Achieving Continuous Traceability: From Requirement to Code and Back

NotebookLM

The Traceability Gap: Why Manual Processes Break Down

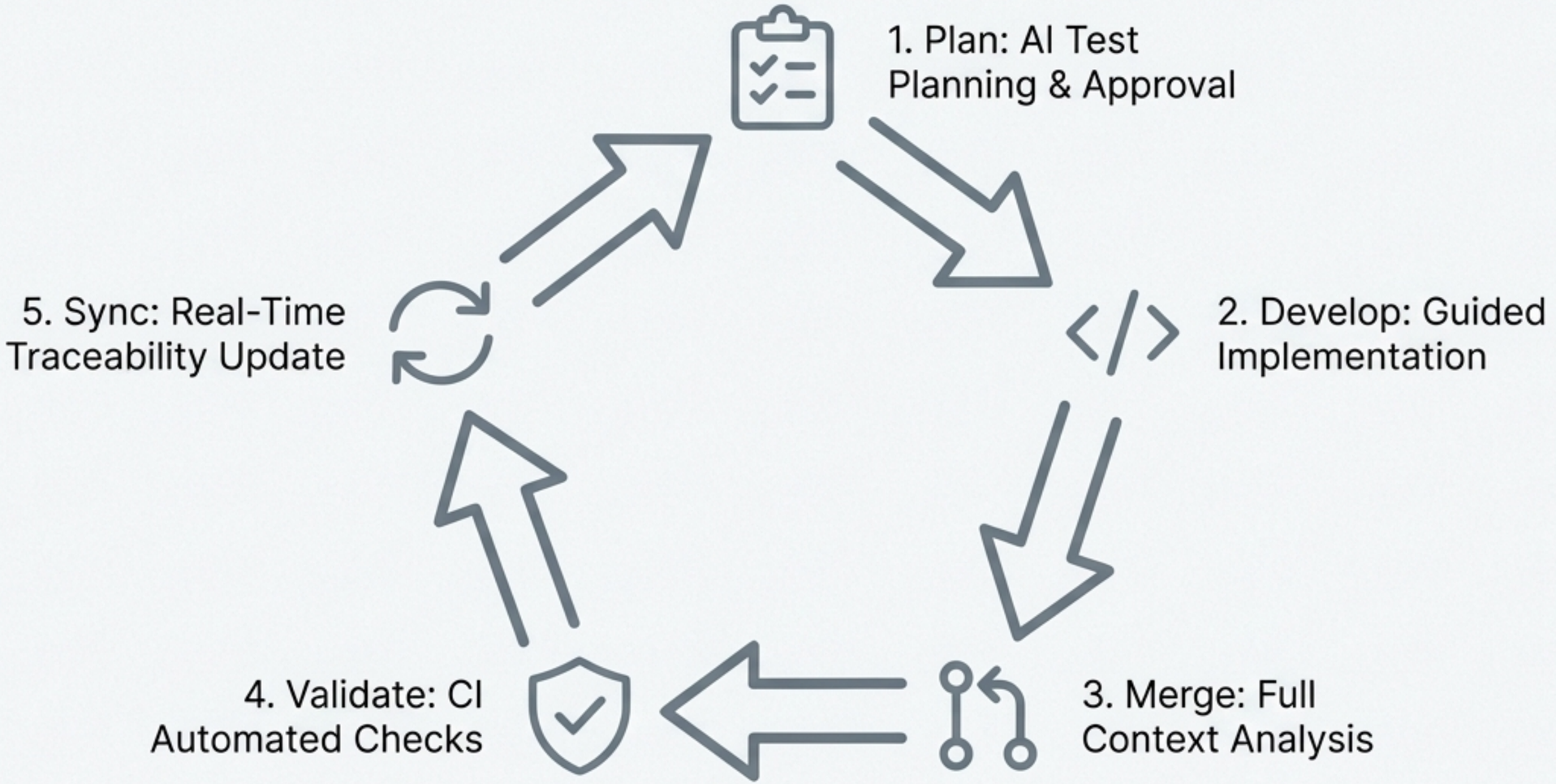
In complex software development, especially in regulated industries, maintaining a clear link between requirements, code, and tests is critical. Manual approaches are often:

- **Brittle:** Links break with every code change, becoming outdated almost immediately.
- **Time-Consuming:** Developers and QA spend countless hours manually updating test cases and traceability matrices.
- **Risky:** Gaps in traceability lead to failed audits, missed requirements, and critical quality issues discovered too late.

