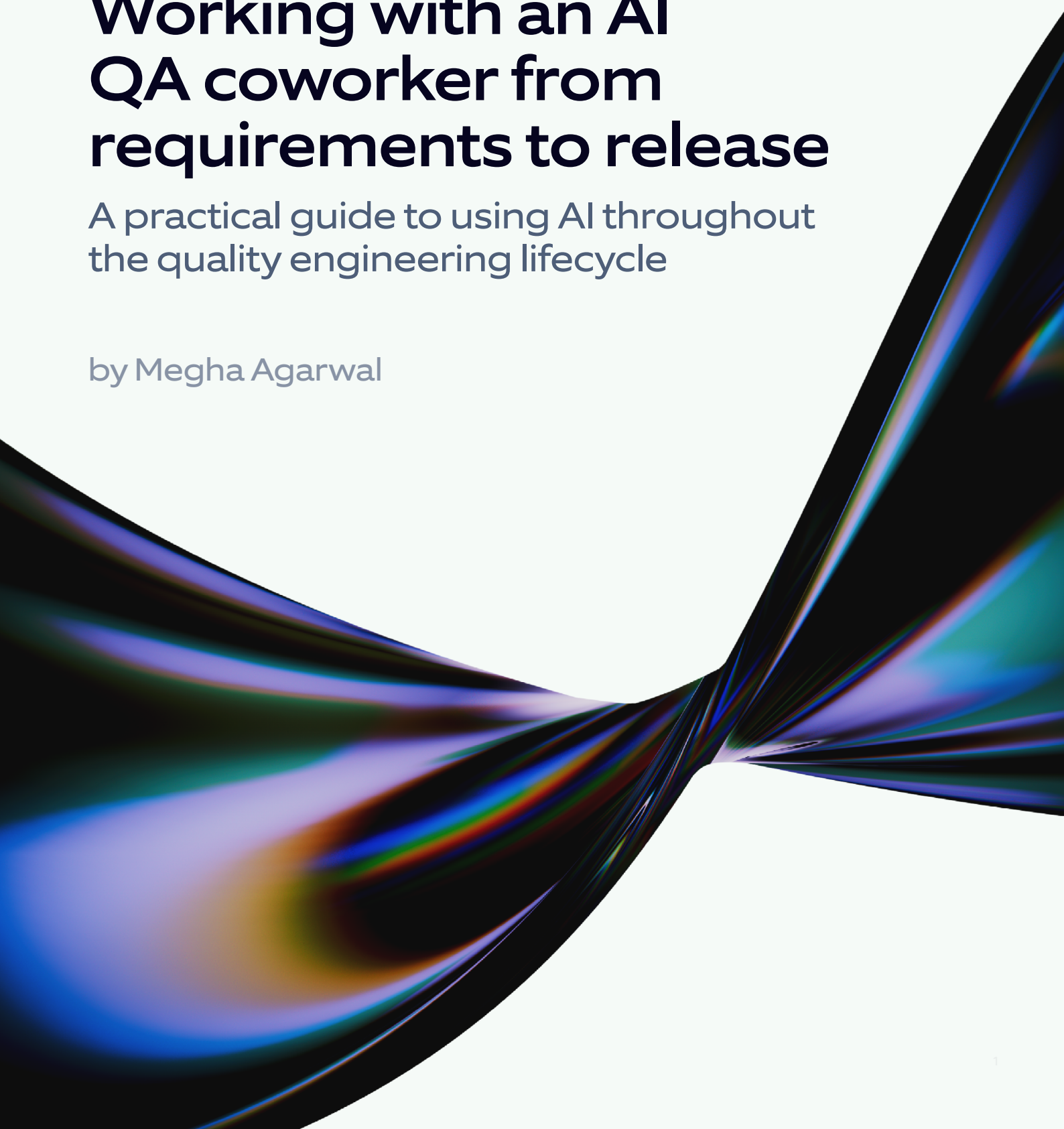
nagarro

Beyond test cases: Working with an AI QA coworker from requirements to release

A practical guide to using AI throughout the quality engineering lifecycle

by Megha Agarwal



nagarro

Table of contents

© 2026 Nagarro

Executive summary

Quality engineering has evolved beyond validating completed software. Modern QA engineers contribute throughout the Software Development Life Cycle (SDLC), from refining requirements and understanding implementation changes to planning verification, maintaining automation, investigating failures, and supporting release decisions.

