Case Study: Using Generative AI to Accelerate Test Automation Development

How Structured AI Code Generation Transforms SenseTalk Handler Creation

SenseTalk is a powerful scripting language used across software and hardware development for test automation. It is part of the Eggplant Functional ecosystem, a platform known for automating testing through image recognition, text search, and system interaction.

Unlike traditional programming languages, SenseTalk focuses on behavior-driven testing, where scripts mimic how real users interact with interfaces. This makes it ideal for validating complex systems that include both software and hardware components.

At the core of every SenseTalk test is a handler — a reusable block of code that performs a specific action, such as clicking a button, entering data, or validating system responses. Handlers are the building blocks of automated test suites. They encapsulate logic, enforce structure, and ensure repeatable, consistent test behavior.

Creating handlers manually can be time-consuming. Each one must follow strict standards for naming, documentation, error handling, and structure. Variations in how handlers are written lead to inconsistency, rework, and reduced maintainability — especially when multiple engineers or contractors contribute to a project.

By introducing Generative AI into the development process, we can automate the creation of these handlers while maintaining compliance with coding standards. The AI generates ready-to-use, fully documented SenseTalk handlers that fit seamlessly into existing frameworks. This approach shortens development cycles, increases consistency, and ensures test assets remain usable long after their original creators have moved on.

Executive Overview — The Shift to AI-Augmented Development

Automation programming is changing fast. Generative AI is no longer a test tool or an experiment. It's becoming part of how we build and manage code. In our environment, AI helps write SenseTalk handlers that follow the same standards every time.

This approach gives us a clean way to produce code that is correct, documented, and reusable without constant human rework. The goal is not to replace engineers, but to make sure every engineer can create production-quality code faster and with fewer mistakes.

By enforcing structure through templates, AI gives us consistency that manual coding never could. The end result is faster output, fewer errors, and code that stays useful long after it's written.

The Challenge of Manual SenseTalk Development

Manual SenseTalk coding works well, but it's slow and inconsistent. Every engineer writes code a little differently, and that becomes a problem when multiple people contribute to the same project.

There are a few recurring pain points:

- Time spent on setup and documentation.
- Missed syntax or formatting details.
- Logic errors that repeat across scripts.
- Rework every time new engineers join.
- Lost knowledge when contractors finish their term.

Without clear standards, code turns into isolated efforts that don't scale. Over time, this slows down testing, increases risk, and makes it harder to keep projects aligned.

The AI-Driven Framework

The solution is structure. Every handler starts from a single AI-driven template that defines how the code should look and act. The engineer gives the intent — what needs to happen — and the AI builds the handler to match the standard.

Each generated handler includes:

- A complete comment header with author, purpose, and usage.
- Defined control flow that uses "if," "then," "else," and repeat loops the right way.
- Consistent logging for success, warning, and error messages.
- Parameterized inputs instead of hardcoded values.
- Clear error handling and exit conditions.

The code looks the same every time, no matter who creates it. It's easy to review, easy to reuse, and fits directly into existing frameworks.

The Advantage of an Offline AI Environment

In secure or regulated testing environments, external connectivity isn't an option. To maintain security while still using Generative AI, we deploy an Offline AI environment — an isolated model trained for structured code generation.

This Offline AI generates SenseTalk code that matches our standards even without internet access. Because it operates entirely within the local network, all data stays on-site with no external data flow. This makes it ideal for classified, enterprise, or government systems that require full control over inputs and outputs.

The Offline AI balances performance with predictability. It's not designed for general creativity — it's designed for consistency, compliance, and reliability. It understands logic, syntax, and the structure required for automation development, making it a trusted component of secure automation workflows.

Governance, Validation, and Continuous Improvement

AI output only works when it's managed. Each handler generation is tested, tracked, and reviewed. All versions are stored in source control with metadata for author, date, and purpose. This creates full traceability and makes it possible to audit code later.

Prompt templates that guide the AI are also version-controlled. When issues or improvements are found, the template is updated and revalidated. This feedback loop helps the AI perform better over time.

We test and validate AI-generated code before release, the same way we would human code. Syntax, function, and logic are verified in controlled environments. Regular validation checks also ensure that the model stays accurate and does not drift from the standard.

This process creates a closed-loop system: generate, validate, improve, repeat. The more we use it, the more accurate and reliable it becomes.

Collaboration and the Role of the Engineer

AI changes the job, not the need for it. The engineer defines what needs to be done. The AI builds the structure around that goal. This allows developers to focus on creative logic, test strategy, and system design instead of repeating setup and formatting tasks.

AI enforces consistency, but people still drive the logic and intent. Engineers review, refine, and integrate the output into larger workflows. This partnership reduces workload while improving the overall quality of automation.

It also helps with contractor transitions. Since all generated code follows the same format, a new engineer can pick up where another left off without confusion. Documentation and comments are built-in, so knowledge transfer happens automatically.

The Path Forward — Building a Sustainable Automation Ecosystem

The long-term vision is an ecosystem where AI supports every stage of automation — from code generation to documentation to test validation. Multiple specialized models can work together, each doing a part of the job: one builds handlers, another documents them, and another checks results.

This approach scales without increasing headcount or risk. It creates a continuous improvement cycle that keeps code clean, compliant, and ready for use.

Generative AI does not replace people. It raises the baseline. Every line of code produced through this framework meets the same high standard, whether it's written by a full-time engineer or a short-term contractor.

The combination of structure, governance, and AI assistance gives us faster delivery, better accuracy, and a repeatable process that stands the test of time.

References

1. SenseTalk Reference — Eggplant Functional Documentation, comprehensive reference for SenseTalk syntax and language use.
2. Objects and Messages — Eggplant Functional Documentation, explains handlers, objects, and message-passing concepts.
3. About SenseTalk — Eggplant Functional Documentation, provides an overview of the SenseTalk scripting language.
4. Google Engineering Practices Guide — covers code review, documentation, and commenting standards for maintainable code.
5. PEP 8 Style Guide for Python Code — widely recognized for coding style, readability, and documentation discipline.
6. NASA Software Engineering Handbook, Section 5.2 — outlines coding standards emphasizing clarity and maintainability.
7. NASA-HDBK-2203 Software Engineering Handbook — governance and traceability for software development.
8. IEEE Std 829-2008 — Software and System Test Documentation standard for structured and traceable testing documentation.
9. IEEE Std 828-2012 — Configuration Management in Systems and Software Engineering, covering consistency and revision control.
10. CERT SEI Secure Coding Best Practices — official SEI publication promoting secure and maintainable coding standards.
11. NIST AI Risk Management Framework (AI RMF 1.0) — provides guidance for trustworthy and responsible AI system management.
12. NIST AI RMF Playbook — companion resource for implementing AI governance and validation procedures.

For more information reach out to Technical Systems integrators, Inc.

www.tsieda.com

info@tsieda.com

1-407-339-4874 x102