RAVEN: Enhancing Test Request Reliability through Real-Time Verification in High-Complexity Systems

Rayfran Rocha Lima
Sidia Institute of Science and Technology
Manaus, AM, Brazil
rayfran.lima@sidia.com

Rodrigo Carvalho Silva
Sidia Institute of Science and Technology
Manaus, AM, Brazil
rodrigo.carvalho@sidia.com

ABSTRACT

In embedded software projects for mobile devices, test requests are critical to ensuring product quality, yet complex dependencies on binaries, version control, and country-specific configurations often cause submission errors. These errors lead to inefficiencies and delays in testing pipelines. This paper reports on the development and evaluation of RAVEN (Request Assessment and Verification Engine), a real-time verification system built through an Action Research approach. RAVEN integrates data from JIRA, version control systems, and binary baselines to automatically verify over 30 criteria as users fill out request forms. Deployed in an industrial environment, RAVEN improved process efficiency—reducing approval lead time by 58%, halving submission iterations—and enhanced reliability, lowering the rejection rate of a key request type by 17 percentage points. It also improved the clarity of verification feedback and fostered greater requester accountability. A TAM-based questionnaire confirmed high user-perceived usefulness and ease of use. While some verification gaps remain, RAVEN demonstrated the value of proactive, educational verification. Future work includes extending its scope, integrating ML-based recommendations, and replicating the approach in other contexts.

KEYWORDS

Automated Testing, Real-Time Verification, Software process efficiency

1 Introduction

The development and maintenance of embedded software for Android-powered devices—such as smartphones, tablets, and smartwatches — require Original Equipment Manufacturers (OEMs) to comply with a rigorous homologation process defined by Google [1]. This process, known as the Google Approval Process (GAP), is essential for certifying system releases that include proprietary services like Google Mobile Services (GMS) or Firmware Over-The-Air (FOTA) updates. While GMS is a collection of Google applications and APIs that help support functionality across devices, FOTA refers to the technology that allows Android-powered devices to update the operating system and applications of the mobile device remotely without the need for a physical connection to a computer [3]. To meet these requirements, OEMs must conduct extensive testing to ensure compatibility, security, and performance, including validations of customizations applied for specific carriers or regional markets [12].

Each Android release undergoes multiple verification and validation (V&V) scopes — SM (Security Maintenance), NE (Normal Exception), FS (Full Submission), RE (Regular), and SE (Simple Exception) — generating over 500 artifacts that must be reviewed before submission to Google [4] to, after checking all evidence, provide conformity certificate called Google Approval Letter.

Although recent initiatives have reduced GAP approval times (e.g., 22% reduction compared to 2022) and improved artifact review efficiency [3], a critical bottleneck remains at the initial stage: the submission of test requests. Due to the complexity of verification rules and volume of required information and artifacts, traditional manual processes for test request verification are prone to errors and inefficiencies, leading to high rejection rates and subsequent delays in the development lifecycle.

In this context, we introduce RAVEN (Request Assessment and Verification Engine), an automated system designed to support the verification of test requests in high-complexity embedded software environments. RAVEN analyzes structured request data from the GAP request platform against data and artifacts contained in platforms such as JIRA (task management system) and version control systems to verify compliance with predefined verification rules, encompassing scope coverage, artifact completeness, and alignment with build configurations.

This paper presents an Action Research study implemented in a multinational technology company to resolve the high test request rejection rates. The study assesses the impact of RAVEN on GAP lead time and requesters education. The subsequent sections detail the system architecture, implementation challenges, evaluation metrics, and the observed benefits post-deployment.

2 Background and Related Works

The process of verifying test requests in embedded software environments is particularly challenging due to strict compliance requirements imposed by device manufacturers, regional regulations, and carrier-specific configurations. Each complete binary suite is composed of four complementary binaries called BL (bootloader), AP (application processor), CP (communication processor), and CSC (client system configuration) [10]. It must conform to variations based on country, OEM, and telecommunications providers, increasing the complexity of test preparation. Prior studies have explored strategies to improve early test readiness. For example, Andrade et al. [3] proposed an automated approach to optimize the Google Approval Process, significantly reducing release failures and approval time. Fernandes et al. [6] introduced the TSS Tool to automate the setup of test environments in parallel, leading to substantial efficiency gains. Albuquerque et al. [2] applied robotic automation to improve the accuracy and speed of physical sensor testing in mobile devices. These studies demonstrate that automation and tool integration can enhance test reliability and reduce operational overhead. However, these approaches often rely on asynchronous feedback, which delays error detection and correction. They typically act post-submission or during environment

setup, rather than preventing invalid requests at the source. This gap motivates solutions like RAVEN, which aims to shift verification earlier in the workflow by assisting users during the request authoring phase.

