QUALITY ENGINEERING & DEVELOPER EXPERIENCE

# Complex work, made simple.

I turn product and engineering needs into software, testing platforms and tools that teams can use with confidence.

My career connects business analysis, software development and quality engineering. I approach a need through the people who use a system, the processes around it, its technical constraints and the evidence needed to validate the result. That combined view helps me frame the problem, build a response and equip teams to carry it forward.

## FROM NEEDS TO SYSTEMS

- **Connect quality with business needs**
  Fnac, Manpower, AFNOR, EDF (nuclear division) / Before 2014
- **Understand the context**
  OPCAIM / 2014-2016
- **Build the response**
  BearingPoint & BNP Paribas Cardif / 2017-2021
- **Make it testable**
  Agorapulse / 2021-2022
- **Read delivery signals**
  Deezer / 2022
- **Validate under constraints**
  HarfangLab / 2023
- **Equip the teams**
  Believe / 2023-2025
- **Formalise the knowledge**
  Independent R&D / 2026 - Ongoing

## IN THIS PORTFOLIO

**QaaS** 2
From business scenario to execution

**Products & DevEx** 3
beOne, interfaces and local assistance

**Quality & performance** 4
Signals that support decisions

**QA Hub** 5
Define the system's guarantees

**Agent Assurance** 6
Outcomes, paths and evidence

## GET IN TOUCH

**lydia.arad@proton.me**

Work at Believe (2023-2025) and independent research. Follow the links in this portfolio to explore each full case study.

01 / QUALITY PLATFORM

# QaaS
# Keep the scenario in focus.

I designed and developed the shared test foundation and integrations for QaaS, working with engineers from the contributing teams. The goal: make test authoring, execution and maintenance more accessible to teams.

**Business scenario**
Fixtures, commands and factories

→

**Shared access**
API clients, configuration and identity

→

**Execution**
Launchers and runtimes

→

**Feedback**
Consoles, reports and metrics

### APIs / pytest

Fixtures prepare clients and data; shared assertions express the checks. Public interfaces, metadata, decorators, registries and plugins organise test declaration and selection.

### Web / Cypress

Business commands combine session setup and recurring workflows. Selectors, waits and technical status mappings are maintained in the shared foundation.

### Performance / k6

Factories assemble operations, load profiles, checks and metrics. Shared HTTP and instrumentation layers keep calls and measurement consistent.

**2 weeks → 2h**
Getting started with QaaS

**80 → 21**
Lines in a pytest scenario

**13 → 2**
Cypress calls in a business example

What the figures show: the onboarding gain covers QA engineers and developers getting started with QaaS. Code reductions illustrate individual scenarios; preparation and recurring checks live in the shared foundation.

### Prepare service access once

OpenAPI contracts referenced in Backstage, the internal service catalogue, feed Python client generation. A shared layer prepares their configuration and authentication; Vault and OIDC integrations prepare reusable access patterns.

### Prepared execution environments

Docker and Compose supply versioned runtimes, development and debugging profiles, caches and persistent results. Launchers prepare runs; k6 tooling generates Kubernetes Jobs and follows status, logs and cleanup. Adaptations cover macOS, Linux and Windows through WSL2.

Authors assemble business capabilities while recurring mechanics stay maintainable in a shared foundation. A common request format and scenario catalogue are the next platform direction across the engines.

Architecture and examples  Illustrated gains

02 / PRODUCTS AND DEVELOPER EXPERIENCE

# Build the product. Equip the team.

Understand how developers, QA engineers and Product Owners work, then bring preparation, scenario authoring, execution and result interpretation into a usable workflow.

### beOne / from idea to MVP

Product scoping, UX/UI and React/TypeScript development of an internal onboarding portal. Contribution checks, versioning and adapted GitLab pipelines support previews and delivery to integration through Argo CD on GCP. The deployed MVP gives the team a product to use and evaluate.

### Interface Mapper / automation groundwork

An MCP server with Playwright inspects a page, extracts elements and candidate selectors, and checks uniqueness. The map supports Cypress command preparation. A separate browser add-on analyses the DOM to generate tests and commands; a Faker extension prepares test data.

MAKE RESULTS READABLE

Structured pytest output with Loguru, readable API responses and per-test status and duration; a Cypress formatter separating execution phases with four verbosity levels, captures and videos; a VS Code API Tester with Swagger selection, request fields and history. The interfaces connect the test context with useful diagnostics.

## Eve / a natural-language entry point to QaaS knowledge

I designed and developed a local documentation RAG assistant: extraction of documentation, docstrings and comments, semantic retrieval and generation with source references. It helps engineers find a convention, understand a command and follow the explanation back to its source file.

**Extract**
Passages and source paths

→

**Index**
MiniLM and FAISS

→

**Retrieve**
Question-related context

→

**Answer**
llama.cpp and references

A Flask service, index and model runtime are packaged with QaaS; Phi-2 and Mistral provide local model options. Inference runs locally after installation.

Next steps: evaluate answer quality, reference accuracy and index freshness. Preparing and launching campaigns through a common orchestrator is an extension being explored.

beOne and Interface Mapper  Interfaces and consoles  Eve in detail

03 / QUALITY AND PERFORMANCE

# Connect results with decisions.

At Believe, I connected k6 results, Datadog observations and business risks with Go/No-Go recommendations. Findings were presented to leads and VPs for rollout decisions by business unit, including work around the GCP migration.