NotebookLM

The Intelligent Quality Loop: A Five-Stage Automated Workflow

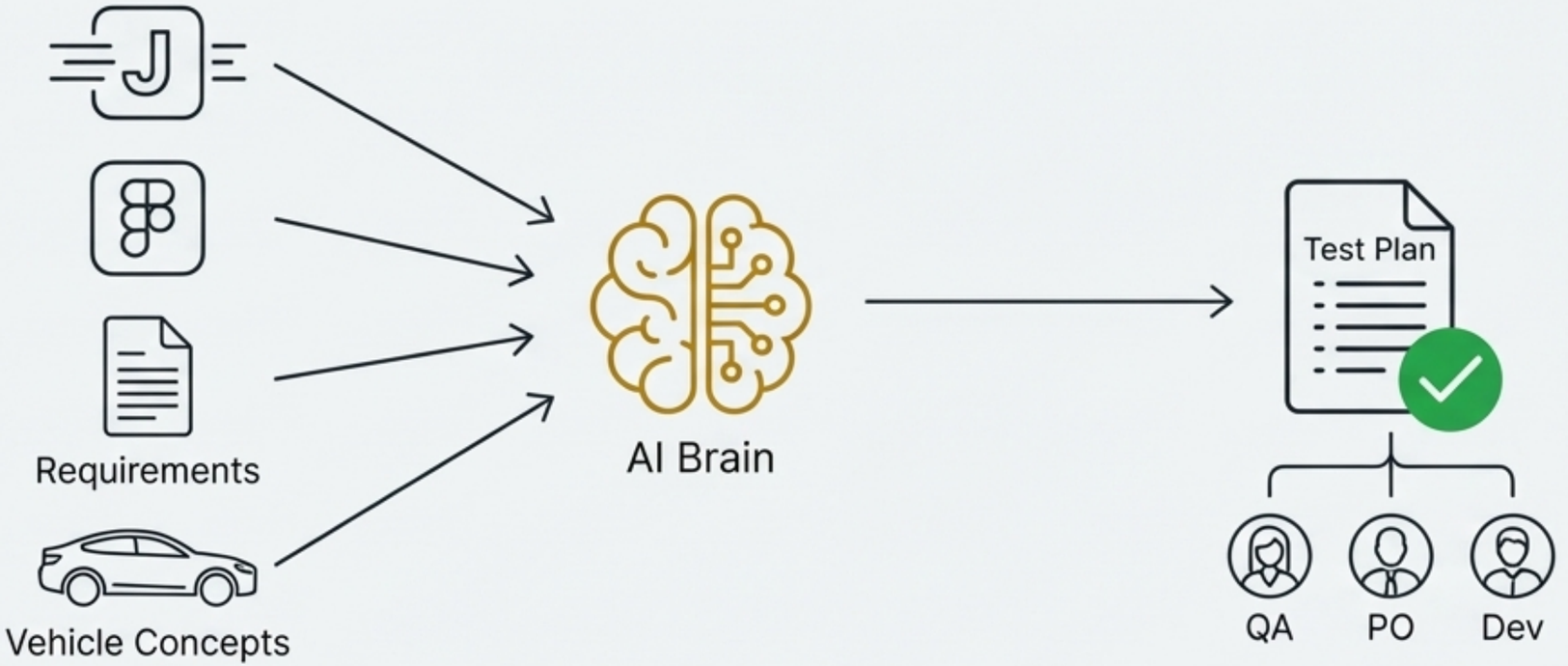
We solve this with a closed-loop system powered by AI that ensures traceability is continuous, automated, and enforced at the most critical points of the development lifecycle.



