TNG TECHNOLOGY CONSULTING

Rise of the AI Testers

Generate unit-tests with AI agents

Dr. Marie Bieth
Dr. Michael Oberparleiter

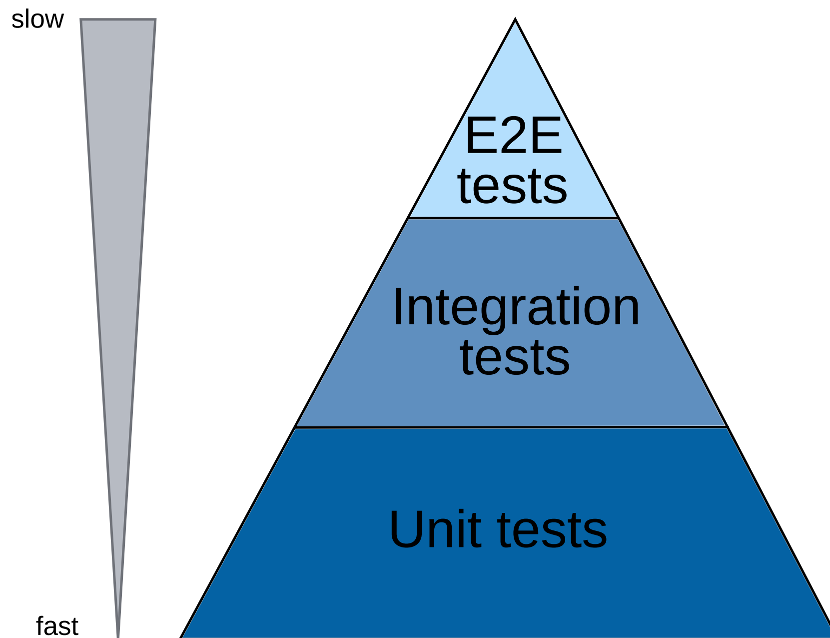
26/09/2025

Legacy code without tests

TNG TECHNOLOGY CONSULTING

Why Unit Tests?

TNG TECHNOLOGY CONSULTING



1 | Agents for unit tests generation

The simple approach

Providing an LLM with the test class and instructions to write tests.

```
public class AccountServiceImplTest {
    private User user;
    private Account account;

    @Test
    void testCreateAccount() {
      when(accountRepository.save(any(Account.class))).thenReturn(account);
      Account createdAccount = accountService.createAccount(user);

      //not testing the logic inside createAccount, only the mock
      assertEquals(account.getAccountNumber(), createdAccount.getAccountNumber());
      assertEquals(account.getBalance(), createdAccount.getBalance());
      assertEquals(account.getUser(), createdAccount.getUser());
    }
}
```

