MSc Computer Science Summer Projects 2025

LLM-Powered Anti-Pattern Resolution Project

Exploring AI Proof of Concepts with UCL and IBM

Client: Amrin Maria & John McNamara, IBM

Andrei Constantin, andrei.constantin.24@ucl.ac.uk
Abhaya Ghimire, abhaya.ghimire.24@ucl.ac.uk
Avinash Mallick, avinash.mallick.24@ucl.ac.uk
Gavriel Neuman, gavriel.neuman.24@ucl.ac.uk
Maoqin Lei, maoqin.lei.24@ucl.ac.uk
Vamsi Mukkamala, vamsi.mukkamala.24@ucl.ac.uk

Who are we?

- Master’s students at UCL in Software Systems Engineering.
- Collaborating with IBM representatives Dr. Amrin Maria and John McNamara.
- Working under the tutelage of Dr. Jens Krinke, our supervisor at UCL.

What are we building?

- An automated tool to help developers detect and fix anti-patterns, with a focus on legacy codebases.
- Harnessing the power of Large Language Models (LLMs) like IBM Granite.
- Using state-of-the art agentic approaches both in the cloud (IBM WatsonX) and locally, on personal devices.
- Designed with sustainability in mind, minimising ecological cost wherever possible.

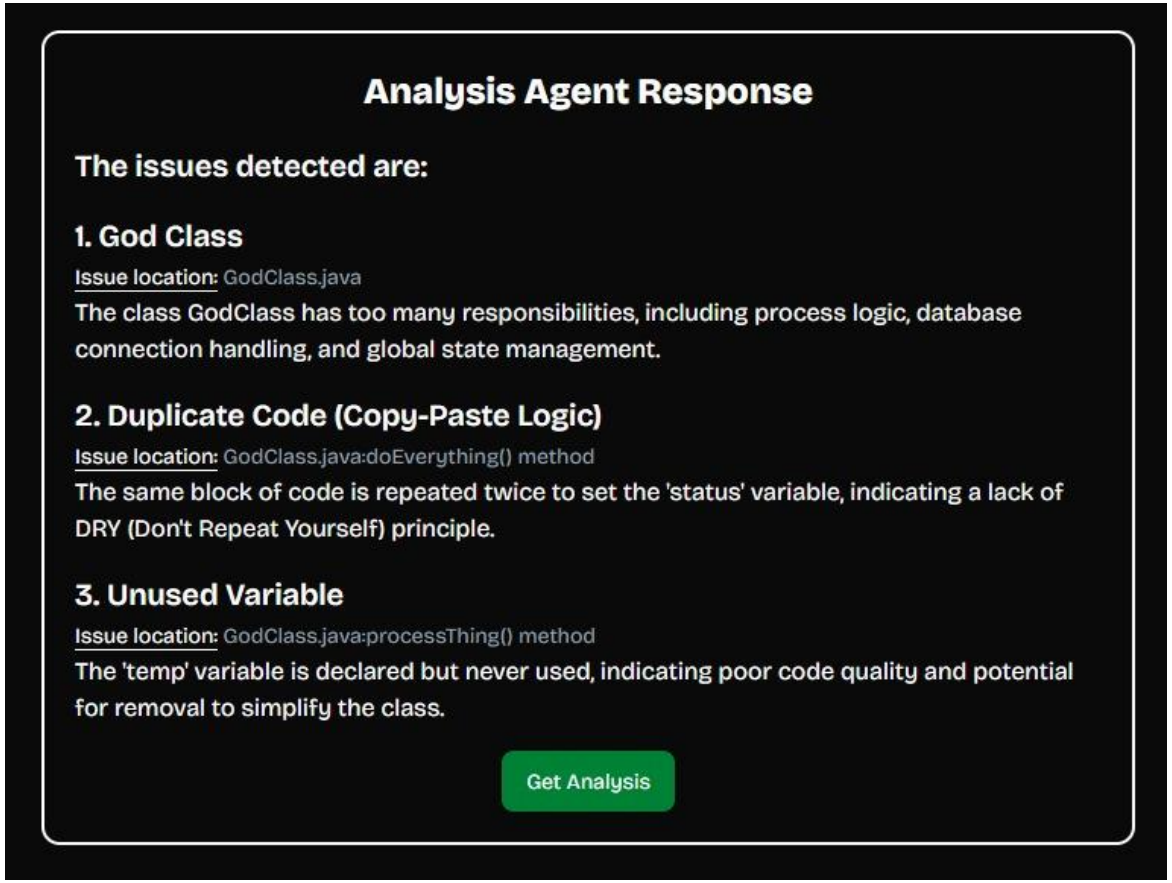
Why are we building this tool?