NotebookLM

Stage 1: Building a Foundation of Clarity Before Code

Pre-Development AI Test Planning & Team Approval



Key Outcome: The Jira story is marked "**Ready for Implementation**." This creates a locked-in, baseline test plan that guides all subsequent work.

NotebookLM

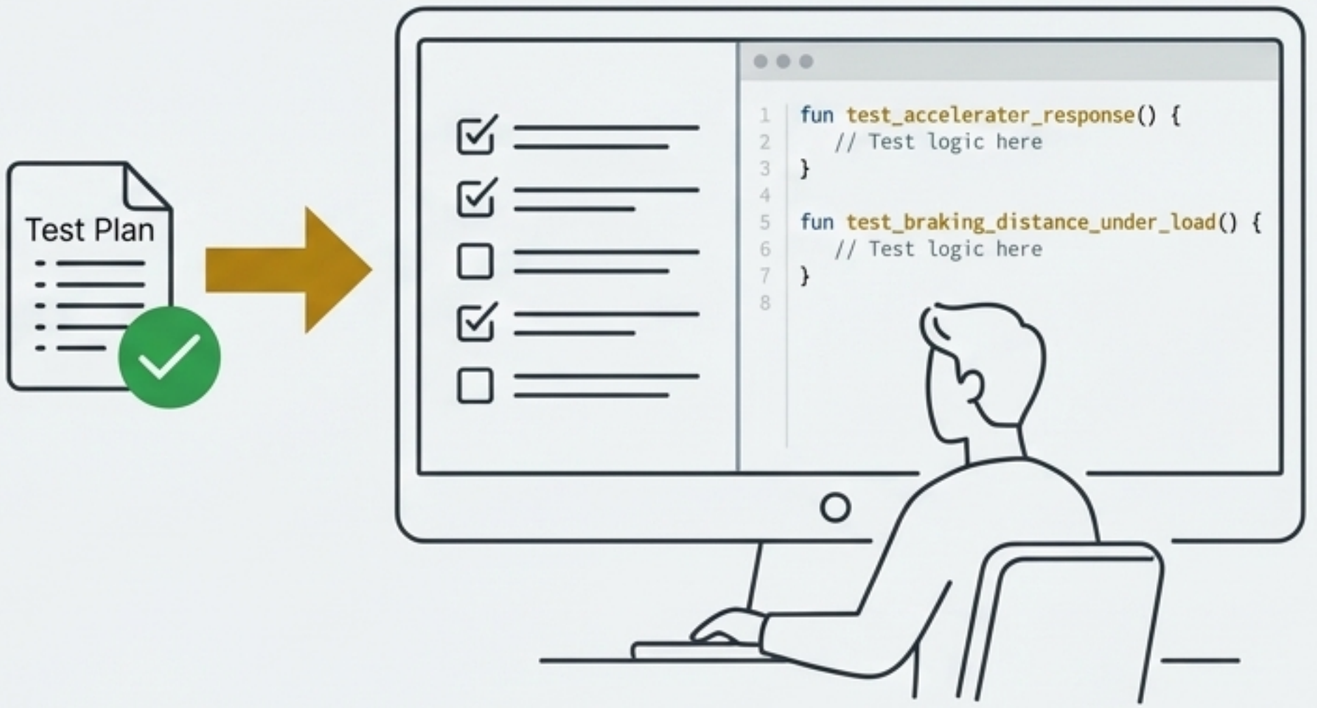
Stage 2: Developing with Precision and Intent

Guided Implementation Based on the Approved Test Plan

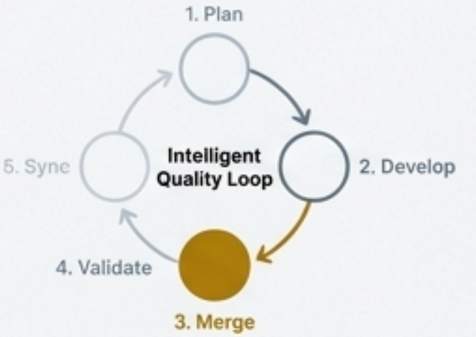
Developers don't just implement the feature; they simultaneously write the automated tests required to meet the approved coverage. The high-level plan from Stage 1 provides full clarity on what needs to be tested.