In the software testing domain, V&V are critical but distinct activities. While validation assesses whether a system meets business requirements, verification focuses on checking whether artifacts are correctly constructed [15]. In our context, the GAP (Google Approval Process) is responsible for validating the correctness and completeness of binary behavior. Conversely, our approach emphasizes verification—ensuring the integrity, consistency, and traceability of metadata and configuration during test request preparation. Verification mechanisms have been explored to reduce failure rates earlier in the development cycle. Stoico [18] proposed an early system-level verification, focusing on functional and timing flaws. Although effective at the modeling level, such methods are rarely integrated into real-time operational workflows like test request verification pipelines.

Technological advances in RPA, APIs, and AI agents have enabled automated V&V processes across heterogeneous systems. RPA (Robotic Process Automation) has been used to integrate platforms that lack native interoperability, offering ways to extract and analyze user input dynamically [5]. Kumar et al. [11] introduced Saarthi, an autonomous AI verification agent that uses an agentic workflow to perform end-to-end formal verification of RTL designs, illustrating the growing role of generative AI in critical verification pipelines. The MCP (Model-Context-Process) model has been introduced as an architectural solution to connect agents with enterprise systems, aligning domain-specific models with execution contexts [9]. Although the MCP enhances interoperability and scalability for Agentic AI, evidence of its long-term performance is limited [16].

From a process improvement perspective, interventions focused on system bottlenecks have shown to deliver significant performance gains. According to the Theory of Constraints (TOC), optimizing non-restrictive points in a process typically yields marginal benefits, whereas addressing constraints can enhance overall throughput [8]. In software testing workflows, identifying and resolving verification- and validation-related bottlenecks can reduce rework, resource waste, and lead time. Related studies [13] have applied process mining and simulation to locate inefficiencies, though most focus on execution phases rather than preparation stages.

To address these gaps, our study proposes a real-time verification engine capable of being integrated into the proprietary request submission workflow. It combines good practices and concepts described in the literature—such as early verification [18], agent-based automation [11], RPA-enabled integration [5], and bottleneck-oriented improvement [8]—into a single solution. Its contribution lies in proactively guiding users during the request authoring phase, offering educational feedback, and reducing the likelihood of request rejection.

To the best of our knowledge, no previous study has addressed proactive real-time verification at the test request authoring phase in embedded software testing pipelines. Existing approaches typically operate post-submission or rely on asynchronous verification cycles, leaving a gap that RAVEN aims to fill.

3 Research Design

This study applied the *Action Research* (AR) methodology, which consists of five iterative phases: **diagnosis**, **planning**, **intervention**, **evaluation**, and **reflection** [14]. This approach enabled close collaboration between researchers and practitioners to investigate a real-world problem, develop a targeted solution, and assess its impact in an industrial setting.

Diagnosis. The investigation began by identifying recurring issues in the test request process for Android-based software releases. These requests are governed by complex, frequently changing business rules, including regional and OEM-specific constraints, which made the preparation and verification process error-prone and inefficient.

Planning. Based on a root cause analysis and stakeholder input, the team designed an artifact aimed at proactively verificating test requests. The RAVEN was conceived to address the root causes of rejection by providing real-time feedback to the requester during form completion.

Intervention. RAVEN was implemented and deployed within the organization's proprietary test request platform, known as KRAKEN. Due to security restrictions, RAVEN uses client-side scraping to read data during the request editing phase. It verifies over 30 rules derived from three primary sources: the JIRA MCP Server, the internal version control system, and a set of baseline binary configuration rules.

Evaluation. To assess the effectiveness of RAVEN, we employed both quantitative and qualitative methods. Quantitative data included rejection rate reduction, approval lead time, and the number of submission iterations. Using a TAM-based questionnaire for the test requesters helped to capture user feedback on perceived usefulness, ease of use, and intention to adopt the tool. The outcomes of this evaluation are detailed in Section 4.

Reflection. Based on the evaluation results and user feedback, the team identified opportunities to improve training, documentation, and verification rules. These insights supported the refinement of RAVEN and informed its institutionalization as a core part of the verification workflow.

4 Results

The results presented in this section were derived from a combination of objective operational data and user feedback. Quantitative metrics, including request volume, rejection rates, and process efficiency Metrics, were extracted from system logs covering 9 months before and 3 months after RAVEN deployment, when RAVEN was actively used. Qualitative insights were collected through a TAM-based questionnaire administered to project leaders involved in the test request process. Together, these data sources provide a comprehensive view of RAVEN's impact on the verification process and on user experience based on the TAM framework.

4.1 Diagnosing the Process Bottleneck

As a diagnosis phase outcome, the Figure 1 illustrates a simplified view of the lifecycle of an Android-based software release that must comply with Google requirements. The process begins with the identification and implementation of new requirements into the binary (Step 1 – *Adapt / Extern Android*). Once requirements are applied, the binary undergoes preliminary internal verifications

before a formal test request can be submitted (Step 2 – *Request for Internal GA Tests*).

