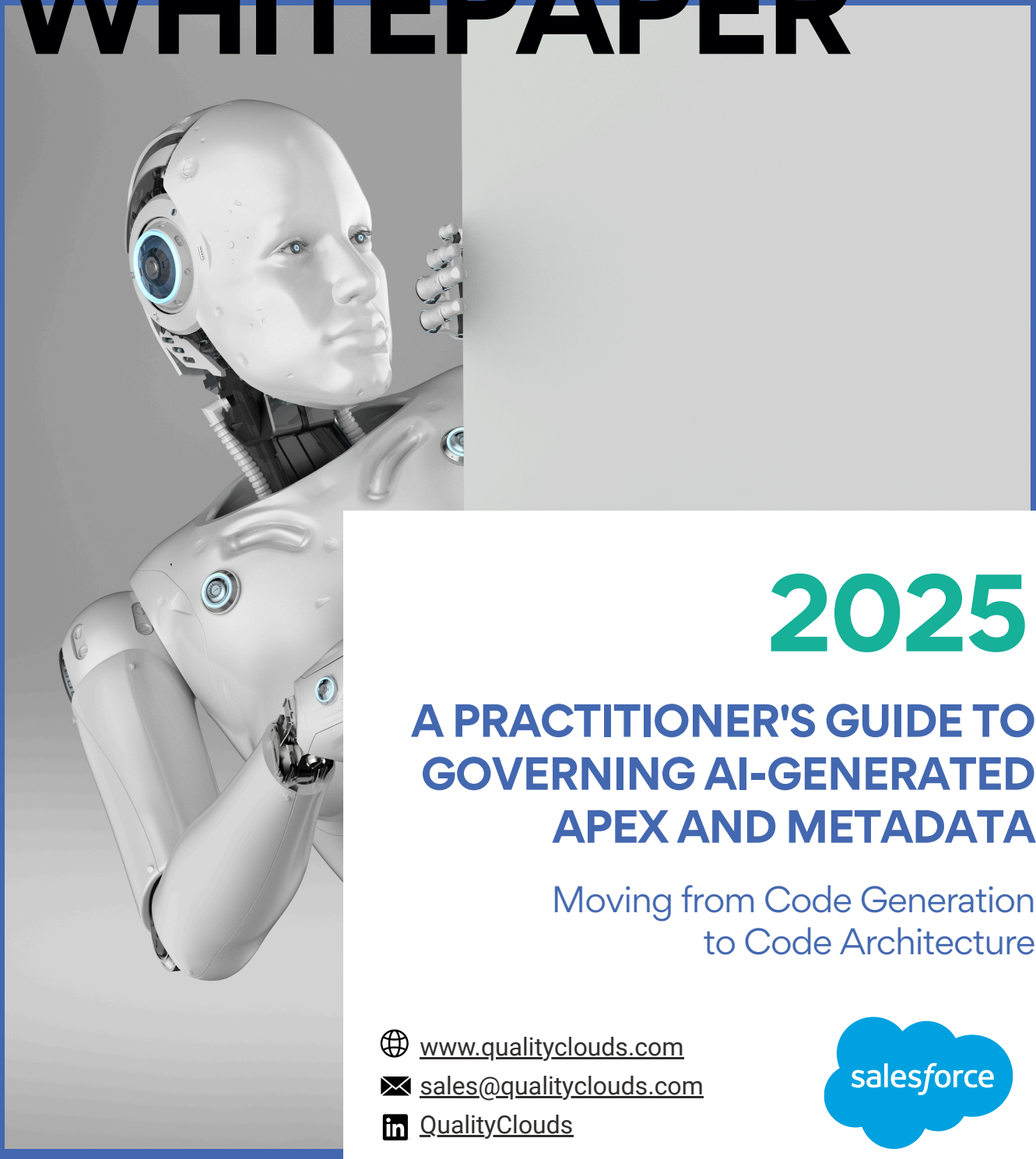
QualityClouds

BEYOND THE PROMPT WHITEPAPER



2025

A PRACTITIONER'S GUIDE TO GOVERNING AI-GENERATED APEX AND METADATA

Moving from Code Generation to Code Architecture

www.qualityclouds.com
sales@qualityclouds.com
QualityClouds

salesforce

TABLE OF CONTENTS

Quality Clouds

About Quality Clouds for Salesforce

Quality Clouds is a Salesforce-native governance platform that brings real-time quality and compliance enforcement directly into the development lifecycle.

Trusted by global enterprises since 2019, Quality Clouds continuously validates AI-generated and human-built code, configuration, and metadata, ensuring every change meets organizational standards before it reaches production.

By embedding governance directly into the development process, Quality Clouds enables teams to move fast with AI while maintaining control, compliance, and long-term platform health.

Linde
DHL
edp
ZURICH
USAA
MAYO CLINIC
WELLS FARGO
bp
Nestlé
IHG HOTELS & RESORTS
Ameriprise Financial
MERCK
citi
BASF
CaixaBank
Heraeus
JPMorgan Chase & Co.
CommonwealthBank
CUSHMAN & WAKEFIELD
BARCLAYS

1. The New Unit of Work: From "Coder" to "Architect"

In the space of two years, the definition of "Salesforce Development" has fundamentally shifted. The days of hand-crafting every line of Apex or meticulously building every Flow element are fading. Today, development is increasingly about prompting an assistant for a feature—"Create a trigger to sync Contacts to a custom object"—and reviewing the output.

This shift changes your role. You are no longer just a writer of syntax; you are becoming an Architect of Logic.

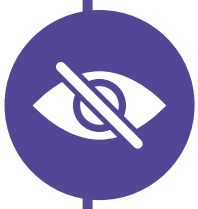
With tools like Einstein for Developers and GitHub Copilot, **productivity has jumped by 35–45%.** But this speed comes with a hidden cost: the collapse of the review cycle. Research shows that **94% of AI code suggestions are accepted without manual review** once they appear functional.

For Salesforce practitioners, this creates a new career imperative. The market value is no longer in generating the code—AI does that for free. The value is in governing it. The most successful developers of 2025 will be those who understand the business context deeply enough to prevent AI from rebuilding what Salesforce already provides.

2.The Hidden Trap: Why "Working Code" Fails in Production

AI models are trained on billions of lines of generic code, but they are not trained on your Org or the full breadth of the Salesforce AppExchange. They optimize for plausibility, not for your specific governor limits, sharing model, or architectural standards.

Our empirical data shows that AI-generated logic introduces approximately **one issue every six lines of code.**
These aren't just syntax errors; they are architectural blind spots:

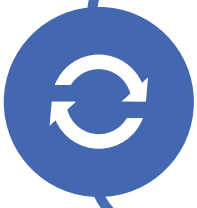


Feature Blindness

AI will happily write 500 lines of custom Apex to solve a problem that standard Salesforce CPQ, OmniStudio, or a simple Flow could solve out of the box. It doesn't know you already own a license for an App that handles this, leading to massive, unnecessary technical debt.

Context Hallucinations

Referencing custom fields or picklist values that exist in public training data but not in your specific Org.



The Loop Trap

Placing SOQL queries or DML operations inside loops because the "logic flow" made sense to the LLM, ignoring Salesforce's multi-tenant governor limits.

QualityClouds

The "Invisible" Risk: AI-Generated Metadata

The risk isn't limited to code. AI tools now generate metadata—Flows, validation rules, and configuration XML—that have just as much impact on your Org's health.

Apex Code

- ✓ Reviewed
- ✓ Tested
- ✓ Looks safe

MEDATA & CONFIG

- Flows
- Validation Rules
- Execution Context

INVISIBLE RISK

A "clean" Apex class can still be dangerous if the accompanying metadata opens a security hole. For example, an AI-generated Flow might default to "System Context," bypassing your carefully crafted sharing rules. Traditional code reviews often miss these configuration drifts because they happen outside the IDE, in the "declarative" layer.

3.The Practitioner's Toolkit: How to Audit AI Output

Until automated governance is fully adopted, every Salesforce Architect should apply a strict manual audit to AI contributions. Use this 5-Point Checklist to catch the issues that standard unit tests will miss:

1

The Functional Duplication Audit (Build vs. Standard)

- Check: Is the AI building a custom "Round Robin" assigner when you already have Sales Cloud Assignment Rules or an installed AppExchange package that does this?
- Check: Does this custom LWC replicate standard functionality that could be configured instead of coded?

2

The Governor Limit Sensitivity Check

- Check: Look for SOQL/DML inside loops.
- Check: Verify that lists are used for bulk processing rather than single-record transactions

3

The Metadata & Flow Context Audit

- Check: Is the Flow running in "System Mode" or "User Mode"? Ensure it respects the running user's permissions.
- Check: Are there hardcoded IDs in decision elements or criteria?
- Check: Does this Flow duplicate logic that already exists in an Apex Trigger, creating a race condition?.

4

The Security "Blind Spot" Check

- Check: Does the Apex explicitly check isAccessible() before reading data?.
- Check: Are inputs sanitized before being used in dynamic SOQL to prevent injection attacks?.

5

The "Zombie" Logic Check

- Check: Did the AI create a new Utility Class (e.g., for Logging) instead of using your Org's existing framework?.
- Check: Is it introducing a new architectural pattern that conflicts with your established standards?.

4.Building a Governance Protocol (Without New Tools)

Not every team is ready to deploy automated governance immediately. Furthermore, many Salesforce teams operate in environments where Git is "invisible"—abstracted away by DevOps tools (like Copado, Flosum, or Salesforce DevOps Center) or bypassed entirely via direct deployments.

If you cannot run CLI commands or see Git diffs, you must move governance upstream (to the User Story) and downstream (to the Platform Metadata).

To govern AI in a "No-CLI" environment, we recommend the **"Declaration & Description" Protocol**.

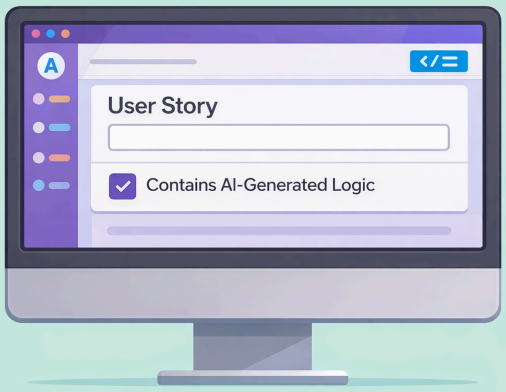
QualityClouds

"Declaration & Description" Protocol

I. The "User Story" Declaration (Upstream)

If you can't tag a commit, you must tag the requirement. Update your project management workflow (Jira, Azure DevOps) to include a mandatory "AI Usage" field on every User Story.

- **The Rule:** Before a story moves to "Ready for QA," the developer must check a box: "Contains AI-Generated Logic."
- **Why:** This signals your Release Manager or Lead Architect that this specific deployment carries higher risk (hallucinations, context gaps) and requires a "Logic Audit" rather than just a functional test. Or even for you for further reference.



II. The "Description Field" Log (Downstream)

AI code often lacks context. Since you might not have easy access to a Git history to read commit messages, use the platform itself.

- **The Rule:** Enforce a policy where the Description field of any new Flow, Apex Class, or Custom Field must start with [AI-Gen].
- **Why:** This stamps the provenance directly onto the metadata. Six months from now, when a Flow fails in production, the admin debugging it will immediately know to check for "AI hallucinations" rather than assuming human intent.

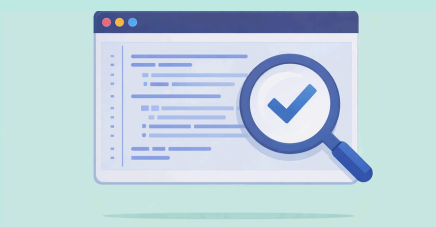
QualityClouds

"Declaration & Description" Protocol

III. The "DevOps Scanner" Toggle

Most commercial DevOps tools (Copado, Flosum) have built-in static analysis gates (often PMD-based) that are frequently left disabled to "speed up deployments."

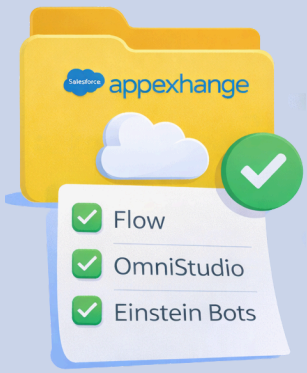
- **The Rule:** Enable the strictest available rule set on your deployment pipeline, specifically for "Bulkification" and "Security" rules.
- **The Limitation (Generalist vs. Specialist):** Be aware that standard scanners are static. They check syntax (e.g., "Missing curly brace") but rarely check context (e.g., "Is this Flow running in System Mode?" or "Does this field exist in the target Org?").
- **The Solution:** This is why Specialist Governance Platforms like Quality Clouds are distinct from standard code scanners. While a generic scanner looks at the text file, a specialist platform looks at the Salesforce Platform—validating metadata, configuration, and code together to catch the architectural risks that PMD misses. If your DevOps tool allows it, swap the generic scanner for a specialist integration to close this gap.



IV. The "AppExchange First" Mandate

Before prompting an AI to build a feature, enforce a mandatory "Capability Check."

- **The Rule:** Implementation Guides must explicitly list standard Salesforce features (e.g., Flow, OmniStudio, Einstein Bots) that must be ruled out before custom Apex is authorized.
- **Why:** AI loves to write code. It will happily write a custom lead routing engine that duplicates standard Sales Cloud Assignment Rules. A human architect must interrupt this "build-first" bias.



5. The Architect's Future

The strategic choice for Salesforce professionals is not "AI vs. No AI." It is "Fast, Opaque Change" vs. "Fast, Governed Change".

By moving from a mindset of "writing code" to "architecting logic," you position yourself as the indispensable guardian of the Org.

Tools like **LivecheckAI** are the enablers of this transition, turning your governance policies into automated guardrails that allow your team to move at AI speed—safely.

Take the Next Step

Don't just trust the prompt. Audit your AI readiness and see what "invisible" risks are already sitting in your codebase.

Audit Your AI Readiness

THANK YOU

Validate Your AI Logic, FREE

www.qualityclouds.com
sales@qualityclouds.com
QualityClouds

20-22 Wenlock Rd / London N1 7GU
224 W 35th St, Suite 500 PMB 112 / New York, NY 10001
Carrer de Torres i Amat 21, 1" / 08001 Barcelona
LONDON / NEW YORK/ BARCELONA