Developer Output:

- SWE.4 automated Kotlin tests (by Dev)
- SWE.5 automated Kotlin tests (by Dev)
- SWE.5 automated Kotlin tests (by Dev)
- SWE.6 automated Kotlin tests (by QA)



NotebookLM



Stage 3: The Merge Request Becomes the Point of Full Context

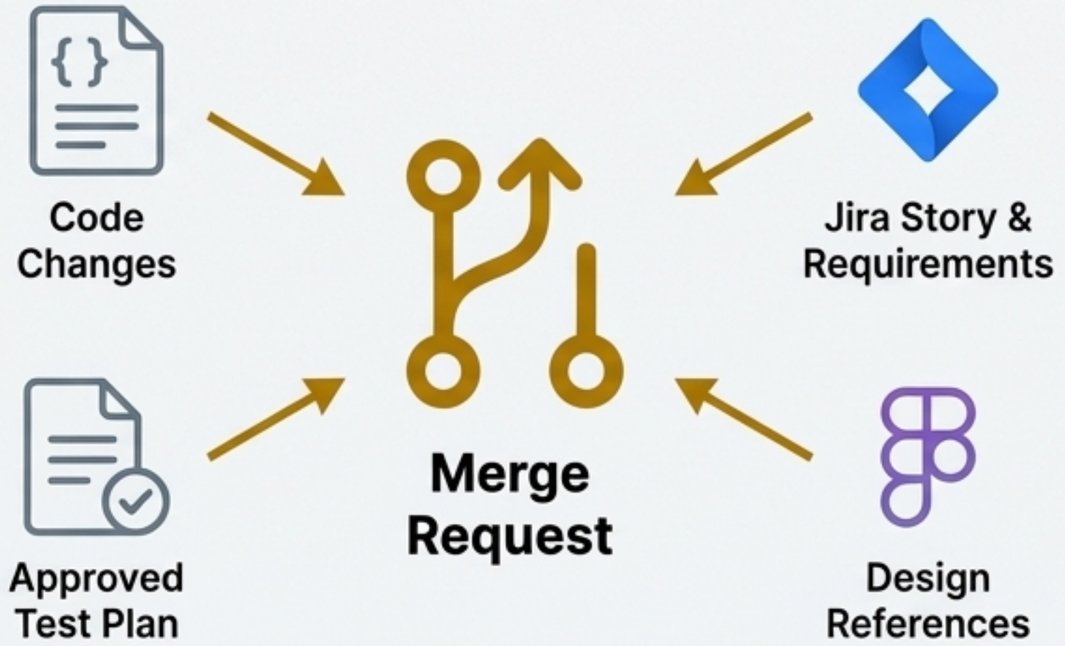
Triggering Deep Analysis Before Code is Merged

The Trigger

A traceability check is a **mandatory gate** before any Merge Request (MR) can be closed. Developers or QA can also trigger it manually at any time during the review process.

The Critical AI Action

Using this full context, the AI generates highly **Detailed Test Cases**, including preconditions, test steps, and expected results. This level of detail is only possible now that the actual implementation code is available for analysis.