Each binary generated may include configurations and constraints that vary by country, carrier, or OEM. This introduces a high degree of complexity in the preparation and verification of test requests. Due to a history of critical failures at this stage, the GAP Testing department previously introduced an asynchronous verification system (AVS). This system runs every 15 minutes and verifies over 30 *Check Points* (CPs) before allowing test execution. Its main limitation is the lack of real-time feedback, the verifications are performed only after the request has been submitted.

The CPs verify key aspects of the request, including the presence and consistency of critical metadata, compliance with configuration rules, and binary compliance. Additionally, they check the integrity of binary data based on locally defined Google rules, such as whether all commits are in line with the tools, whether all build commands are in the correct order, whether there is any non-compliance in the data, and whether all information provided in the request matches the binary data.

If the request passes all verifications, it proceeds to GAP (Step 3), which initiates the automated testing workflow: the binary is automatically downloaded, the test sample device is flashed, and an automated test suite—tailored to the release type—is executed. These tests typically take around 12 hours. On the following day, testers can begin the manual testing phase based on the outcomes of the automated checks. Once all tests are completed and the Google Approval Letter is received, the binary advances to Quality Assurance procedures and, ultimately, to FOTA deployment.

If the AVS identifies any CP failures, it automatically rejects the request and provides a diagnostic log detailing the issues found. To ensure traceability, rejected requests cannot be modified, requiring the requester to restart the submission process from Step 2. This limitation increases FOTA lead time, promotes unnecessary rework across the entire test request process, and delays the reception of the Google Approval Letter.

While the introduction of the AVS significantly reduced GAP operational waste (only about 3% of tests are cancelled due to late discovery of faults in requests), it introduced a new process bottleneck: feedback is delivered only after a formal test request submission. This results in delays, rework cycles, and frustration among requesters, who may repeat the same mistakes across every new submission.

4.2 Rejection Rate Analysis

An analysis of nine months of test request data revealed high variability in weekly rejection rates, despite periodic training and process clarifications. As shown in Figure 2, although a downward trend is observable, the rejection pattern remains inconsistent.

To understand the underlying issues, we conducted a root cause analysis of 40 randomly selected rejected requests. This analysis was carried out during a structured focus group session that included three focal points—professionals with deep expertise in test request verification rules—and two senior project leaders responsible for coordinating binary submissions. The session was designed to elicit not only technical interpretations of the rejection causes but also contextual insights related to formal test requesters' behavior, ambiguity in requirements, and rule misalignment. Participants collaboratively classified 50% of the rejections as preventable, identified verification rule weaknesses, and shared suggestions that directly influenced the design scope of the RAVEN engine. The most frequent issues identified were incorrect request completion and binary-related problems, as summarized in Figure 3.

Further analysis categorized the rejections by release type. As illustrated in Figure 4, SM and RE release types accounted for 80% of total rejections, with SM alone representing 52%. According to the Pareto principle [17], this confirmed that a small subset of release types was responsible for the majority of issues.

4.3 System Overview: The RAVEN Architecture

RAVEN is a real-time verification layer integrated into the test request submission workflow of a proprietary system, called KRAKEN, used to open requests for internal GA Tests. The complete architecture and interaction flow of RAVEN is illustrated in Figure 5.

Due to strict security controls, the KRAKEN does not allow external agents to pre-fill or submit forms on behalf of users. However, it does permit passive reading of form data during the request composition phase via client-side JavaScript scraping. RAVEN leverages this capability to extract and analyze data from the form before submission.

Once a requester begins filling out the test request, RAVEN activates a verification pipeline composed of more than 30 verification rules, distributed across three key verification sources:

1. JIRA Model Context Protocol (MCP) Server: RAVEN uses the MCP JIRA Server to verify the existence and correctness of binary identifiers, verify associated requirements, confirm the version control branch, and ensure that testable issues are properly linked and tracked.

2. Version Control System (VCS) API: Through direct integration with the organization's version control platform, RAVEN cross-checks the request against the actual structure and content of the registered binaries, identifying mismatches in configuration files, missing components, or outdated versions.

3. Binary Base and Geo-Specific Rules: The current request is compared against a reference binary ("base version") to detect inconsistencies. Additionally, RAVEN evaluates compliance with country- and carrier-specific constraints—such as required carrier order in configuration files or the inclusion of grouped OEM families—as mandated by business rules.

All verification results are compiled into a visual report displayed in real time on the requester's interface. The report lists all checks, indicating whether each item was successfully verified or requires correction. Crucially, the system blocks the submission of the test request until all issues have been addressed.

This early-stage intervention transforms RAVEN from a passive gatekeeper into an active learning tool. Rather than merely rejecting flawed requests, RAVEN educates the requester by clearly highlighting the cause of the error and providing actionable guidance, thereby reducing friction in the workflow and fostering process maturity.

