# An Agentic Approach to Model-Based Testing for Auditable Test Generation

Frédéric Tamagnan – Smartesting

## Identification of Use Cases

On embedded projects, the engineers responsible for verification and validation (V&V) design and maintain test campaigns. The volume and scope of those test campaigns increase at each new version. Current practices can no longer keep up. Test design is based on system engineering artefacts and accounts for a significant proportion of this effort. These artefacts are heterogeneous: their level of detail varies, they are spread across several documents and describe variants of the system. Establishing the completeness of test coverage remains a manual and iterative task, which leads to rework. Identifying every test condition in the system engineering artefacts is therefore difficult to automate using simple search and regex rules. One immediate solution is to ask a language model to generate test scenarios directly from the specifications[1, 2]. The effort is not reduced, it moves from generation to review. The cost of that review grows with the volume of tests produced, while that volume is what generative assistance increases. The net benefit to the V&V cycle is therefore not guaranteed. Generative AI assistance that can be used in this context must meet two conditions: it must produce a result that the engineer can verify at a reasonable cost, and it must enable the coverage achieved to be justified.

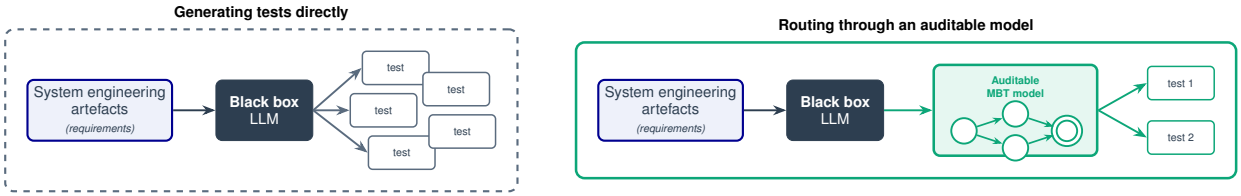


*Figure 1 – On the left, the engineer must review every test. On the right, the engineer reviews one model.*

The work relies on two case studies, both modelled in Arcadia/Capella. The first, PythaDrone, is an open-source case study describing an inspection drone. The second is a real embedded defence software at Thales, currently in service and already qualified. It contains about 600 system-level requirements and 5,000 subsystem-level requirements, spread across four functional increments. PythaDrone represents 1 to 10 per cent of that volume. The two cases complement each other, with the first enabling development on a smaller corpus and the second demonstrating scalability.

## Proposed Technical Approach

The methodology draws inspiration from research that requires the language model to generate a structured test specification [3]. Here, the language model does not generate the tests, it assists the engineer in constructing the test model. The Model-Based Testing tool then derives the test scenarios from the validated model, according to explicit coverage criteria. Coverage is guaranteed by construction, and the engineer's review focuses on the model rather than on the generated tests. The work builds on Yest, a Model-Based Testing tool already deployed at Thales, to which a conversational assistant has been added. That assistant analyses the request and delegates it to one of two specialised sub-agents. The Yest sub-agent drives the MBT model through 35 typed tools that wrap the Yest API, one per model operation such as create a workflow or add a task. The Text2GraphQL sub-agent translates the question into a GraphQL query, runs it against the Arcadia knowledge graph and returns the relevant system engineering artefacts. The engineer reviews the proposed model, corrects it, and then validates it. The validated model serves as the single point of reference for test generation.

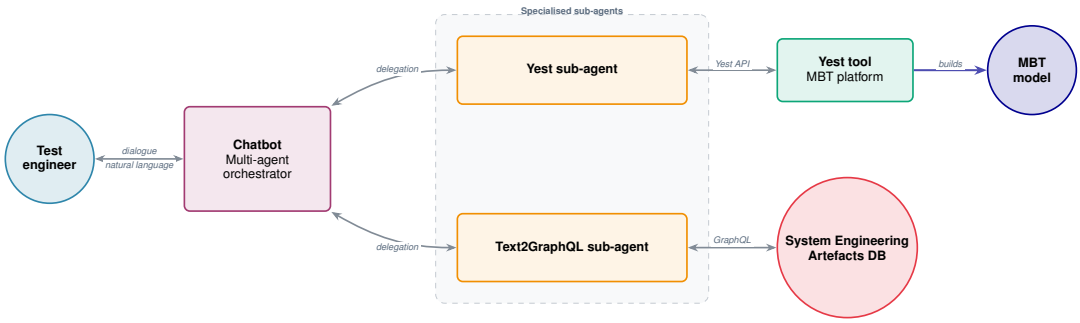


*Figure 2 – The engineer drives the assistant in natural language; the sub-agents reach the MBT tool and the requirements repository.*

The two sub-agents carry different risks. The first can produce a non-relevant or invalid model. The second can write executable but wrong queries. Two distinct protocols therefore evaluate them separately.

**Evaluating the MBT sub-agent.** The evaluation covers 18 LLM-generated MBT use cases. Each case carries its own assertions, and each falls into one of three levels: *easy*, *medium* or *hard*.

Three checks examine each produced model. Integrity verifies that the model follows standard MBT good practice. Assertions verify the properties that the use case expects. Judge is an LLM-as-judge assessment, which reads the *diff* between the model before and after the agent's run, rather than the agent's own summary.

| | *easy* (8 cases) | | | *medium* (5) | | | *hard* (5) | | |
|---|---|---|---|---|---|---|---|---|---|
| **Model** | Integ. | Assert. | Judge | Integ. | Assert. | Judge | Integ. | Assert. | Judge |
| Claude Haiku 4.5 | **75** | **75** | **62** | **60** | **80** | **80** | **100** | **100** | **80** |
| Claude Sonnet 4.5 | **75** | 50 | 50 | **60** | **80** | **80** | 60 | **100** | 60 |
| Mistral Medium 3-5 | **75** | 62 | 50 | **60** | **80** | **80** | 40 | 80 | 25 |
| Mistral Small | **75** | 50 | 50 | **60** | 60 | **80** | 60 | 80 | 20 |

*Table 1 – Yest MBT sub-agent: the three checks (%) per difficulty level, 18 cases × 4 models.*

The three checks do not yet agree with one another, so the right set of integrity rules, per-case assertions and judge prompt still has to be found. The Anthropic models also perform better than the ones currently available at Thales, which leaves room to improve the process on the models actually deployed there.

**Evaluating the Text2GraphQL sub-agent.** The evaluation uses a dataset of questions to put to the agent, each with an expected answer. Each question falls into one of two levels, *easy* or *hard*, and an LLM-as-judge scores the relevance of the answer in $[0, 1]$.

The evaluation compares three configurations of the sub-agent. In *draft*, the language model writes the GraphQL query itself, from the question and the schema alone. In *+ template*, the agent can call pre-defined GraphQL templates that cover the frequently asked questions. In *+ code agent*, the agent can process the data with a Python script it generates, for advanced computation that the query language cannot express.

| | | *draft* | | *+ template* | | *+ code agent* | |
|---|---|---|---|---|---|---|---|
| **System** | **Model** | Easy | Hard | Easy | Hard | Easy | Hard |
| PythaDrone | Mistral Medium 2508 | 0.67 | 0.47 | 0.82 | 0.59 | 0.85 | 0.67 |
| | Mistral Small 2603 | 0.63 | 0.59 | 0.88 | 0.79 | 0.88 | 0.66 |
| | Mistral Medium 3-5 | **0.70** | 0.67 | **0.90** | 0.69 | 0.88 | 0.72 |
| | Claude Sonnet 4.6 | **0.70** | **0.74** | **0.90** | **0.88** | **0.92** | **0.86** |
| Real industrial | Mistral Medium 2508 | – | – | 0.62 | 0.31 | 0.68 | 0.23 |
| | Mistral Small 2603 | – | – | **0.87** | **0.52** | – | – |

*Table 2 – Text2GraphQL relevance: PythaDrone, and the same protocol on real industrial data.*

Adding the templates was worthwhile, as they help every model. The code agent is a further gain, but only for the models capable of using it. Mistral Medium 2508 was the first model available at Thales when the first runs were made, and it does not perform well on the real industrial data. Mistral Small 2603, recently available through their stack, reaches 0.87 on the easy questions and 0.52 on the hard ones. Recent models therefore give reasonable results on industrial data.

**Deployment.** This work is carried out within the RAPID VVAMIA project (TRL 5). The system has been deployed to three V&V engineers at Thales, who are now using it on their own projects. Their feedback is awaited and will guide the next iteration.

**Future work.** The language model should stop paraphrasing the source data. The agent should instead generate Jinja templates that display the source data directly, so the engineer reads the artefact itself rather than a summary of it. Evaluating the pipeline end-to-end, from the system engineering artefacts to the generated model, also remains to be done.

## References


[1] Margarida Ferreira, Luís Viegas, João Pascoal Faria, and Bruno Lima. Acceptance test generation with large language models: An industrial case study. In *2025 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 1–11. IEEE, April 2025.

[2] Satoshi Masuda, Satoshi Kouzawa, Kyousuke Sezai, Hidetoshi Suhara, Yasuaki Hiruta, and Kunihiro Kudou. Generating high-level test cases from requirements using llm: An industry study, 2025.

[3] Dragan Milchevski, Gordon Frank, Anna Hätty, Bingqing Wang, Xiaowei Zhou, and Zhe Feng. Multi-step generation of test specifications using large language models for system-level requirements. In Georg Rehm and Yunyao Li, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 132–146, Vienna, Austria, July 2025. Association for Computational Linguistics.