## AN ANONYMISED SYNCHRONISATION CAMPAIGN

The campaign followed submission, synchronisation, asset upload, transfer, encoding and the draining of accumulated work. A successful upload was tracked through to downstream availability.

### Volume

Release-count target reached

### Delay

Downstream availability still too slow

### Action

Prioritise bottlenecks before wider rollout

### Observe the full chain

Transfer and encoding delays kept growing as accumulated work drained. Analysis connected business delays, errors, behaviour under load and threshold compliance with Datadog signals: browser experience, CPU and pod behaviour.

### Turn findings into priorities

Recommendations focused on the synchronisation backlog, downstream throughput, transfer and encoding. Clearer progress feedback would also help users understand the state of a release.

## READ SIGNALS AT THE RIGHT LEVEL

In a k6 reporting example, the global response-time threshold is met while an individual operation exceeds its stricter threshold. Functional checks provide another signal about the expected data and behaviour.

| Scope | Reading the result |
| --- | --- |
| Global response times | Overall threshold met |
| Individual operation | Its operation-specific threshold exceeded |
| Functional checks | Compare received data, expectations and test assumptions |

The approach connects the workload profile, observed behaviour and infrastructure. It makes release-readiness criteria, priority corrections and follow-up checks explicit.

Approach and campaign  k6 reporting example

04 / INDEPENDENT R&D

# QA Hub
# Define the right guarantees.

A quality decision-support platform to define what a system must guarantee, then prioritise its tests. It connects domain expertise, product constraints and verification methods.

**Working product POC / Engine architecture and composition rules specified**

**Confirm**
Context and assumptions

→

**Compose**
Versioned quality packs

→

**Prioritise**
Risks, properties and tests

→

**Explain**
References and decisions

### A model fitted to the system

In the defined architecture, the Quality Model Engine selects Quality Model Packs and composes them through deterministic rules. Outputs connect risks, properties, metrics, strategies, tools and artefacts. Each pack exposes its knowledge and coverage.

### Inspectable decisions

The Quality Evidence Layer connects context, pack versions, references, assumptions, conflicts and expert choices in a decision record. Experts can adjust priorities with traceability. AI supports interpretation and generation; the decision core retains explicit rules.

EXAMPLE / MOBILE MUSIC RECOMMENDATIONS

A service that recommends tracks and starts playback combines several expectations: recommendation relevance, listening continuity, network interruptions and a response budget. The model must surface their interactions: a relevant method may demand a cost incompatible with the mobile budget.

The proposed strategy connects relevance criteria, datasets or human judgements, listening journeys and observations under load. Conflicts and partially supported dimensions stay attached to the reasoning for expert arbitration.

### Available / product experience

A four-question intake and five sectors contextualise strategies and exportable artefacts. The POC uses parameterised templates and simulated agent activity to demonstrate the user journey.

### Next milestones / engine and evidence

Build the Evidence Layer, implement the pack schema and deterministic engine, then validate decisions and provenance on reference cases. The longer-term vision connects applications, DOM, APIs, documentation, Jira, Confluence, tests and traces to map journeys and their evidence.

Product, architecture and full example

05 / INDEPENDENT R&D

# Agent Assurance
# Outcomes, paths and evidence.

Justify trust in an agent's work: verify its result and the boundaries respected throughout execution. The priority hypothesis studies capabilities created by the composition of individually permitted operations.

**Logical architecture and protocol defined / Next: a controlled MCP experiment**

### Work Assurance / the result

Assess delivered work against functional requirements, regression tests, security, performance and business expectations. The task type determines appropriate evidence and verifiers.

### Path Assurance / the path

Connect operations, identities, delegations, data flows and shared resources. Locally permitted operations can combine into a capability prohibited by the task contract. MCP is the first experimental surface.

## RESEARCH SCENARIO / ONE RESOURCE ACROSS TWO RUNS

Run A may write an artefact; Run B may read from that store. Local permissions permit both operations while task policy requires the runs to remain isolated. The experiment needs to retain the resource's identity and provenance across the two runs.

**A writes**
Artefact and origin identified

→

**B requests a read**
Resource linked to A

→

**Assess the path**
Contract, provenance and policy

Expected decision: deny the read before its effect when the prohibited cross-run channel is established. A sharing workflow explicitly authorised by the contract provides a comparison case.

## EXPRESS CERTAINTY AND ACTION SEPARATELY

### Assurance state

VERIFIED: sufficient observations and integrity within the declared boundary.
INCONCLUSIVE: available evidence is insufficient for a conclusion.
DEGRADED: an observation or control surface is weakened.

### Execution decision

ALLOW: execute after the applicable checks.
DENY: stop before the effect.
REQUIRE_APPROVAL: ask an authorised person to decide.

The protocol and its 14 reference scenarios define a comparison of individual permissions with contextual assessment: false positives, missed violations, latency, explanation quality and incomplete observations. The controlled MCP experiment is intended to establish feasibility and practical decision value.

Hypothesis, architecture and protocol