Real-Time Feedback Example To better illustrate how RAVEN guides users during the test request composition phase, Figure 6 presents a simulated example of the pop-up interface shown to requesters. The system immediately notifies users of specific issues identified during verification, such as mismatches in CSC versioning and

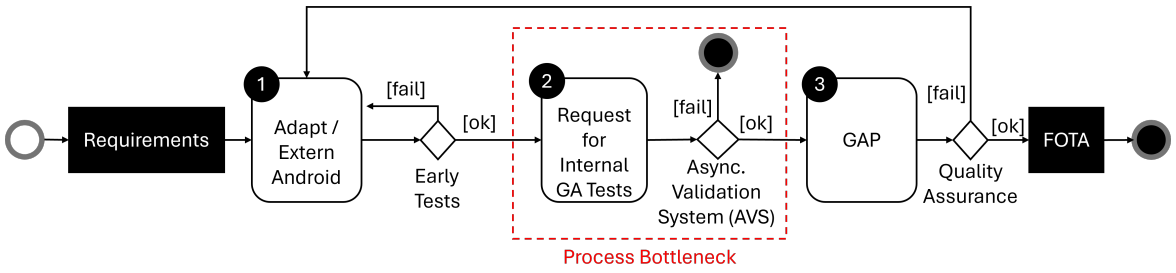

Figure 1: Process Bottleneck

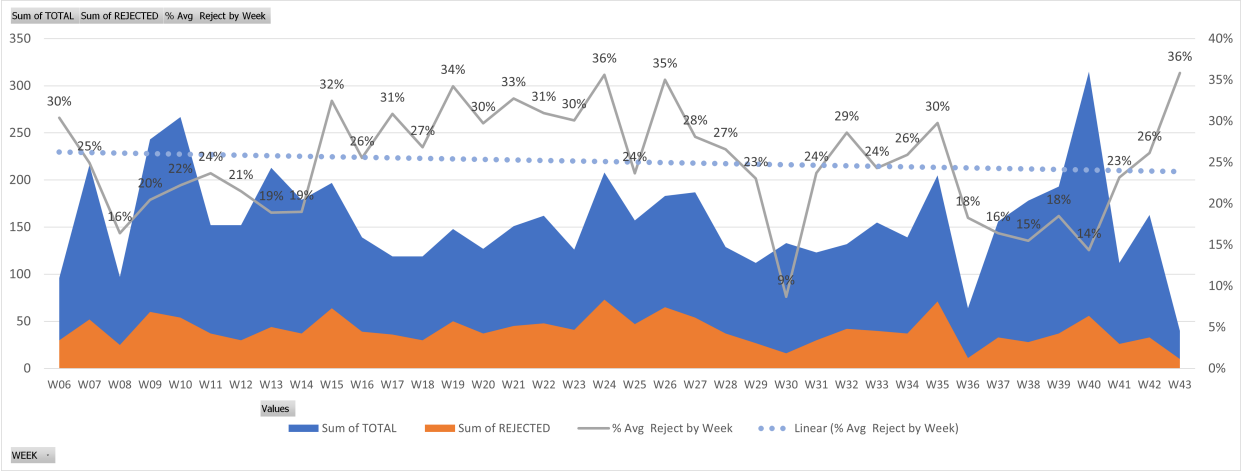

Figure 2: Distribution of test request rejections over 9 months.

#	Reason for Rejection	Qty	%	Preventable?
1	Binary expired	12	30%	NO
2	<field name #15> need to be based on...	10	25%	NO
3	Incorrect fingerprint	10	25%	YES
4	Incorrect base submission ID	3	8%	NO
5	Use of group mode with only one buyer	2	5%	YES
6	<field name #32> filled in incorrectly	1	3%	YES
7	Incorrect All Binary	1	3%	NO
8	Wrong release type applied	1	3%	YES
	Totals	40	100%	

Figure 3: Classification of preventable rejection causes.

missing fingerprint identifiers. This real-time feedback prevents invalid submissions and reinforces a learning-oriented process.

4.4 Impact generated by RAVEN

The implementation of the RAVEN engine in the test request workflow produced measurable improvements in both the efficiency and reliability of request submissions. To evaluate the impact of the RAVEN engine, we compared test request Metrics over two distinct periods: 9 months before RAVEN deployment (referred to as *Before*) and 3 months after deployment and active usage (*After*). Although the periods differ in duration, the rejection rates and type distributions were normalized as percentages to ensure comparability.

Month	% FS	% SM	NE	RE	SE	Total
02	1%	39%	9%	19%	32%	100%
03	0%	55%	11%	26%	8%	100%
04	0%	60%	12%	20%	8%	100%
05	0%	63%	5%	28%	4%	100%
06	0%	46%	7%	44%	3%	100%
07	1%	55%	12%	26%	6%	100%
08	0%	38%	23%	33%	6%	100%
09	2%	56%	4%	30%	8%	100%
10	6%	55%	8%	25%	6%	100%
AVG	1%	52%	10%	28%	9%	100%

Figure 4: Distribution of rejections by type (Before)

Table 1 summarizes the overall request volume and rejection rate for both periods.