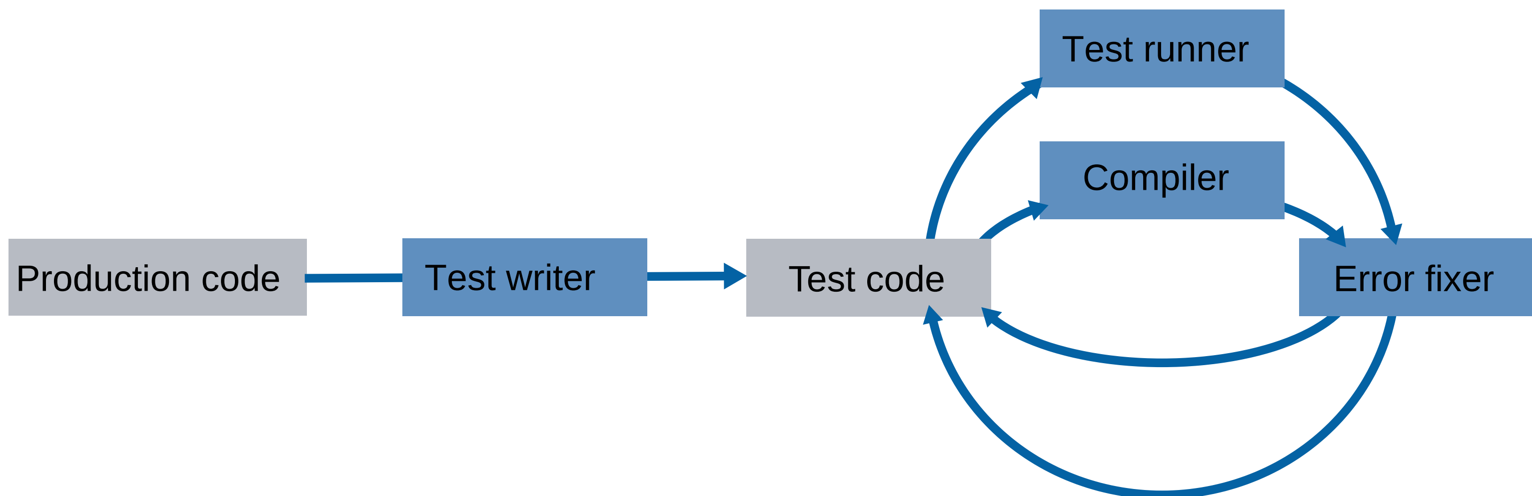
The simple approach

To write good tests, you need to:

- Understand the code
- Generate test scenarios
- Generate tests
- Ensure that they compile and pass

The agentic approach

TNG TECHNOLOGY CONSULTING



Unit Test Writer

Responsible for initially writing the tests, in multiple steps:

- Describe what each public method does
- Generate test scenarios for each public method
- Write test code for each scenario
- Merge tests into a single class (including mocks, setup and teardown)

Compiler and test runner

- No LLMs here, just shell scripts and log files
- Recognize the tooling used to compile and run the project

```
Tests
✓ UserService.test.js                  8 paıssed   1 faiel
✓ ✓ should create a new user           1 skipped   1 fotal
✓ AuthController.test.py
✓ ✓ should authenticate user
⚠ ✓ should handle missing cre dentials
✓ Payment.test.rb
⚠ ✓ should rrocess payment                         2 total
⚠ ✓ should refund payment
✓ ProductAPI.test.go
✓ ✓ should list produts                            2 total
✕ ✓ should create a produt iẋ                     10 total

4 passed    8 passd    1 skipped   10 total       3.25s
1 failed    8 skipped   1 falel                    92%
```

Debugging with the error solver

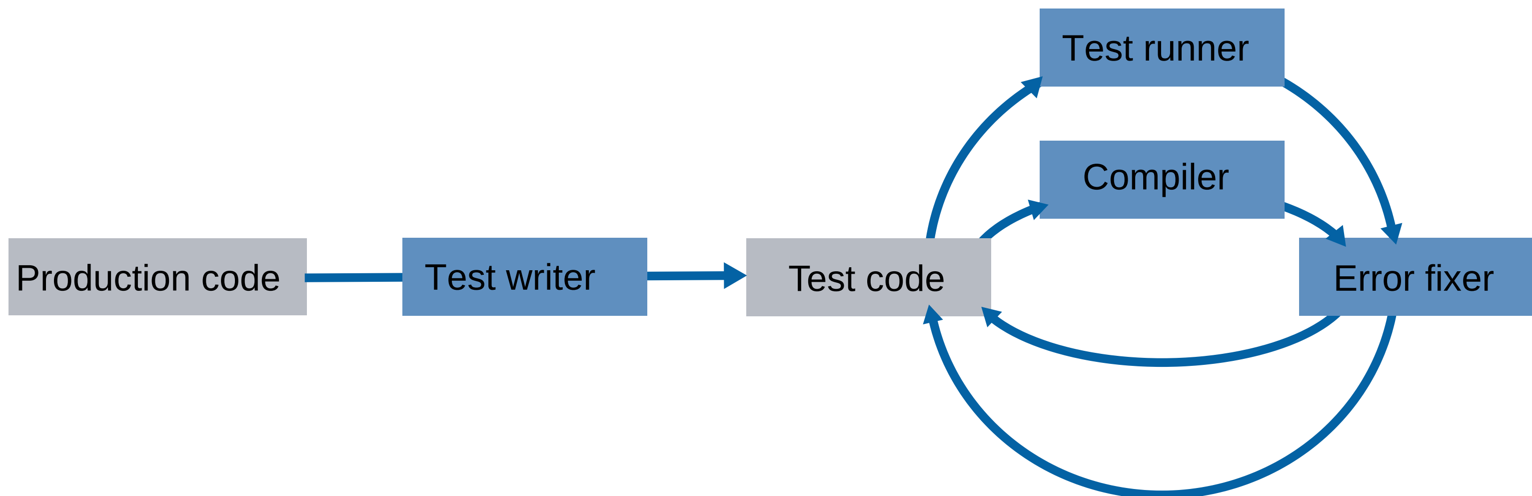
Responsible for fixing compiling and test errors:

- receives the problematic test class
- receives error from compilation or test logs
- explains the error (cause, location, solution strategy)
- generates a patch to solve error

Patches avoid unintended modifications to the rest of the class

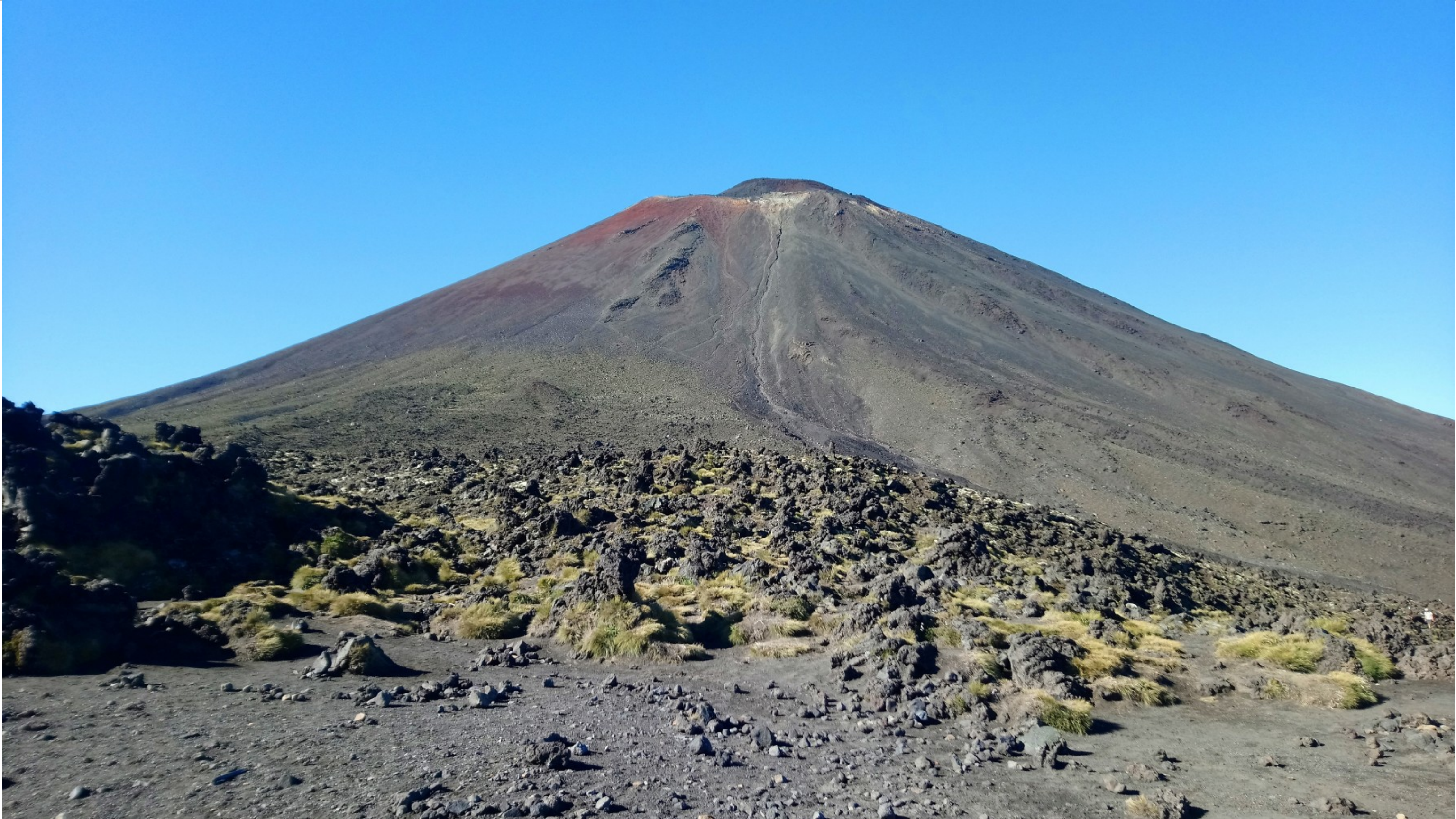
The agentic approach

TNG TECHNOLOGY CONSULTING



We need context...

TNG TECHNOLOGY CONSULTING



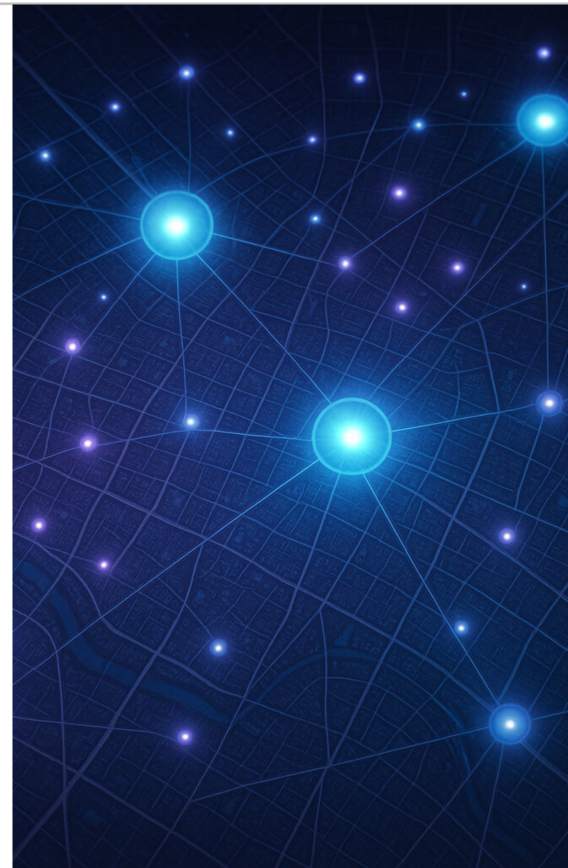
...so does an LLM

The code of the class to test is not self-contained and we need at least the signature of dependencies.

- Brute-force: Just pass the entire repo in the prompt. This can work, but:
  - Hard token limits, token costs, long response times
  - LLM gets confused by irrelevant information
- Other extreme: Turn the repo into a graph, put it in a database, do retrieval-augmented generation
  - Complex setup, but suitable for large repos and complex tasks
  - Overkill for unit tests
- Sweet spot: Repository map

What is a repository map?

- Concept and code based on https://github.com/Aider-AI/aider with underlying tree-sitter library
- Condense the repo into a map of classes and functions.
- Type and call signature are needed for correct mocks.
- We still do not want to send the entire map:
  - Graph-ranking algorithm finds the important parts of the code base.
  - Token budget limits what is deemed still relevant.



2 | Results

What is a good test?

TNG TECHNOLOGY CONSULTING

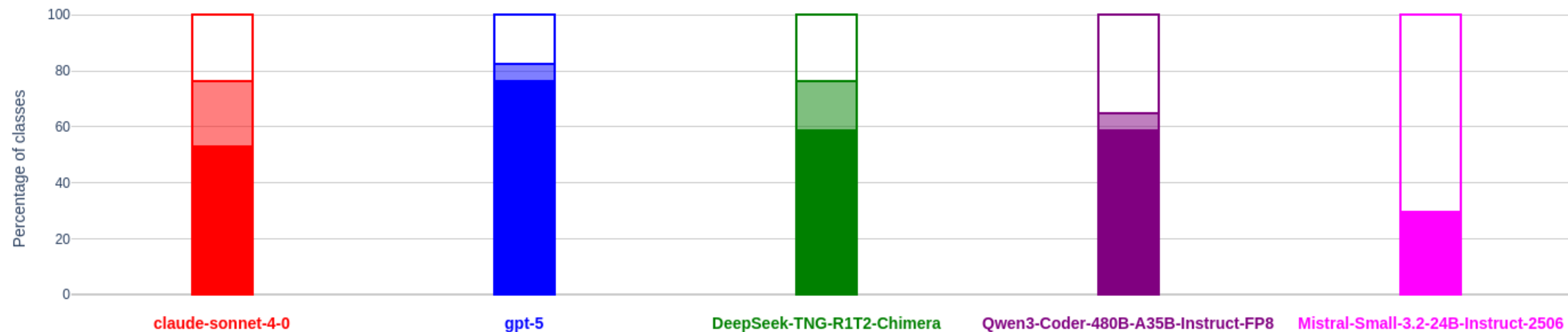
- Code coverage: how many lines of code are run by tests
- Mutation score: how many code changes can the tests catch

```
public class AccountServiceImplTest {
    private User user;
    private Account account;

    @Test
    void testCreateAccount() {
      when(accountRepository.save(any(Account.class))).thenReturn(account);
      Account createdAccount = accountService.createAccount(user);

      //not testing the logic inside createAccount, only the mock
      assertEquals(account.getAccountNumber(), createdAccount.getAccountNumber());
      assertEquals(account.getBalance(), createdAccount.getBalance());
      assertEquals(account.getUser(), createdAccount.getUser());
    }
}
```

Benchmarking: Open weight vs commercial LLMs

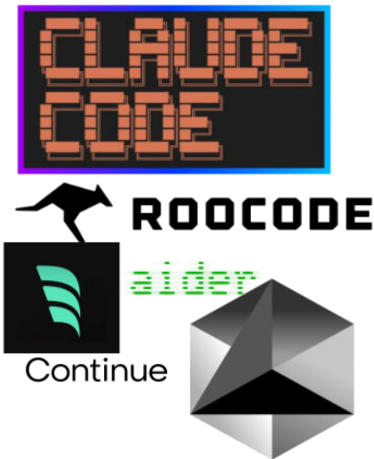
TNG TECHNOLOGY CONSULTING

- All state-of-the-art commercial LLMs perform very well, open-weight LLMs are still suitable
- Set of 17 classes from 5 open-source repos, ranging from Spring-Boot examples to thread benchmarking
- Success (solid fill): mutation coverage > 80%, failure (no fill): 0% coverage
- Larger, newer models perform better. But even small ones will do a decent job.



Can't I just ask Copilot/Roo/Claude for tests?

- Especially agentic assistants are suited: feedback loops are important.
- With a top-of-the-line LLM in the background you get high quality tests.
- General purpose vs specialized tool:
  - Handholding during run needed (2h vs 3min in extreme case).
  - Will modify the tested code to make tests work sometimes.
  - Isolation of generated code is your own problem.
- Large codebases exacerbate these differences:
  - Flexibility becomes a curse.
  - Unclean environments lead to confusion.



Challenges and next steps

- We started this project end of 2024 and got nice results, but the LLM tool landscape evolved rapidly in 9 months.
- Who knows what will be there in another 6?
- Current limitations in our approach:
  - The productive code needs to be testable
  - Tests are generated for the **current** behavior of the productive code
  - There is such a thing as "too many tests"
- Next steps:
  - Separate approach for E2E tests
  - Keep improving user experience
  - Test-driven development

TNG TECHNOLOGY CONSULTING

Thank you for your attention

Any questions?

Dr. Marie Bieth
Principal Consultant
marie.bieth@tngtech.com

Dr. Michael Oberparleiter
Senior Consultant
michael.oberparleiter@tngtech.com