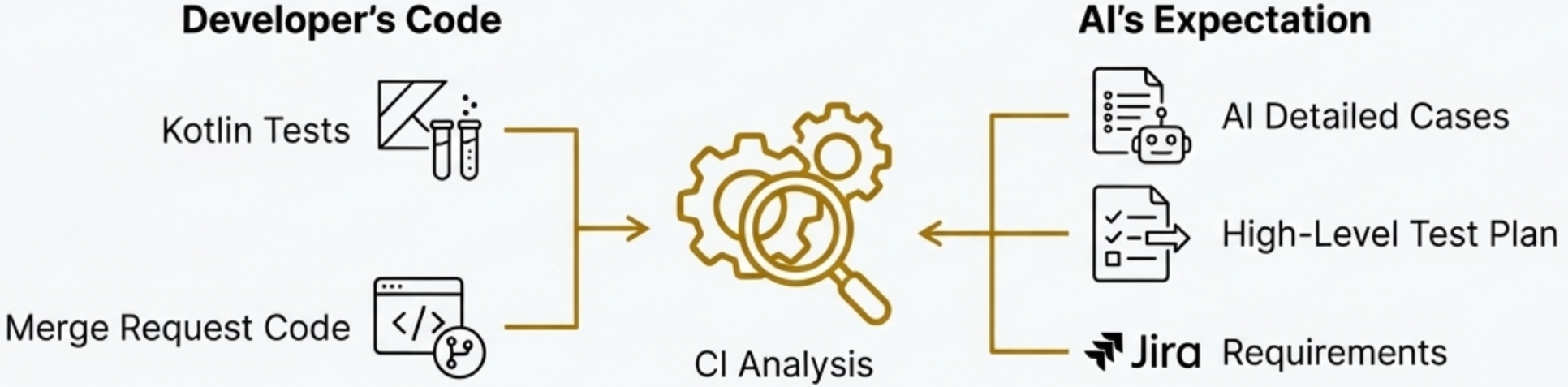


AI Achieves Full Context

NotebookLM

Stage 4: CI Automatically Validates Implementation Against Expectation

The Merge Request Validation Process



Key Checks Performed

- ✔ Do the implemented tests match the AI's detailed cases?
- ✔ Is the expected test coverage fully implemented, not missing or partial?
- ✔ Is the Code → Requirement → Test alignment correct and unbroken?

NotebookLM

Stage 5: Surfacing Coverage Gaps in Real-Time

Providing Immediate, Actionable Feedback within the Merge Request

Merge Request #1337: Add Adaptive Cruise Control Logic

AI Traceability Check

Status: Action Required

- Covered: Scenario: Maintain safe following distance above 50 kph.
- Partially Covered: Scenario: Handle lead vehicle cut-in event.
- **Missing:** Scenario: Test system behavior in heavy rain conditions.
- Covered: Scenario: Disengage system upon driver brake input.

Enforced Quality Gate: Developers and QA are required to address all `Partially Covered` or `Missing` cases before the Merge Request is allowed to be approved and merged.

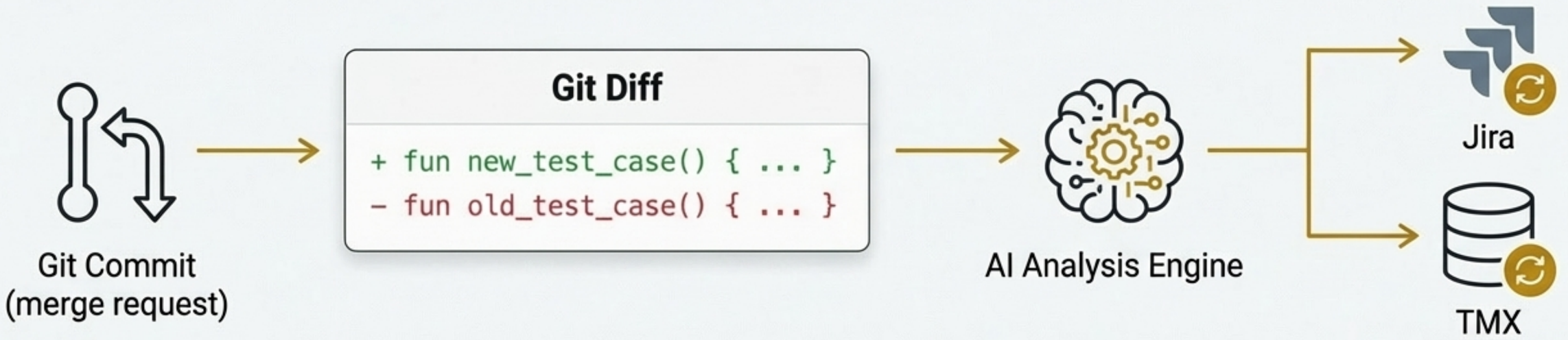
NotebookLM

The Synchronization Engine: How Git Diffs Drive Continuous Traceability

The final stage is not just about detecting gaps, but about keeping the entire test management system perfectly in sync with the codebase. Our system uses the **Git diff** of the automated Kotlin test files as the definitive trigger for all traceability updates.