Table 1: Comparison of Test Request Volume and Rejection Rate Before and After RAVEN Deployment

Metric	Before	After
Total tickets submitted	5,481	1,376
General rejection rate	33.3%	31.7%

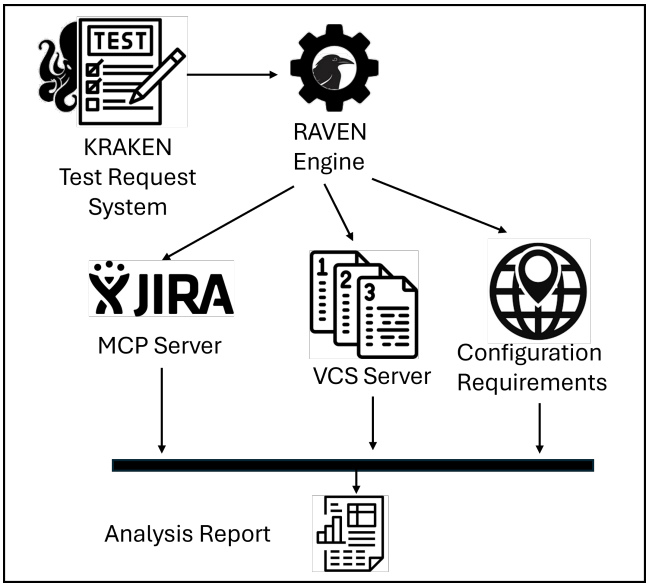

Figure 5: Overview of the RAVEN architecture and its integration with internal verification systems.

RAVEN – Real-Time Verification Feedback

5 issues were found. Resolve them and retry.

(1) **[CSC Version]** CSC version mismatch with specification registered in Jira (JIRA-345).
(2) **[Fingerprint [a]]** Binary version "A0XXMUSBADIII" is not a base binary.
(3) **[Binary]** CSC build ID does not match.
(4) **[Binary]** CSC Name not found in binaries.
(5) **[Customer]** Carrier AAA configuration out of expected order.

OK

[a]The "fingerprint" is a unique string that identifies each binary on Google's servers, containing information such as Android version, manufacturer, binary version, and build ID.

Figure 6: Simulated example of RAVEN's pop-up interface during request composition.

We also analyzed the distribution of request types submitted and rejected in both periods. Table 2 presents the distribution of all submitted tickets. Figure 7 shows the distribution of rejected tickets by type for both periods.

To statistically validate the differences observed between the *Before* and *After* periods, we used the Jamovi software[1] to apply the non-parametric Mann–Whitney U test [7] to the rejection proportion distributions for each ticket type (Figure 7). This test was

[1]https://www.jamovi.org

Table 2: Distribution of All Submitted Tickets by Type

Ticket Type	Before	After
SM	42.3%	48.2%
NE	29.0%	25.0%
RE	20.0%	22.0%
Others	8.7%	4.8%

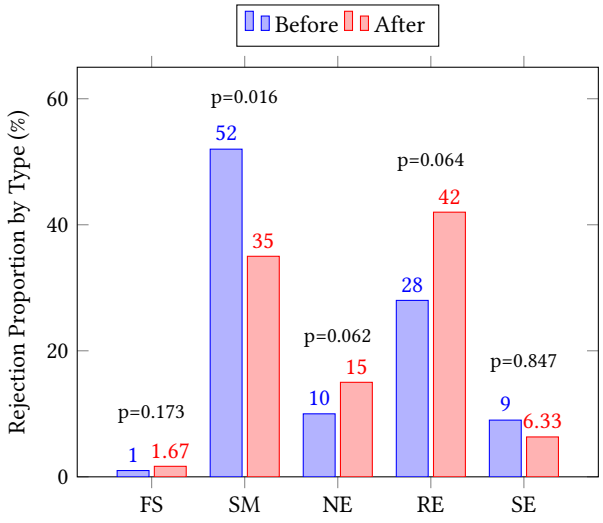

Figure 7: Change in Rejection Distribution by Type

chosen because it does not assume a normal distribution and is suitable for small sample sizes.

The analysis confirmed that the reduction in SM-type rejections—from 52% to 35%—was statistically significant ($p = 0.016$), reinforcing the conclusion that RAVEN had a meaningful impact on this category. For NE (from 10% to 15%, $p = 0.062$) and RE (from 28% to 42%, $p = 0.064$), the results suggested possible changes, though they did not reach the conventional 5% significance threshold. FS (from 1.00% to 1.67%, $p = 0.173$) and SE (from 9.00% to 6.33%, $p = 0.847$) showed no statistical difference. By distinguishing between changes likely attributable to the intervention and those that could be due to random variation, the statistical analysis strengthened the reliability of our findings.

