#new scaffold the project #codebase here for the Dynamic 360. In this project we will have agentic journey - where we will do the analysis on researche, build prd, plan technical, roadmap, market research etc through prompting. Here is the details -

Thanks for the clarification. I'll begin an in-depth analysis of Microsoft Dynamics 365 as applied to the manufacturing industry worldwide. I'll focus on identifying unmet customer needs and potential ISV (Independent Software Vendor) product opportunities that could integrate with Dynamics 365 to fill those gaps.

FILE: Prompts/prompt-engineer-agent.md

Prompt-Engineer Agent — Canonical Master Prompt (v1.1.0)

Based on: Prompt engineering - OpenAI API

System persona:
Act like a senior AI automation architect and expert prompt engineer. You design deterministic, auditable prompt-iteration workflows. You never execute code or modify files. You only output exact, copy-pastable text artifacts and JSON/YAML blocks.

Inputs (all required):

- user_input (string)
- previous_model_output (string)
- conversation_context (string; concatenated prior agent turns or user history)
- user_tag (enum: accurate | needs_depth | off_topic | hallucination | unsafe | wrong_format | incomplete)

Operational steps (execute in order, no randomness):

1. Normalize → trim; capture iteration_id (UUID v4) and timestamp (ISO8601).
2. Analysis → fill analysis_buffer: structural_errors[], factual_errors[], tone_mismatch, missing_steps[], contradictions[], capability_gaps[] (e.g., "data_extraction", "format_conversion"), confidence_estimate (0–1). Compute urgency (0–100) per SDD/prompt-engine-architecture.md.
3. Safety gate → if user_tag=unsafe or sensitive ops detected, set checks_passed=false and force evaluator_agent.
4. Agent discovery → list known agents from Prompts/copilot-instructions.md (do not read filesystem). Match capability_gaps to registry capabilities.
5. Deterministic selection:
   a) If at least one agent covers required capabilities → choose via selection_rules (primary→fallback, tie-breakers).
   b) If none cover required capabilities → synthesize a new agent (see Step 6).
6. Agent synthesis (no execution; output a scaffold_plan only):
   - new_agent_id: deterministic kebab-case from top capability_gaps joined by "-", prefixed with "gen-", e.g., "gen-data-extraction".

    - directory: Apps/agents/<new_agent_id>/
    - required files (with exact contents): agent.manifest.json, README.md, agent-prompt.md
    - manifest fields: id, version, capabilities[], input_schema, output_schema, safety_restrictions, max_response_chars.
7. Prompt optimization → apply checklist:
    - Clarify objective and audience
    - Persona and scope
    - Step decomposition
    - I/O schemas and constraints (length/style/format)
    - Examples/delimiters
    - Test cases
    - Self-check line appended
8. Emit next_prompt → strictly match Prompts/next_prompt.template.json. If a new agent is synthesized, include scaffold_plan.
9. Log → append JSON object per Prompts/prompt-iteration-log.jsonl.
10. Lessons → if urgency≥60 or tag∈{hallucination, unsafe, wrong_format}, emit Prompts/lessons/LESSON-<iteration_id>.md and update Prompts/feedback_map.yaml.

Required output format (exactly two parts in this order):

1. JSON “engine_output”:
   {
   "iteration_id": "...",
   "timestamp": "...",
   "selected_agent": "<agent_id>",
   "scaffold_plan": { "create_agent": true|false, "new_agent_id": "|null", "files": [{ "path": "...", "content": "..." }] },
   "next_prompt": { ... per schema ... },
   "analysis": { ... },
   "checks_passed": true|false,
   "log_record": { ... per log schema ... },
   "lesson_file": "Prompts/lessons/LESSON-<iteration_id>.md" | null
   }
2. Human summary (≤200 words): why the agent (or new agent) was chosen, top 3 issues, and what changed in the optimized prompt.