The Logic

By analyzing precisely what test code has been added, changed, or removed, the AI can execute targeted, automatic updates in Jira and the Test Management eXchange (TMX).



NotebookLM

Keeping Jira and Code in Sync: The Three Core Scenarios

➕ NEW Tests Detected

Action
AI and Jira automatically create new TMX test case entities.

Linkage
These new TMX tests are automatically linked to the related Requirements in Jira.

Outcome
A complete, new traceability chain is established instantly.

✏️ MODIFIED Tests Detected

Action
AI identifies the corresponding TMX test using the function or class name as a key.

Linkage
Jira updates the content (steps, expected results) of the existing TMX test to reflect the new behavior in the code.

Outcome
The requirement mapping is verified and updated automatically.

🗑️ DELETED Tests Detected

Action
Jira finds the corresponding TMX test linked to the removed code.

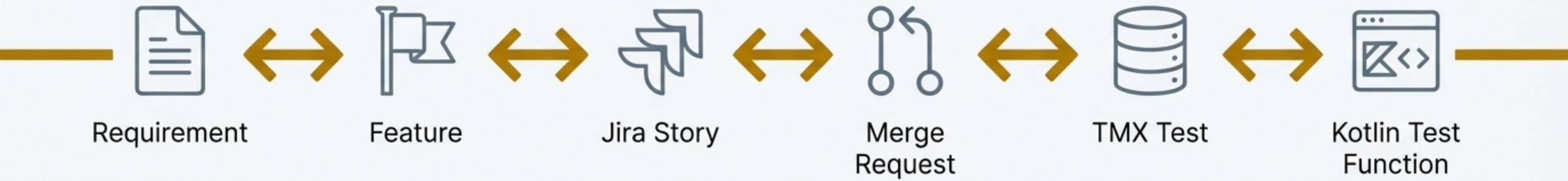
Linkage
The TMX test's status is automatically marked as **Obsolete**.

Outcome
The test plan accurately reflects that the test has been intentionally removed from the codebase.

NotebookLM

The Unbroken Chain: End-to-End, Bi-Directional Traceability

The result is a live, unbreakable link from the highest-level business requirements directly to the specific functions in the code that implement and test them—and back again.



NotebookLM

The Intelligent Quality Loop at a Glance

- **Before Dev**: AI creates a high-level test plan for team approval.
- **The Gate**: A story is only **Ready for Implementation** after plan approval.
- **At MR**: With **Full Context** (including code), AI generates detailed test cases.
- **The Check**: The MR cannot be closed until a traceability check passes.

- **In CI**: The CI pipeline compares implemented Kotlin tests against AI expectations.
- **The Sync**: **Git diff** logic keeps TMX tests perpetually in sync (New/Modified/Deleted).
- **The Result**: Full, real-time traceability is maintained across Jira, code, and tests.

NotebookLM

Tangible Outcomes Across the Development Lifecycle

Reduce Risk & Accelerate Delivery

- Create a complete, automated audit trail for compliance.
- Detect requirement gaps and test coverage issues before they are merged.
- Shorten MR review cycles by providing full context automatically.

Enhance Developer & QA Productivity

- Eliminate the manual, error-prone task of updating test cases in Jira/TMX.
- Provide absolute clarity on test expectations before development starts.
- Focus reviews on logic and quality, not on chasing traceability links.

Build a Foundation of Quality

- Ensure test coverage is a non-negotiable part of the definition of done.
- Maintain a live, accurate view of test status against requirements.
- Transform traceability from a compliance chore into a quality-driving asset.

NotebookLM

This Isn’t Just Automation. It’s Continuous Intelligence.

The traditional approach treats traceability as a historical record, often compiled after the fact. This system transforms it into a **live, intelligent guide that actively informs development, enforces quality in real-time,** and creates a resilient, self-maintaining link between intent and implementation. It is the new standard for engineering excellence.

NotebookLM