7. Development & Testing

Summary

- Embed AI-assisted development in GitHub-centric workflows with mandatory human reviews, infrastructure-as-code, and clear documentation intent.
- Apply a full testing spectrum—unit, integration, functional, performance, and chaos/fault tests—to validate every code change.
- Implement transparent observability via OpenTelemetry, Sentry and cost-aware metrics+logging, plus living docs (Confluence, README, ADRs).

Pending Decisions

- Define governance and guardrails for AI-generated code (tools, review process, approval criteria).
- Select standard test frameworks and coverage/quality thresholds for unit, integration, and E2E tests.
- Configure the observability stack, sampling rates, and retention policies.
- Establish the branch/PR policies.

Open Items

- Define performance/load testing tools and baseline SLIs/SLOs.

Overview, core tenets

code development = application code + test code + infrastructure-as-code + configuration-as-code

1. Every code development ideally precedes some form of an intent in a recorded written format (i.e. documentation). At least an AI prompt. One step better is a well written feature request ticket. Even better a proper design document, in which business and architecture is in symbiosis.
2. Every code development starts on GitHub
3. Every code development can and encouraged to be enhanced or completely driven by AI
4. Every code development is reviewed by human eyes
5. Every code development also means the inclusion of one or more ways of testing the results
6. Every code development brings with it ways of monitoring its state, its performance, its errors
7. Every code development keeps track of its dependencies, especially open-source components

Develop

Like its AI or never. With the important note to review code carefully, especially one generated by monkeys optimistically typing on keyboards (i.e. generative AI)

Test

(Almost) Everything. Especially AI code. It will be better for everyone. Use AI to keep AI on the right tracks.

Depending on what is being written, consider one or more ways to validate that it is correct. It can also count as documentation, as knowledge base. TDD-it.

- When writing pure logic, code for its own sake, DDD, etc.. : **unit** test it
- When writing anything that interacts with something external to those nice sentences that you have just written down: **integration** test it
- When a user functionality nicely takes its first proper shape: **functionally** test it
- When more users want to enjoy the nice functionalities: **load** and **stress** test it
- When the universe has a glitch: **fault** test it

Monitor

Do not create a magical blackbox. Ensure transparency, let it be monitorable so others can help.

- Push to **Sentry**
- Expose traces, metrics and logs as **OpenTelemetry** compatible records. Push them or let them be pulled by external tools
- And at all times consider that this all costs. It costs resources (e.g. CPU from the source), it costs money, etc.. so do it wisely

Documentation

See 1st tenet. Unless someone comes with a better idea, let's stick to Confluence & README's in the respective GitHub repo, for the source code, for documentation (apart from the exceptional cases mentioned). Always consider adding visuals too. Diagrams in various forms, wire-frames, screenshots, videos, ticket-links (.e.g Jira), architecture decision records (Architecture decision record - Microsoft Azure Well-Architected Framework) etc..