LLMs and Software Testing

From Test Generation to Autonomous Vulnerability Discovery

IN3240/IN4240 Software Testing

ISTQB & GenAI

Spring 2026

LLMs for Writing Tests

How testers use LLMs today (ISTQB CT-GenAI; Certified Tester – Generative AI Testing):

1. **Test case generation** — from requirements, user stories, or API specs via structured prompts
2. **Test data synthesis** — realistic edge-case data, boundary values, equivalence partitions
3. **Test script drafting** — Cypress, Playwright, pytest scaffolding from natural language
4. **Defect prediction** — analysing code changes to prioritise where to test

Key ISTQB concept: The tester remains accountable for evaluating and validating all LLM-generated output.

Prompt Engineering Pattern

Role: You are a senior QA engineer.
Context: REST API endpoint `/users`
Instruction: Generate boundary value test cases for the age field (integer, 0–150).
Constraints: Include valid, invalid, and edge cases.
Output: Table with input, expected result, and test type.

Risk: Hallucination

LLMs may generate plausible but incorrect test cases. Always verify against the specification.

Testing LLM-Generated Code

Why LLM code needs extra scrutiny:

- **Non-determinism** — same prompt may yield different code each time
- **Subtle logic errors** — code compiles and "looks right" but fails on edge cases
- **Security blind spots** — models may produce code with injection flaws or hardcoded secrets
- **Outdated patterns** — training data may include deprecated APIs or libraries with known CVEs (Common Vulnerabilities and Exposures)

ISTQB principle: Testing must be adapted to the context — AI-generated code is a **new context** requiring adjusted test strategies.

Recommended Test Strategy

Technique	Focus
Code review	Logic, security, style
Unit tests	Boundary values
Static analysis	SAST tools (SonarQube)
Mutation testing	Test suite adequacy
Fuzzing	Unexpected inputs
Dependency scan	Known CVEs

Key Insight

The faster code is written (by humans *or* AI), the more important systematic testing becomes. Velocity without verification is technical debt.

From Software Testing to Vulnerability Discovery



Shared Foundations

Software testing and security testing use the same fundamental techniques: boundary value analysis finds buffer overflows, equivalence partitioning identifies input validation gaps, and control flow testing reveals authentication bypasses.

The AI Amplifier

LLMs can now apply these techniques at scale — reading millions of lines of code, reasoning about complex interactions, and identifying flaws that decades of human review missed.

AI-Driven Vulnerability Discovery: The State of the Art

Evolution of AI in security testing:

2014 Google Project Zero founded — elite human researchers finding zero-days

2023–24 LLMs begin assisting with code review and simple vulnerability detection

2025 Google's Big Sleep finds first real-world zero-day using an LLM agent

Apr 2026 Anthropic's Claude Mythos Preview autonomously discovers *thousands* of zero-days across major operating systems and browsers

The jump from "assists humans" to "autonomously finding and exploiting vulnerabilities" happened in under two years.

Reported Capability Jump

- **Project Zero:** elite human team, typically ∼20–30 zero-days per year
- **Claude Opus 4.6**: 2 successful exploit developments from Firefox JS-engine (version 147) vulnerabilities
- **Claude Mythos Preview:** 181 working exploits on the same benchmark
- **Interpretation:** not an apples-to-apples comparison, but a striking illustration of scale

Notable Mythos Findings

- 27-year-old remote code execution bug in OpenBSD
- 16-year-old flaw in FFmpeg
- Memory corruption in a *memory-safe* VM monitor

Project Glasswing(April 7, 2026): AI Discovery Meets the Security Ecosystem

What Project Glasswing is:

- Anthropic's restricted program for using Claude Mythos Preview in **defensive cybersecurity**
- Selected partners and 40+ additional critical-software organizations get access
- Anthropic provides up to **$100M in Mythos Preview usage credits** across these efforts
- Anthropic also provides **$4M total in direct donations** to open-source security organizations

Glasswing strengthens the *discovery* side of security work, but it does not replace the broader institutions needed for coordination, disclosure, and patching.

Important limitation

- Glasswing is **not the whole security ecosystem**
- In particular, it does **not** include:
  - **CERT/CC:** the long-standing coordination tradition for responsible disclosure
  - **Project Zero:** elite bug hunting plus strong disclosure pressure

Why this matters for testers

- Testing may increasingly include **AI-assisted vulnerability discovery**
- But findings still need **reproduction, prioritization, reporting, and coordination**
- So tester roles may shift from only *finding bugs* toward also *managing bug-discovery pipelines*

Key Takeaways and Discussion

What You Should Remember

1. LLMs are powerful **tools** for testers — but the tester owns the quality judgment
2. LLM-generated code requires **adapted test strategies** with extra focus on security and correctness
3. Software testing and security vulnerability research: **same foundational techniques**
4. AI has crossed a threshold where it can find vulnerabilities **faster and at greater scale** than human experts
5. The tester's role is evolving: understanding AI capabilities and limitations is now a **core competence**

ISTQB Certifications

CT-AI (2021) — Testing *of* AI-based systems: data quality, bias, non-determinism, ML metrics.

CT-GenAI (2025) — Testing *with* generative AI: prompt engineering, LLM evaluation, RAG, integration into testing practice.

Both build on Foundation Level.

Discussion Questions

- Should AI-generated code be labelled?
- Who is liable when an AI-written test misses a critical bug?
- Is the tester's role expanding or shrinking?

Test Driven Development (TDD) vs prompt generation

OsloMet · 2026

LLM-Based Test Case Generation

Evaluating TDD Approaches with Large Language Models



Non-TDD | Batch-TDD | Iterative-TDD

TDD x LLM

Three Pipelines on the 164 HumanEval Tasks

What was evaluated?

- We evaluate the three prompting strategies only on the **164 HumanEval programming tasks**.
- HumanEval is a benchmark of **Python coding problems**, where a model must generate a correct function that passes the task's hidden official tests.
- **HumanEval** is used as a correctness metric: *percentage of problems with all test cases passed.*

Iterative = enforced TDD

- First, the LLM writes **one test** for the task
- The system runs that test; it should initially **fail**
- The LLM is then asked to write or revise code so that **this test passes**
- Next, the LLM writes **another test**, and the loop repeats

Batch = all tests first

- The LLM first writes the **entire test suite** for the task
- Only after that does it write the implementation
- The system runs the full suite against the generated code
- If tests fail, the LLM gets feedback and may revise the solution

NoTDD = direct prompt

- The LLM is asked directly for a solution
- No tests are generated first
- No test-first feedback loop is enforced
- This is the baseline: just prompt for code and evaluate the result afterward

HumanEval Results Across Models

Model	Pipeline	pass@1	Passed	Total
Qwen3:8b	iterative	0.884	145	164
	batch	0.872	143	164
	notdd	0.842	138	164
gpt-oss:20b	iterative	0.945	155	164
	batch	0.951	156	164
	notdd	0.933	153	164
Gemma3:12b	iterative	0.842	138	164
	batch	0.811	133	164
	notdd	0.835	137	164
Gemma3:27b	iterative	0.878	144	164
	batch	0.848	139	164
	notdd	0.860	141	164