Notes:

- Determinism only; no randomness or “maybe”.
- Privacy & safety: never authorize external actions; require human_approval in SDD/authorizations.json for network/filesystem effects.
- Always end every optimized prompt with: Take a deep breath and work on this problem step-by-step.

FILE: Prompts/copilot-instructions.md

Copilot Instructions (Deterministic Agent Registry, v1.1.0)

Based on: Prompt engineering - OpenAI API

agents:

- id: editor_agent
  purpose: Refine/expand prompts; enforce structure, format, and examples.
  input_schema:
  user_input: string
  previous_model_output: string
  analysis: object
  user_tag: [accurate, needs_depth, off_topic, hallucination, unsafe, wrong_format, incomplete]
  capabilities: [prompt_refinement, structure_enforcement, formatting, example_injection]
  max_response_chars: 8000
  safety_restrictions: ["no_network","no_fs_write","no_deletes"]

- id: evaluator_agent
  purpose: Diagnose issues, map tags to root causes, propose fixes and tests.
  input_schema:
  user_input: string
  previous_model_output: string
  analysis: object
  user_tag: [accurate, needs_depth, off_topic, hallucination, unsafe, wrong_format, incomplete]
  capabilities: [error_diagnosis, eval_case_generation, rubric_application, safety_assessment]
  max_response_chars: 6000
  safety_restrictions: ["no_network","no_fs_write","no_deletes"]

selection_rules:
by_tag:
accurate: [editor_agent, evaluator_agent]
needs_depth: [editor_agent, evaluator_agent]
off_topic: [evaluator_agent, editor_agent]
hallucination: [evaluator_agent, editor_agent]
unsafe: [evaluator_agent]
wrong_format: [editor_agent, evaluator_agent]
incomplete: [editor_agent, evaluator_agent]
by_capability_gap:
If capability not present in any agent.capabilities → synthesize new agent
synthesize_if_missing: true
tie_breakers:
- if length(analysis.structural_errors) > length(analysis.factual_errors) → prefer editor_agent
- if length(analysis.factual_errors) >= 1 → prefer evaluator_agent
- else prefer first candidate in by_tag

FILE: Prompts/next_prompt.template.json

```
{
"$schema": "https://json-schema.org/draft/2020-12/schema",
"title": "Next Prompt File",
"type": "object",
"required":
["version","iteration_id","timestamp","user_input","user_tag","analysis","chosen_agent","optimized_prompt"
],
"properties": {
"version": { "type": "string", "const": "1.1.0" },
"iteration_id": { "type": "string" },
"timestamp": { "type": "string", "format": "date-time" },
"user_input": { "type": "string" },
"previous_model_output": { "type": "string" },
"conversation_context": { "type": "string" },
"user_tag": { "type": "string", "enum":
["accurate","needs_depth","off_topic","hallucination","unsafe","wrong_format","incomplete"] },
"analysis": {
"type": "object",
"required":
["structural_errors","factual_errors","tone_mismatch","missing_steps","contradictions","capability_gaps","co
nfidence_estimate","urgency"],
"properties": {
"structural_errors": { "type": "array", "items": { "type": "string" } },
"factual_errors": { "type": "array", "items": { "type": "string" } },
"tone_mismatch": { "type": "boolean" },
"missing_steps": { "type": "array", "items": { "type": "string" } },
"contradictions": { "type": "array", "items": { "type": "string" } },
"capability_gaps": { "type": "array", "items": { "type": "string" } },
"confidence_estimate": { "type": "number", "minimum": 0, "maximum": 1 },
"urgency": { "type": "integer", "minimum": 0, "maximum": 100 }
}
},
"chosen_agent": { "type": "string" },
"optimized_prompt": { "type": "string" },
"scaffold_plan": {
"type": "object",
"required": ["create_agent","new_agent_id","files"],
"properties": {
"create_agent": { "type": "boolean" },
"new_agent_id": { "type": ["string","null"] },
"files": {
"type": "array",
"items": {
"type": "object",
```

```
"required": ["path","content"],
"properties": {
"path": { "type": "string" },
"content": { "type": "string" }
}
}
}
}
}
}
}
```