Even though AI is increasingly penetrating engineering workflows, yet chunk of that work focuses on generating code, creating test cases or producing automation. While these capabilities improve productivity, they represent only a small part of a Quality Engineer's responsibilities. And the scope for leveraging AI in quality engineering goes way beyond.

This paper proposes maintaining **one continuous AI conversation throughout the lifecycle of a feature**, allowing recommendations to evolve as additional engineering context becomes available. Rather than treating AI as a code generator, the assistant reviews engineering artefacts, identifies ambiguity, organizes information, and supports engineering activities while leaving judgement and decision-making with the Quality Engineer.

Using a representative feature from the fictional connected vehicle platform **AutoPulse**, the paper demonstrates how the same engineering conversation progresses from requirement refinement through implementation review, verification planning, automation maintenance, CI/CD investigation, and release readiness.

Although the examples use Claude, the workflow is independent of any specific AI platform and can be adapted to any capable assistant.

© 2026 Nagarro

Introduction

Today's Quality Engineers contribute long before testing begins and continue after the implementation. Their responsibilities include reviewing requirements, understanding implementation changes, defining verification strategies, maintaining automation, analyzing pipeline failures, and assessing release readiness.

Currently the application of AI in software testing is limited to generating artefacts like test cases or automation scripts. However, these represent only a fraction of the decisions made during a typical sprint.

A much better way to approach AI application in quality engineering is to ask: **How AI can support the day-to-day engineering activities instead?**

This paper answers the question through a workflow that maintains **one continuous engineering conversation** throughout the a feature lifecycle.

As new artefacts emerge throughout the sprint, including requirements, code changes, regression suites, execution results, and investigation evidence, the conversation builds on existing engineering context instead of starting from scratch each time.

The objective is not automated decision-making. It is better-informed engineering artefacts. (In this paper, engineering artefacts include story analysis, gap analysis, test cases, automation structures, draft pull requests, and other tangible outputs produced during the quality engineering process.)

© 2026 Nagarro

AI as a QA assistant

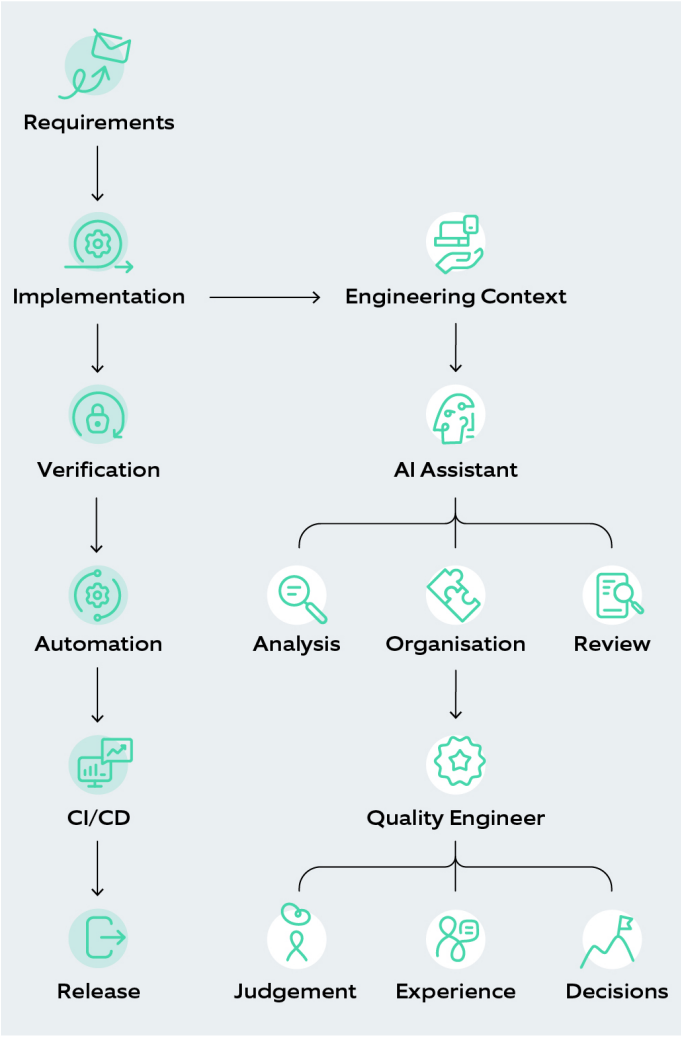
Every feature generates engineering artefacts for instance:

- Requirements describe expected behaviour
- Implementation introduces change
- Regression suites capture existing coverage
- Automation reflects the verification strategy
- CI/CD pipelines produce execution evidence
- Defects reveal unexpected behaviour and remaining risks

Each artefact captures a different aspect of the feature. Together, they provide a complete view of its current engineering state. AI assistant supports the quality engineer by reviewing these artefacts as they become available throughout the sprint.

Instead of treating each activity as an isolated prompt, the same conversation continues from one stage to the next, allowing recommendations to evolve as additional engineering context is introduced.

The responsibilities remain clearly separated.



Assistant	Quality engineer
Reviews engineering artefacts	Interprets business and technical context
Identifies ambiguity and potential risks	Defines the verification strategy
Organises engineering information	Investigates technical issues
Highlights areas requiring attention	Makes engineering decisions

© 2026 Nagarro

AutoPulse case study

The following representative case study illustrates how the proposed workflow applies to feature development.

Let's imagine AutoPulse, a connected Vehicle platform that enables users to monitor vehicle health information, including battery status, tyre pressure, diagnostics, service reminders, and maintenance alerts.

During one sprint, the engineering team enhances the vehicle health dashboard by replacing a single telemetry service with multiple independent services. The change improves resilience and availability while introducing new engineering considerations:

- **Asynchronous responses,**
- **Retry behaviour,**
- **Cached information,**
- **Partial service failures,**
- **Data consistency across services.**

The Quality Engineer follows the feature throughout the sprint, using the engineering context available at each stage to support requirement discussions, implementation review, verification planning, automation updates, investigation, and release readiness.

The following sections show how an AI assistant such as Claude supports the Quality Engineer from story analysis through implementation review and verification.

Requirement refinement

Requirement refinement provides the earliest opportunity to improve software quality. Before implementation begins, the Quality Engineer reviews the user story and acceptance criteria to identify ambiguity, missing business rules, assumptions, and edge cases.

ILLUSTRATIVE PROMPT

REVIEW THE USER STORY AND ACCEPTANCE CRITERIA.

IDENTIFY AMBIGUITY, MISSING BUSINESS RULES, ASSUMPTIONS, AND EDGE CASES.

DO NOT GENERATE TEST CASES.

The review highlights several areas requiring clarification.

The Quality Engineer, with the relevant product and engineering stakeholders, resolves these questions. AI only helps surface them. Clarifying requirements early reduces ambiguity before it becomes an implementation or verification risk.

Observation	Quality engineer
Data freshness	Should the dashboard display when the information was last refreshed?
Partial service failure	Should available information remain visible if one telemetry service is unavailable?
Retry behaviour	What retry limits and timeout expectations should apply?
Conflicting telemetry	Which source should be considered authoritative?

© 2026 Nagarro

AutoPulse case study

Reviewing implementation changes

Once development begins, the quality engineer reviews how the implementation affects application behaviour.

Rather than analyzing the code alone, they review the implementation against the approved requirements to identify behavioral changes that may influence regression coverage.

ILLUSTRATIVE PROMPT

REVIEW THE IMPLEMENTATION CHANGES AGAINST THE APPROVED REQUIREMENTS.

SUMMARISE BEHAVIORAL CHANGES AND IDENTIFY AREAS THAT MAY AFFECT REGRESSION COVERAGE.

IGNORE CODING STYLE RECOMMENDATIONS.

The review identifies the following factors.

Implementation change	Quality consideration
Multiple telemetry services introduced	Validate data aggregation behaviour
Independent widget loading	Verify partial dashboard availability
Cached information retained	Confirm expected behaviour during service interruptions
Retry mechanism added	Validate recovery and timeout scenarios

The behavioural review provides a focused understanding of the implementation changes and helps prioritize regression activities where application behaviour has evolved.