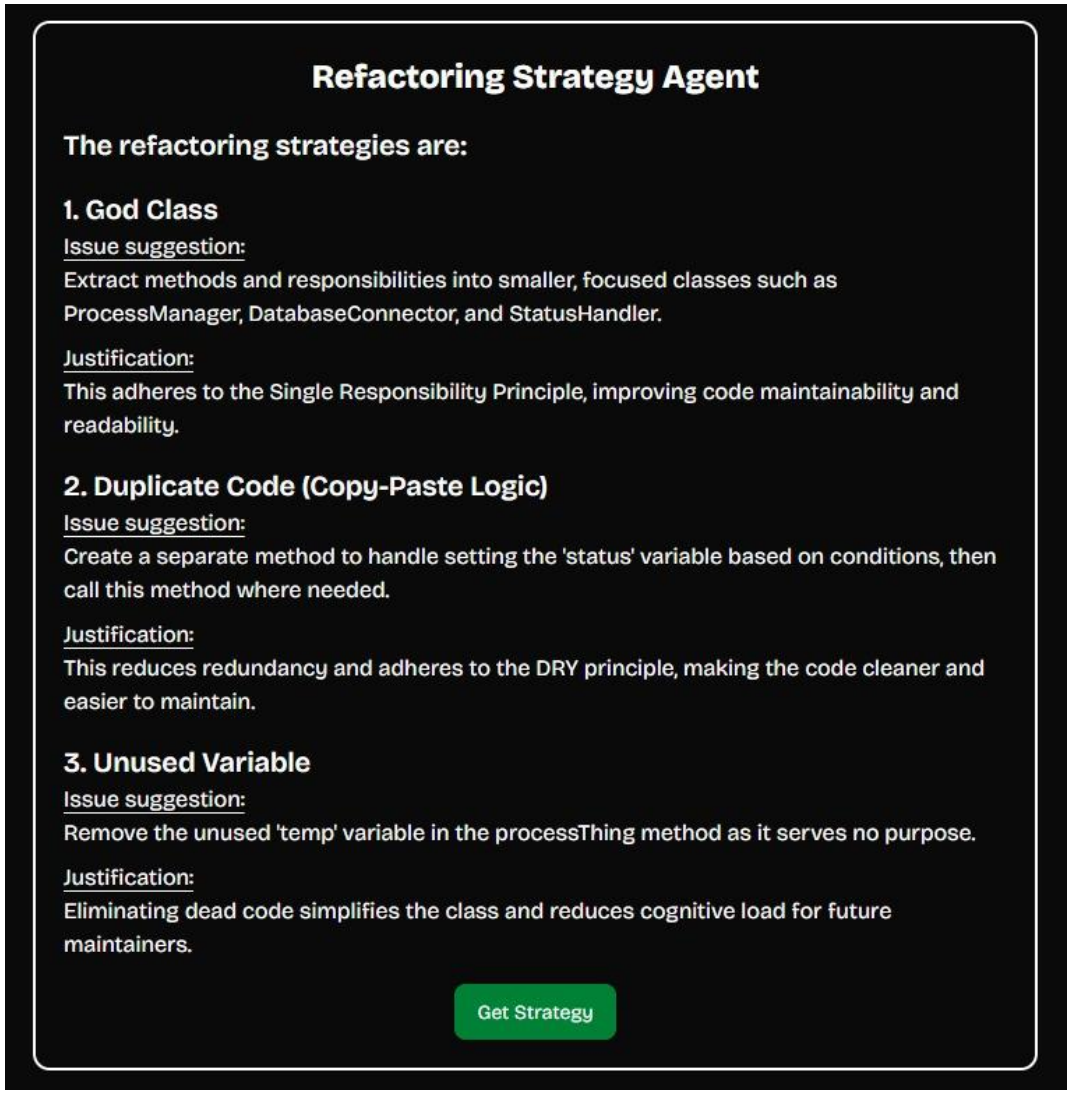
- Anti-patterns are code structures that technically work but unintentionally reflect poor practice and are hard to maintain, scale, or optimise.
- They waste resources, increase technical debt, and often require significant developer effort to resolve.
- Legacy codebases are especially vulnerable, as they tend to accumulate large volumes of anti-patterns over time, making long-term maintenance costly.
- Yet fixing them is rarely prioritised, as it demands high effort with little short-term return, despite being critical for the longevity of software.

How would one use it?