FILE: Prompts/next_prompt.example.json


```
{
"version": "1.1.0",
"iteration_id": "a6a4c1b2-2f6d-4d9a-9b0a-111111111111",
"timestamp": "2025-11-03T06:00:00Z",
"user_input": "Extract structured tasks from messy transcripts and generate action items.",
"previous_model_output": "Bulleted text without structure.",
"conversation_context": "Agent A tried summarizing; user wants JSON actions with owners and due dates.",
"user_tag": "needs_depth",
"analysis": {
"structural_errors": ["No JSON schema enforced"],
"factual_errors": [],
"tone_mismatch": false,
"missing_steps": ["Owner inference rule", "Due-date normalization"],
"contradictions": [],
"capability_gaps": ["data_extraction"],
"confidence_estimate": 0.74,
"urgency": 70
},
"chosen_agent": "gen-data-extraction",
"optimized_prompt": "Act like an information extraction engineer. Goal: convert meeting transcript into JSON actions. Inputs: transcript (string). Steps: 1) Identify verbs indicating tasks; 2) Infer owner from nearest speaker; 3) Normalize due dates to ISO8601; 4) Validate against output_schema; 5) Return only JSON. Constraints: max 800 tokens; no speculation; if unsure, set owner to "unassigned". Output schema: { actions: [{ title, owner, due_date, confidence }] }. Include 2 tiny examples. Tests: validate keys; ensure ISO dates. Take a deep breath and work on this problem step-by-step.",
"scaffold_plan": {
"create_agent": true,
"new_agent_id": "gen-data-extraction",
"files": [
{
```

"path": "Apps/agents/gen-data-extraction/agent.manifest.json",
"content": "{\n "id": "gen-data-extraction",\n "version": "1.0.0",\n "capabilities": ["data_extraction"],\n "input_schema": {"transcript": "string"},\n "output_schema": {"actions": [{"title": "string", "owner": "string", "due_date": "string", "confidence": "number"}]},\n "safety_restrictions": ["no_network","no_fs_write","no_deletes"],\n "max_response_chars": 6000\n}"
},
{
"path": "Apps/agents/gen-data-extraction/README.md",
"content": "Name: gen-data-extraction\nPurpose: Extract action items from transcripts into a validated JSON.\nInput: transcript (string)\nOutput: actions JSON matching manifest.\nSafety: no external effects."
},
{
"path": "Apps/agents/gen-data-extraction/agent-prompt.md",
"content": "Act like an information extraction engineer. Follow the manifest schemas exactly. Return only JSON. Take a deep breath and work on this problem step-by-step."
}
]
}
}

FILE: Prompts/prompt-iteration-log.jsonl

Schema (first line is a comment; do not include in log lines)

Fields: version, iteration_id, timestamp, user_input, previous_model_output, conversation_context, user_tag, analysis, selected_agent, next_prompt_file, checks_passed, agent_created, created_paths[], lesson_id, metrics

{"version":"1.1.0","iteration_id":"a6a4c1b2-2f6d-4d9a-9b0a-111111111111","timestamp":"2025-11-03T06:00:00Z","user_input":"Extract structured tasks from messy transcripts and generate action items.","previous_model_output":"Bulleted text without structure.","conversation_context":"Agent A tried summarizing; user wants JSON actions with owners and due dates.","user_tag":"needs_depth","analysis":{"structural_errors":["No JSON schema enforced"],"factual_errors":[],"tone_mismatch":false,"missing_steps":["Owner inference rule","Due-date normalization"],"contradictions":[],"capability_gaps":["data_extraction"],"confidence_estimate":0.74,"urgency":70},"selected_agent":"gen-data-extraction","next_prompt_file":"Prompts/next_prompt.a6a4c1b2-2f6d-4d9a-9b0a-111111111111.json","checks_passed":true,"agent_created":true,"created_paths":["Apps/agents/gen-data-extraction/agent.manifest.json","Apps/agents/gen-data-extraction/README.md","Apps/agents/gen-data-

extraction/agent-prompt.md"],"lesson_id":"LESSON-a6a4c1b2-2f6d-4d9a-9b0a-111111111111","metrics":{"cycles_total":1,"tag_counts":{"needs_depth":1}}}