© 2026 Nagarro

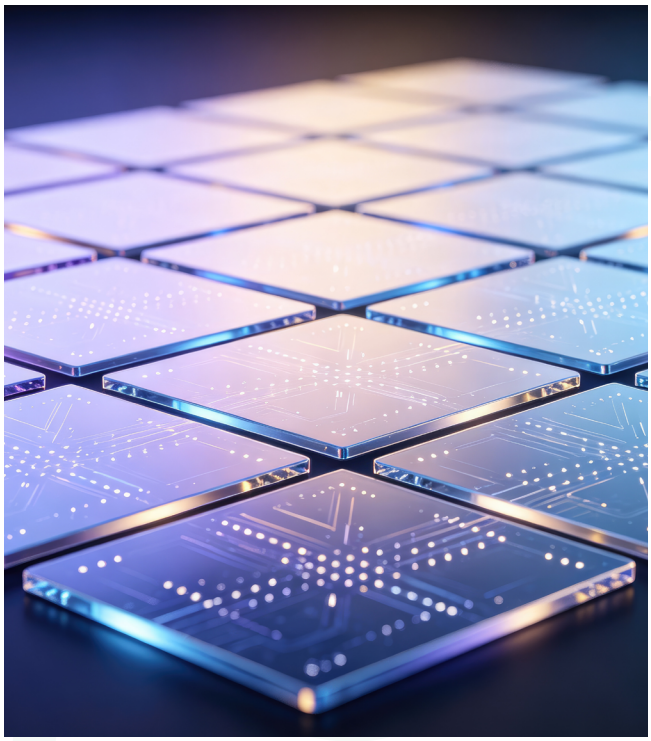
AutoPulse case study

Verification strategy

With both the requirements and implementation context available, Quality Engineer can now determine where to focus their verification efforts.

ILLUSTRATIVE PROMPT

REVIEW THE REQUIREMENT CHANGES, IMPLEMENTATION IMPACT, AND EXISTING REGRESSION COVERAGE. IDENTIFY VERIFICATION AREAS AFFECTED BY THE CHANGE. DO NOT CREATE A NEW TEST SUITE.



The review identifies the following factors.

The objective is not to expand the regression suite unnecessarily. It is to ensure that existing verification remains aligned with the current application behaviour.

Area	Verification focus
Functional behaviour	Dashboard loading, refresh behaviour, and displayed information
Service interaction	Telemetry availability, retry handling, and response handling
Regression impact	Existing scenarios affected by dashboard behaviour changes

A mature quality engineering practice measures success by verifying the right risks rather than by increasing the number of test cases.

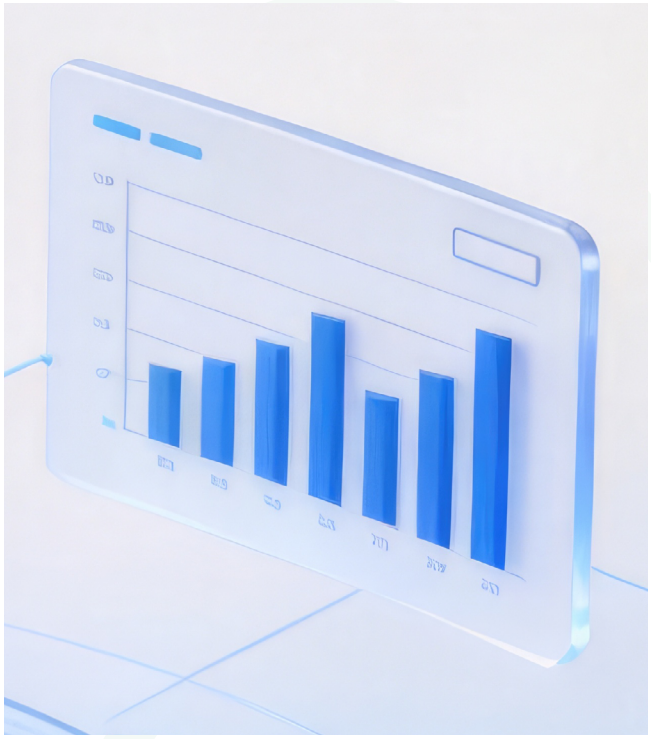
© 2026 Nagarro

AutoPulse case study

Automation development

Implementation changes rarely require an entirely new automation suite. More often, they require existing automation to evolve alongside the application.