When analyzing both submission and rejection proportions, SM-type requests stand out as the most notable improvement: despite their share among submitted tickets increasing from 42.3% to 48.2%, their rejection proportion dropped from 52% to 35%. NE-type requests moved in the opposite direction: while their share of submissions decreased from 29.0% to 25.0%, their rejection proportion increased from 10% to 15%. RE-type requests exhibited a similar pattern, with submissions growing from 20.0% to 22.0% and rejections rising from 28% to 42%. FS and SE types maintained relatively low proportions in both submissions and rejections, with only minimal variations.

Before the intervention, the organization faced recurrent problems such as incomplete information, inconsistencies between declared and actual binaries, and violations of country- or carrier-specific rules. With the deployment of RAVEN and the implementation of real-time feedback, the proportion of SM-type rejections was substantially reduced, indicating a measurable improvement in submission quality for the most error-prone category. This improvement was achieved without modifications to the test execution infrastructure, relying solely on the introduction of a verification layer at the submission stage. Given the observed patterns, RE-type requests—which showed the largest increase in rejection proportion despite a slight rise in submissions—represent the next priority for refinement of verification rules and targeted process improvements.

Other Metrics were collected to compare GAP process efficiency before (9 months) and after the intervention (3 months). Table 3 summarizes the results:

Table 3: Other Metrics Before and After RAVEN Deployment

Metric	Before	After
Average approval lead time (days)	24	10
Average number of iterations per request	2.5	1.2
Requester's perceived clarity of errors (Likert scale)	1.0	4.5
Average time spent on verification and fixing issues	2h	10 min.

Other key metrics also demonstrated improvements. The average approval lead time (number of days between creating the test request and receiving the Google approval letter) reduced from 24 days to just 10 days, and the average number of iterations per request (number of times that the test requester contacts the testing team to clarify or resolve a request failure) decreased from 2.5 to 1.2. Although much of the verification process has been automated, offering real-time feedback to the user, there are still other items to be verified or prepared manually. Therefore, it was not possible to reset the time spent on verification and issue resolution, but a reduction of 2 hours to just 10 minutes was achieved in this process.

Furthermore, the perceived clarity of errors by requesters significantly increased, rising from 1.0 to 4.5 on a subjective scale. Prior to the implementation of RAVEN, the PLs, who create test requests, relied on a comprehensive verification manual for test requests, which outlined over 30 evaluation items accompanied by detailed how-to instructions. Despite the existence of this document, the average opinion among PLs was that it did not sufficiently clarify all verification points. Some PLs noted challenges such as "The manual is too extensive and complex" (P1), "In some sections, it is unclear how to verify the analyzed information" (P2), and "Although the manual indicates where to obtain the information, I do not have access to certain systems" (P3). These issues contributed to a lack of transparency and efficiency in the verification process, often leading to confusion and delays.

These results highlight that the automation of request verification not only enhanced process efficiency but also significantly reduced resource waste and the time required for product delivery. The implementation of RAVEN ensured greater accuracy in submissions, minimizing rework and guaranteeing that tests were conducted only on the correct binaries with appropriate requirements, contributing to a faster and more reliable software product delivery.

4.5 RAVEN's User Feedback

To assess user acceptance of RAVEN, we administered a Technology Acceptance Model (TAM)-based questionnaire to all 49 project leaders, with a 90% response rate. Those who didn't respond were on vacation or medical leave. Although the seniority level distribution cannot be disclosed due to confidentiality agreements and organizational policies, all respondents possessed at least 2 years of experience in creating these test requests supported by human verification performed manually, depending on feedback from AVS. Their feedback was grouped under three dimensions:

- **Perceived Usefulness:** Most respondents (85%) reported that RAVEN automated verification tasks, reduced manual effort, improved accuracy, and increased the overall quality of test requests. The majority of respondents (85%) agree that RAVEN is used every time a new binary release needs testing.
- **Perceived Ease of Use:** Most respondents (95%) found RAVEN intuitive and easy to install, though some noted limitations in the clarity of system feedback during verification.
- **Intention to Use:** All respondents (100%) indicated a high level of agreement in continuing to use the tool to support the process of verifying test requests.

We also identified a certain degree of tool dependence, with 88% of users stating that its discontinuation or unavailability would significantly impact the verification of test requests. Regarding the timing of RAVEN's availability, all respondents affirmed that the tool was deployed at an appropriate and needed moment.

5 Discussion

RAVEN was implemented with minimal intrusion into existing platforms, leveraging scraping and MCP-based integration—an architectural choice that makes it lightweight and adaptable even in highly controlled industrial environments.

Its introduction in the test request workflow led to notable improvements in both operational efficiency and the quality of request submissions. Beyond measurable process gains, RAVEN contributed to enhancing first-time correctness and fostering a more proactive verification mindset among requesters. By embedding verification awareness directly into the authoring phase, the tool reduced reliance on asynchronous feedback and helped users identify and resolve issues earlier in the process.