Example 1 – During regular development

- Our tool can be used ad hoc, by ingesting individual code files or snippets, much like querying ChatGPT during development.
- It automatically detects anti-patterns, formulates appropriate resolution strategies, and outputs a corrected version of the code.
- It also generates a human-readable explanation of the issues found and the rationale behind the fix, which is useful for documentation and future reference.


Analysis Agent Response
The issues detected are:
1. God Class
Issue location: GodClass.java
The class GodClass has too many responsibilities, including process logic, database connection handling, and global state management.
2. Duplicate Code (Copy-Paste Logic)
Issue location: GodClass.java:doEverything() method
The same block of code is repeated twice to set the 'status' variable, indicating a lack of DRY (Don't Repeat Yourself) principle.
3. Unused Variable
Issue location: GodClass.java:processThing() method
The 'temp' variable is declared but never used, indicating poor code quality and potential for removal to simplify the class.
Get Analysis


Refactoring Strategy Agent
The refactoring strategies are:
1. God Class
Issue suggestion:
Extract methods and responsibilities into smaller, focused classes such as ProcessManager, DatabaseConnector, and StatusHandler.
Justification:
This adheres to the Single Responsibility Principle, improving code maintainability and readability.
2. Duplicate Code (Copy-Paste Logic)
Issue suggestion:
Create a separate method to handle setting the 'status' variable based on conditions, then call this method where needed.
Justification:
This reduces redundancy and adheres to the DRY principle, making the code cleaner and easier to maintain.
3. Unused Variable
Issue suggestion:
Remove the unused 'temp' variable in the processThing method as it serves no purpose.
Justification:
Eliminating dead code simplifies the class and reduces cognitive load for future maintainers.
Get Strategy

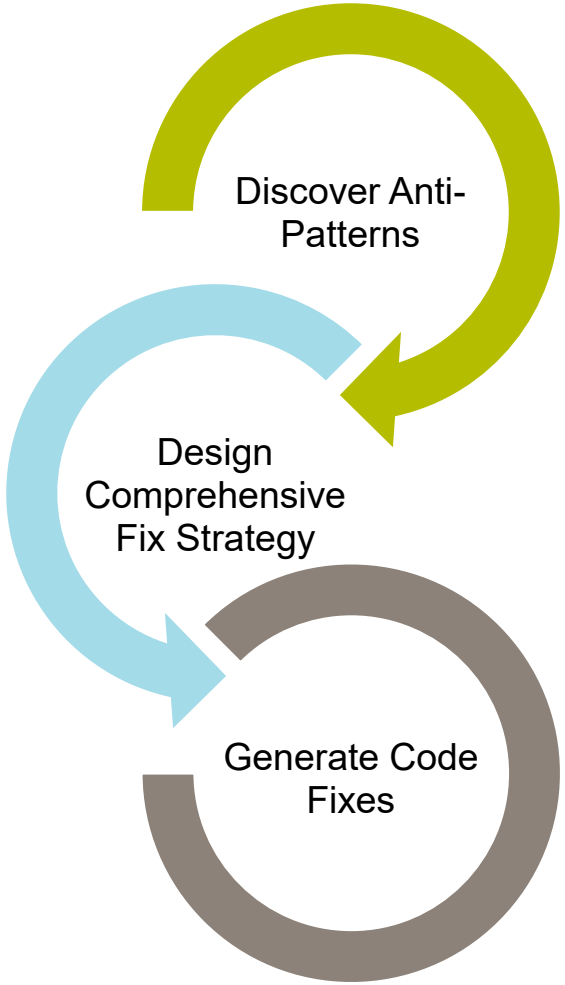
How would one use it?

Example 2 – Automated codebase resolution

- Our tool can be run across entire codebases, not just individual files, enabling large-scale detection and resolution of anti-patterns.
- It generates versioned code changes that are easy to review and integrate, ensuring developers remain in control of what gets applied.
- This significantly accelerates the software modernisation process by automating one of its most time-consuming stages.
- Engineering teams can now redirect their effort toward more meaningful work such as building features or improving design.

Where we are today

- A functional prototype capable of identifying anti-patterns and recommending resolution strategies for individual files and snippets.
- A highly extensible architecture supporting iterative development.
- A well-defined system design, with implementation progressing across key components.



Where we'll be tomorrow

- Automated codebase resolution producing change sets ready for developer review.
- Enhanced pattern detection through integration with static analysis tools like SonarQube.
- Seamless switching between different LLMs for benchmarking and refinement.
- Further optimisation efforts to reduce energy consumption and improve efficiency.

Thank you!

GitHub: Public repository coming soon!

Weekly Blog

Contact Details