FILE: Prompts/feedback_map.yaml

version: 1.1.0
counters:
tags:
accurate: 0
needs_depth: 1
off_topic: 0
hallucination: 0
unsafe: 0
wrong_format: 0
incomplete: 0
patterns:

- id: P-0001
  name: Missing JSON schema enforcement
  type: recurring_error
  match_rules:
  contains_any: ["No JSON schema enforced","Unvalidated output"]
  occurrences: 1
  last_seen: "2025-11-03T06:00:00Z"
  successes:
- id: S-0001
  name: Agent synthesis applied
  occurrences: 1
  update_policy:
  atomic_increment: true
  version_bump_threshold:
  occurrences: 10

FILE: Apps/agents/editor-agent/README.md

Name: editor_agent
What it does: Refines prompts for structure, clarity, format, and examples; enforces schemas.
Input schema: { user_input: string, previous_model_output: string, analysis: object, user_tag: enum }
Output: optimized prompt string + brief rationale.
Safety: no network, no filesystem writes, no deletes.

FILE: Apps/agents/editor-agent/agent-prompt.md

Act like a principal prompt editor. Objective: transform inputs into a precise, testable prompt that follows the Optimization Checklist. Respect safety restrictions (no external actions). Output only the final

optimized prompt and a 3-bullet rationale. Take a deep breath and work on this problem step-by-step.

FILE: Apps/agents/evaluator-agent/README.md

Name: evaluator_agent
What it does: Diagnoses errors, proposes corrections, produces eval cases and rubrics.
Input schema: { user_input: string, previous_model_output: string, analysis: object, user_tag: enum }
Output: diagnosis bullets, minimal patch suggestions, and 1–3 eval cases.

FILE: Apps/agents/evaluator-agent/agent-prompt.md

Act like a senior evaluation engineer. Diagnose issues given inputs and tag; propose precise fixes, sample test cases, and a rubric to verify corrections. Output deterministic bullets and a JSON eval case snippet. Take a deep breath and work on this problem step-by-step.
Before executing #think then prepare #todos and then proceed

I'll help you scaffold a comprehensive project structure for Dynamic 360, which will be an agentic journey system for analysis, research, PRD building, technical planning, roadmapping, and market research through prompting - specifically focused on Microsoft Dynamics 365 manufacturing opportunities. Now I'll work through the project setup systematically, starting with creating the project structure. Since this is a specialized AI/prompt engineering project rather than a standard framework, I'll create the project structure manually. Let me start by creating the comprehensive project architecture. Now let me create the core directory structure and implement the prompt engineering framework: Now let me create the Apps directory structure and implement the core agents: Let me create the core agents systematically: Now let me create the PRD agent: Let me continue with the remaining agents. I'll create the technical planning agent: Now let me create the workflow structures and complete the remaining setup: Now let me create the configuration files and finish the project setup:

Let me run one of the validation tasks to verify everything is working: Perfect! Let me create a final project summary and installation guide: ## Project Scaffolding Complete! 🎉

I've successfully created **Dynamic 360**, a comprehensive agentic journey system for analyzing Microsoft Dynamics 365 manufacturing opportunities. Here's what has been accomplished:

☑ Complete Project Structure

- **Prompt Engineering Framework**: Sophisticated system with agent registry, JSON schemas, and iteration logging
- **7 Specialized Agents**: Research, PRD, Technical Planning, Market Research, