AICE Catalyst Agent Challenge

You will build an AI agent that transforms raw business requirements into a structured, phased development plan. This agent will help developers and code assistants (like GitHub Copilot or Claude) execute projects more efficiently by breaking down business requirements into actionable tasks.

Time Allocation: Abou 2-t 3 days

### Agent Goal

Transform a raw business requirement into a clear, phased development plan that real developers and code assistants can act on immediately.

Keep the solution focused on one agent that reasons step by step and uses tools to extract structure, estimate effort, define acceptance criteria, and prepare prompts for code assistants.

### What to build

A single LangGraph agent that calls a small set of custom tools and/or consists in a deterministic agentic workflow to go from unstructured text to a structured plan.

The agent should be able to read a requirement, ask itself what information is missing, call tools to fill gaps, and output an organized plan in phases and tasks.

You will be asked to explain your agentic architecture.

### Allowed models

Please use a local model or a deployed endpoint.

You can make use of below code snippet for usage of the gpt-4o-mini deployment provided for this exercise if you deem suitable:

```
from langchain_openai import AzureChatOpenAI
from langchain_core.messages import SystemMessage, HumanMessage


endpoint = "https://catalyst-agent.openai.azure.com/"
```

```
deployment = "gpt-4o-mini"
api_key =
"BxLixgXbKkJmDJgJwFt7BwYIxDDwQSxlT9TUwj2eErd7ehB828swJQQJ99BJACY
eBjFXJ3w3AAABACOGbSoo"
api_version = "2025-01-01-preview"


chat = AzureChatOpenAI(
    azure_endpoint=endpoint,
    azure_deployment=deployment,
    api_version=api_version,
    api_key=api_key,
)

# Example
messages = [
    SystemMessage(content="You are a helpful assistant."),
    HumanMessage(content="I am going to Paris, what should I
see?"),
]

resp = chat.invoke(messages)
print(resp.content)
```

### Input to the agent

A short text requirement (around 100–300 words) describing a feature or small project, including any constraints or success criteria if available.

If the input is vague, the agent should still produce a plan by making reasonable assumptions and clearly flagging any risks or unknowns.

### Expected output

A JSON or YAML development plan that is organized by phases, with each phase containing tasks, estimates, dependencies, prompt, and acceptance criteria.

Each task should include a short prompt that a can be leveraged by a code assistant to accelerate implementation.

### Minimum tools to implement

Implement at least five custom tools/method/coroutines so the agent can do practical work rather than only chatting.

- parse_requirements_tool: Extract features, constraints, stakeholders, and success criteria from the raw text into a simple structured object.

- estimate_complexity_tool: For a feature, return a complexity label, estimated days, and notable risks to guide planning.

- generate_tasks_tool: Break a feature into granular tasks with short descriptions and an initial guess at dependencies and priority.

- create_acceptance_criteria_tool: Produce clear, testable criteria using simple Given/When/Then wording for each task or feature. Suite of unit tests and if relevant integration tests descritption should be provided as part of the output of this task.

- generate_prompt_for_copilot_tool: Turn a task and its acceptance criteria into a concise, high-signal prompt suitable for GitHub Copilot or Claude.

- detect_dependencies_tool (optional): From the list of tasks, mark blocking relationships to suggest a sensible execution order.

### Plan structure guidance

Organize the plan into phases so it is easy to read and schedule, such as Foundation, Core Features, and Advanced or Integrations.

Sum up total estimated effort across phases and clearly list risks and assumptions that might affect scope or schedule.

### Quality bar

The output should be specific enough that a developer can start work with minimal clarification, not just a generic outline.

Prompts for code assistants should be concise, state the goal, context, constraints, and acceptance criteria, testable elements , and avoid vague instructions.

### Time guidance

Aim to scope features so the agent can produce a complete, phasebased plan within a single run for a small requirement, keeping results deterministic and easy to review.

### Model logging

Log the final runnable agent as a LangChain flavor model in a local MLflow tracking run, using a supported class so it can be loaded and invoked later.

Briefly document what object you logged and confirm it can be reloaded and used for inference in a clean process.

### Work submission

Please create a new project in your personal github repository, and submit the checked in project URL.

Add a set of inputs and outputs produced by your catalyst agents that you will present us to demonstrate its capabilities.

### Good luck!

You will present your solution in details and what you have learn throughout the development. We're excited to see your AI engineering skills in action!