Using the implementation review together with the existing automation framework, the Quality Engineer identifies which automated scenarios are affected by the behavioral changes.



ILLUSTRATIVE PROMPT

REVIEW THE REQUIREMENT CHANGES, IMPLEMENTATION IMPACT, AND EXISTING REGRESSION COVERAGE. IDENTIFY VERIFICATION AREAS AFFECTED BY THE CHANGE. DO NOT CREATE A NEW TEST SUITE.

The review identifies targeted updates.

Existing automation	Update required
Dashboard validation	Verify independently loaded telemetry widgets
Service availability checks	Validate individual service status indicators
Cached data validation	Confirm previously retrieved values remain visible during temporary service interruptions
Refresh validation	Verify timestamp updates following successful service recovery

Rather than expanding the automation suite, the existing regression coverage is updated to reflect the behavioral changes introduced by the feature.

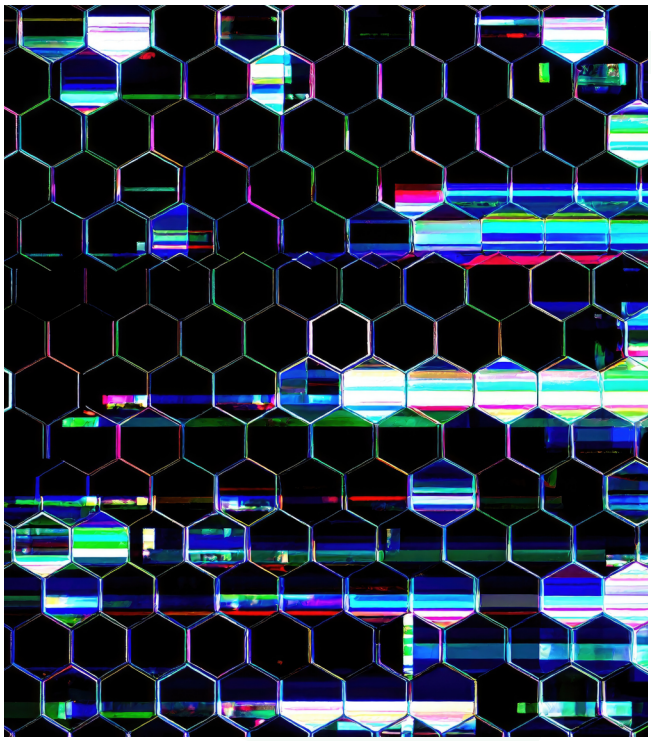
© 2026 Nagarro

AutoPulse case study

CI/CD investigation

As the feature progresses through continuous integration, execution failures begin to appear. Before investigating individual failures, the quality engineer organizes the available execution evidence.

Execution reports, application logs, screenshots, and API responses are reviewed together.



ILLUSTRATIVE PROMPT

REVIEW THE EXECUTION REPORTS AND SUPPORTING ARTEFACTS.

GROUP FAILURES WITH SIMILAR CHARACTERISTICS AND SUMMARISE RECURRING OBSERVATIONS.

DO NOT DETERMINE THE ROOT CAUSE OR RECOMMEND CODE CHANGES.

The review produces a structured investigation summary.

Observation	Engineering focus
Similar failures grouped together	Reduce duplicate investigation effort
Recurring error patterns identified	Prioritize common failure characteristics
Frequently affected components highlighted	Focus technical investigationinterruptions
Consolidated execution summary prepared	Establish a common starting point for analysis

Grouping evidence before investigation allows engineers to begin with an organized view of the failures while preserving independent technical analysis and root cause determination.

© 2026 Nagarro

AutoPulse case study

Release readiness

By the end of the sprint, engineering information is distributed across multiple artefacts, including verification results, automation reports, pipeline executions, and outstanding defects.

Preparing for a release discussion requires consolidating this information into a single engineering view.