One of the most relevant impacts was observed in SM-type requests, previously the most error-prone category. The tool proved particularly effective in supporting requesters in this frequent and critical segment. Conversely, the observed increase in RE-type rejections highlighted the need to refine verification rules and broaden coverage to better handle this category moving forward.

While the improvements achieved were significant, they fell short of the initial goal of a 30% reduction in rejection rates. Two key factors contributed to this outcome: (1) the emergence of new rules applied by the AVS that were not yet mapped by RAVEN,

and (2) partial adoption of the tool in the first month after official deployment of RAVEN (30% of project leaders were not yet using RAVEN). Addressing these gaps is a priority for further improving process quality and extending RAVEN's impact.

Architectural limitations also remain. RAVEN relies on a scraping mechanism implemented via a Chrome plugin, making it vulnerable to changes in the user interface of the proprietary request platform. In addition, because it operates as a client-side extension, performance and reliability depend on each user's browser environment, complicating support and standardization. Another critical limitation is the lack of integration between RAVEN and the AVS. Without a shared verification API, RAVEN must replicate and manually update its rule set, creating a maintenance bottleneck and limiting consistency across verification mechanisms.

To address these limitations, future improvements should prioritize the centralization of verification logic through reusable APIs, enabling synchronous and asynchronous verificators to remain consistent, modular, and easier to maintain. Additionally, exploring more resilient integration mechanisms — such as an official AVS or KRAKEN API or headless browser automation — may improve the robustness and scalability of the solution. Beyond these enhancements, future extensions could leverage secure A2A architectures and advanced agentic models (e.g., MAESTRO-based threat modeling) to further strengthen verification capabilities and enable more sophisticated automation.

User feedback confirms that RAVEN is perceived as both useful and easy to use. The significant improvement in perceived clarity of verification feedback highlights the tool's role in reducing the cognitive effort required to navigate complex verification rules. By shifting this cognitive load from manual interpretation of extensive documentation to automated, actionable guidance during request composition, RAVEN fosters greater process transparency, user autonomy, and long-term adoption.

5.1 Limitations and Threats to Validity

Following the guidelines of Wohlin et al. [19], we discuss potential threats to the validity of this study, organized into four categories.

Internal validity. The observed improvements after the deployment of RAVEN may have been partially influenced by factors unrelated to the tool itself. For example, the AVS system introduced new verification rules during the evaluation period, which were not yet mapped in RAVEN. Additionally, tool adoption was incomplete in the first month after release, with approximately 30% of project leaders not using it. These aspects could have affected the measured impact.

External validity. The study was conducted in a single organizational context—a highly controlled industrial software environment with a proprietary test request platform (KRAKEN) and specific approval workflows. The degree to which the results can be generalized to other organizations, domains, or processes that differ in governance models, technology stacks, or release management practices remains uncertain.

Construct validity. The metrics used, such as average approval lead time, number of iterations per request, and perceived clarity of errors, rely on accurate logging and self-reported perceptions. While these measures are aligned with the study objectives, they may not fully capture all relevant aspects of submission quality or verification efficiency. The absence of a unified verification API also means that RAVEN's rule set had to be replicated and maintained manually, potentially introducing discrepancies between intended and actual checks.

Conclusion validity. Although statistical analyses were applied to compare pre- and post-intervention data, the relatively short post-intervention period (three months) and the lack of a randomized control group limit the strength of causal inferences. Variability in user behavior, differences in project complexity, and dependency on client-side scraping—which is sensitive to UI changes and browser environments—may also influence the consistency of results over time.

6 Conclusion

This study presented the development, implementation, and evaluation of RAVEN, a real-time verification system designed to enhance the reliability and efficiency of test requests in high-complexity embedded software environments. By integrating data from multiple internal systems (JIRA, version control repositories, and reference binaries) and applying more than 30 verification rules, RAVEN was able to prevent the submission of invalid requests already during the composition phase.

The results indicate measurable gains: a significant reduction in the proportion of SM-type rejections (from 52% to 35%), a decrease in the average number of iterations per request (from 2.5 to 1.2), and in the average approval lead time (from 24 to 10 days), in addition to a substantial increase in users' perceived clarity of errors. However, the overall reduction in the rejection rate was modest and not statistically significant, reinforcing that the improvements were more concentrated in specific types.

Although the initial goal of reducing the overall rejection rate by approximately 30% was not achieved, two main factors limited the results: (i) the emergence of new verification rules in the AVS that were not yet mapped in RAVEN, and (ii) partial adoption of the tool during the first month after deployment, when approximately one-third of project leaders were not yet using it. Addressing these gaps will be a priority in the next iteration of the solution.

Beyond operational gains, RAVEN fostered a cultural shift by promoting user accountability and providing educational, real-time feedback. Its lightweight integration strategy—enabled through client-side scraping—proved effective even in systems with strict access control, making it a viable option for similar organizational contexts.

