CI-First QA for Modular Parcel-Sorting Robots: An Industrial Case Study

Lars Talian Stangebye-Hansen, Emil Skogheim, Jacob Forsdahl Iqbal
Department of Computer Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway
Industry partner: OmniMod AS

***Abstract*—This paper summarizes a bachelor-thesis project on automated quality assurance (QA) for modular parcel-sorting robots at OmniMod AS. The main challenge was to combine high-throughput simulation checks with physically grounded validation while remaining compatible with continuous integration (CI). We delivered a three-layer pipeline: a microservice fault-injection layer (specified for follow-on implementation), a simulator validation layer, and an unattended physical-rig validation layer. The deployed workflow ran end-to-end in GitHub Actions, produced deterministic pass/fail gates, and attached compact traces for failure triage. A known control fault was reproduced during physical validation, and internal post-release feedback reported higher confidence and lower manual verification effort (means 8.0–8.5/10).**

***Index Terms*—robotics QA, CI/CD, hardware-in-the-loop, simulation validation, warehouse automation**

I. Problem Setting

Warehouse robotics software has high late-integration risk: behavior can pass unit tests yet fail under coupled physical constraints. For this setting, the goal was an automated pre-release QA workflow that runs unattended, fits existing CLI/CI practices, produces deterministic pass/fail decisions with debugging evidence, and supports simulator-to-hardware fidelity analysis. This objective aligns with established CI quality-engineering principles and design-science framing for artifact-driven problem solving [2], [3].

II. Methodology Snapshot

The project used a five-phase iterative process: exploratory study, QA architecture design, requirement prioritization, MVP implementation, and full integration. The evidence base combined five semi-structured interviews (CTO + four software engineers, February 2025), a two-hour non-participant observation during a client-site upgrade, and four follow-up interviews focused on automation feasibility.

III. Architecture and Workflow

The solution used a three-layer QA architecture: a microservice fault-injection layer specified for follow-on implementation, a simulator layer for fast regression checks, and an unattended hardware-in-the-loop layer on a 2×2 meter rig. The simulator and hardware layers shared scenario semantics and ran through GitHub Actions, which made simulated and physical evidence directly comparable within one release-gating workflow.

Execution followed a fixed sequence: select target mode and scenarios, provision service stacks, run bounded-time scenarios, compute assertions from traces, and publish verdicts with compact artifacts for triage. Ordering was cost-aware: fault-injection checks were designed to execute in seconds, simulator checks in $\sim$30 s–20 min, and physical checks in $\sim$1 min. This staging ensured that slower hardware validation was triggered only after earlier gates passed.

TABLE I
Internal post-release feedback (Likert 1–10).

Statement	Mean	SD
Reduces manual test effort	8.3	0.9
Increases confidence in merges	8.0	1.1
CLI usability	8.5	0.8
Fault-injection usefulness	7.9	0.9
Would recommend to peers	8.4	0.6
Overall satisfaction	8.1	0.7

IV. Validation Criteria

Physical validation was encoded as explicit, automatable checks computed directly from run logs. The criteria required six closed-loop laps (4 m $\times$ 1 m each) within a 90-second lap threshold, a controlled orientation change of $+90^\circ \pm 5^\circ$ at checkpoint 1 with reset at checkpoint 2, obstacle handling with parcel-obstacle IoU below 0.01 across the run, and continued pass behavior with eight randomly disabled OmniUnits. Together, these checks yielded deterministic pass/fail gates rather than manual interpretation.

V. Results

The key technical result was deterministic reproduction of a known arithmetic defect in the control service during physical validation, with trace evidence linked to the CI run. This provided stronger release-gating evidence than simulation-only checks.

Operationally, the pipeline was adopted as a practical pre-release gate for integration-heavy changes. Internal feedback (Table I) indicates strong gains in usability and reduced manual effort.

VI. Operational Findings

In stakeholder evaluation, the delivered system met the intended operational goals: unattended execution, compatibility with existing CLI and CI workflows, deterministic gating with artifact-backed verdicts, and direct comparability between

simulator and hardware runs through shared scenario semantics. The fault-injection layer was specified in architecture and interfaces, but was not productized within the thesis implementation window.

Exploratory work before implementation consistently showed the same baseline failure pattern: excessive manual effort in setup and verification, weak transferability between simulation and physical fault scenarios, and late defect discovery concentrated in on-site testing. Typical manual checkpoints included deliberate microservice shutdown, stray-parcel startup validation, parcel misplacement/removal timing tests, and disembark-accuracy checks. These findings directly motivated the final CI-first architecture.

The most important next step is production implementation of the fault-injection layer, followed by calibrated simulator-vs-rig agreement analysis on a larger scenario set. Practical follow-on work includes repeatable service-boundary fault campaigns, broader physical scenario coverage beyond the 2×2 constraints, and trend tracking for pass-rate drift and recovery time.

VII. Relation to Prior Work

The architecture is consistent with prior work on microservice robustness and fault-injection strategy. Message-level fault classes (loss, duplication, latency, malformed payloads, ordering issues) are well-established as practical probes for distributed-service resilience [4], [5], and they directly informed the specified fault-injection layer in this project.

For simulator-to-hardware evaluation strategy, this work prioritizes deterministic, CI-friendly gating over broad stochastic exploration. This differs from domain-randomization-heavy approaches often used in robotics transfer pipelines [6], but was a deliberate choice given deployment constraints, debugging requirements, and the need for reproducible release decisions.

VIII. Limitations and Scope

This case study is validation-centric, not a benchmark submission. Two constraints dominate interpretation: (1) the 2×2 rig limits external validity to larger floor plans, and (2) the fault-injection layer was specified but not completed as a production component in the thesis timeline. Sensitive implementation details are intentionally omitted to respect partner IP.

IX. Reproducibility Notes

The handover package included CLI extensions, CI workflows, scenario harnesses, and metrics post-processing scripts. These components were sufficient for repeat runs under stable conditions and practical transfer to subsequent contributors.

X. Conclusion

The project delivered a deployable CI-first QA workflow linking simulation and physical validation with deterministic release-gating evidence. In deployment, this improved pre-release fault visibility, reduced manual verification effort, and increased merge confidence. The highest-leverage next steps are production completion of the fault-injection layer and broader physical scenario coverage.

References

[1] L. T. Stangebye-Hansen, E. Skogheim, and J. F. Iqbal, "Design and implementation of an automated testing pipeline: Bridging physical testing and simulation for OmniMod," B.Sc. thesis, NTNU, 2025.

[2] E. M. Maximilien and L. Williams, "Assessing test-driven development at IBM," in *Proc. 25th Int. Conf. on Software Engineering (ICSE)*, 2003, pp. 564–569.

[3] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[5] D. Savchenko, G. Radchenko, and O. Taipale, "Microservices validation: Mjolnirr platform case study," in *Proc. 38th Int. Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015.

[6] F. Muratore *et al.*, "Domain randomization for simulation-to-real transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 4218–4235, 2022.