ILLUSTRATIVE PROMPT

SUMMARISE THE CURRENT VERIFICATION STATUS USING COMPLETED TESTING, AUTOMATION EXECUTION, OUTSTANDING DEFECTS, AND DEPLOYMENT RESULTS.

HIGHLIGHT ENGINEERING RISKS REQUIRING DISCUSSION.

DO NOT RECOMMEND WHETHER THE RELEASE SHOULD PROCEED.

The resulting summary includes:

- **Completed verification activities**
- **Automation execution status**
- **Outstanding defects**
- **Unresolved engineering risks**
- **Areas requiring discussion during the release review**

The summary supports release discussions by consolidating engineering information, while the release decision remains the responsibility of the engineering team.

Observation	Engineering focus
Similar failures grouped together	Reduce duplicate investigation effort
Recurring error patterns identified	Prioritize common failure characteristics
Frequently affected components highlighted	Focus technical investigationinterruptions
Consolidated execution summary prepared	Establish a common starting point for analysis

© 2026 Nagarro

Practical considerations

The effectiveness of this workflow depends less on prompt wording than on the quality of the engineering context maintained throughout the sprint. Maintaining one continuous conversation allows each review to build upon the preceding engineering discussions.

Requirements refined during backlog discussions remain available during implementation review. Behavioral changes identified during development continue to inform verification planning, automation updates, CI/CD investigation, and release readiness.

As the feature evolves, the engineering context evolves with it.

The following practices help maintain an effective workflow:

- **Maintain continuity:** Continue the same conversation throughout the sprint rather than creating isolated interactions for each activity.
- **Provide engineering context:** Requirements, implementation changes, regression coverage, automation, execution reports, and investigation evidence together produce more meaningful recommendations than any individual artefact in isolation.
- **Review every recommendation:** Validate recommendations against project requirements, domain knowledge, and engineering standards.
- **Protect engineering information:** Only share source code, execution logs, and project artefacts with organization-approved AI services and in accordance with security and governance policies.

The workflow complements existing quality engineering practices by improving access to organized engineering information while preserving human judgement and accountability.

© 2026 Nagarro

Conclusion

Rather than focusing on code generation or test case creation, the proposed workflow demonstrates how engineering artefacts can be reviewed continuously as a feature progress through requirement refinement, implementation, verification, automation, CI/CD investigation, and release readiness.

The central contribution is straightforward: ***Maintain one continuous AI conversation throughout the lifecycle of a feature so that engineering recommendations evolve as additional context becomes available.***

In this workflow, the assistant reviews information, identifies ambiguity, organizes evidence, and highlights areas requiring attention. Engineering judgement including verification strategy, technical investigation, and release decisions remains with the quality engineer.

As AI capabilities continue to evolve, individual tools will inevitably change. A workflow centred on engineering context, continuous collaboration, and human accountability is more durable because it is based on engineering practice rather than vendor-specific capabilities.

Ultimately, the value of this approach is not measured by the number of artefacts generated. It is measured by how effectively it helps Quality Engineers understand software change, focus verification effort, and make informed engineering decisions throughout the Software Development Life Cycle.

© 2026 Nagarro

About the author

Megha Agarwal is an Associate Principal Engineer with thirteen years of experience in quality engineering, test automation, and engineering transformation. She has worked across automotive, finance, taxation, and e-commerce domains, leading initiatives in automation architecture, CI/CD, and quality engineering practices.

Her current interests include exploring practical applications of Artificial Intelligence in quality engineering, with a particular focus on workflows that strengthen engineering collaboration while preserving human judgement and accountability.

Megha Agarwal

References

1. Beck, K., et al. ***Manifesto for Agile Software Development***. 2001.
2. Humble, J., & Farley, D. ***Continuous Delivery.*** Addison-Wesley Professional.
3. Forsgren, N., Humble, J., & Kim, G. ***Accelerate: The Science of Lean Software and DevOps***. IT Revolution Press.
4. Fowler, M. ***Continuous Integration***. Thoughtworks.
5. ***Anthropic***. Claude Documentation.
6. ***OpenAI.*** ChatGPT Documentation.
7. ***ISTQB®.*** Certified Tester Foundation Level Syllabus.
8. ***ISO/IEC/IEEE 29119 – Software Testing Standards***.

© 2026 Nagarro