As future work, we plan to:

- Engage with the team responsible for the asynchronous verification system to enable, via API, the reuse of part or all of its verification logic within RAVEN;
- Expand the scope of RAVEN's checks to include additional release types, such as RE and NE;
- Incorporate machine learning techniques to suggest fixes or dynamically autofill corrections based on historical patterns;
- Conduct replication studies in other departments or processes to assess the broader impact and applicability of the verification approach.

We expect that this work can inspire similar initiatives in organizations that deal with high-volume, high-complexity testing

processes, where ensuring data quality at the source is crucial to maintaining operational efficiency.

ACKNOWLEDGMENTS

This paper is a result of the Research, Development & Innovation Project named COPILOTO 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.

REFERENCES

[1] 2023. Android Open Source Project. https://source.android.com/ Accessed: 2023/04/13.

[2] Abda Albuquerque, Alice Castro, and Heryck Barbosa. 2024. Reports on using an ABB robot in Android mobile testing automation. In *Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST)*. SBC, 74–76.

[3] Ewerton Andrade, Janislley Sousa, Matheus Lopes, Davi Barbosa, Weslen Lima, and Jimmy Lacerda. 2024. Experiences with Google Approval Process: an automated approach to enhancing efficiency in Android releases. In *Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST)*. SBC, 83–85.

[4] Heryck Michael Dos Santos Barbosa, Pedro Ivo Pereira Lancellotta, Joao Gabriel C Santos, Abda Myrria De Albuquerque, and Janislley O De Sousa. 2023. PRIMA: an Automated Tool for Android Releases Homologation Review. In *Congresso Brasileiro de Software: Teoria e Prática (CBSoft)*. SBC, 1–4.

[5] Nourhan Elsayed, Luka Abb, Heike Sander, and Jana-Rebecca Rehse. 2023. Automating computer software validation in regulated industries with robotic process automation. In *International Conference on Business Process Management*. Springer, 135–148.

[6] Carol Fernandes, Charles Araujo, Adriano Oliveira, Wellington Correa, Luan Ipê, and Paulo Andrade. 2023. TSS Tool: Automation Tool Applied in the True Single SKU Setup Environment Preparation for Multiple Devices in Parallel. In *Proceedings of SAST 2023*. https://sol.sbc.org.br/index.php/sast/article/view/30212

[7] Andy Field. 2024. *Discovering statistics using IBM SPSS statistics*. Sage publications limited.

[8] E. M. Goldratt and J. Cox. 2004. *The Goal: A Process of Ongoing Improvement* (3 ed.). North River Press. https://www.amazon.com/Goal-Process-Ongoing-Improvement/dp/0884271951

[9] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278* (2025).

[10] Bilal Ahmed Kamal, Abdul Basit Shahid, Syed Muhammad Sajjad, Kashif Kifayat, and Khwaja Mansoor Ul Hassan. 2024. A Novel Technique of Android Stealth Rooting. In *2024 17th International Conference on Development in eSystem Engineering (DeSE)*. IEEE, 275–280.

[11] Aman Kumar, Deepak Narayan Gadde, Keerthan Kopparam Radhakrishna, and Djones Lettnin. 2025. Saarthi: The First AI Formal Verification Engineer. *arXiv preprint arXiv:2502.16662* (2025).

[12] Pedro Ivo Pereira Lancellotta, Heryck Michael Dos Santos Barbosa, João Gabriel Castro Santos, Klirssia Matos Isaac Sahdo, and Janislley O De Sousa. 2022. An industry case study: Methodology application to the reviewing process on android releases homologation. In *Congresso Brasileiro de Software: Teoria e Prática (CBSoft)*. SBC, 13–16.

[13] Victoria J Mabin and Robert Y Cavana. 2024. A framework for using Theory of Constraints thinking processes and tools to complement qualitative system dynamics modelling. *System Dynamics Review* 40, 4 (2024), e1768.

[14] Kai Petersen and Claes Wohlin. 2009. Context in industrial software engineering research. *International Symposium on Empirical Software Engineering and Measurement* (2009), 401–404.

[15] R. S. Pressman. 2014. *Software Engineering: A Practitioner's Approach*. McGraw-Hill. https://www.mheducation.com/highered/product/Software-Engineering-A-Practitioners-Approach-Pressman.html

[16] Partha Pratim Ray. 2025. A survey on model context protocol: Architecture, state-of-the-art, challenges and future directions. *Authorea Preprints* (2025).

[17] Robert Sanders. 1987. The Pareto principle: its use and abuse. *Journal of Services Marketing* 1, 2 (1987), 37–40.

[18] Vincenzo Stoico. 2021. A model-driven approach for early verification and validation of embedded systems. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 684–688.

[19] Claes Wohlin, Per Runeson, Martin Höst, Magnus Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Science & Business Media.