *arXiv preprint arXiv:2405.03256* 10 (2024).

[3] Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2024. From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479* (2024).

[4] Wendell Marques, Andrezza De Melo Bonfim, Edluce Leitao Veras, Daniel Souza, Vinicius Gabriel da Silva, et al. 2025. Application of Computer Vision to image review process in the software development. In *Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST)*. SBC, 150–152.

[5] Malik Abdul Sami, Muhammad Waseem, Zheying Zhang, Zeeshan Rasheed, Kari Systä, and Pekka Abrahamsson. 2024. AI based multiagent approach for requirements elicitation and analysis. *arXiv preprint arXiv:2409.00038* (2024).

[6] Yongjian Tang and Thomas Runkler. 2026. LLM-Based Agentic Systems for Software Engineering: Challenges and Opportunities. *arXiv preprint arXiv:2601.09822* (2026).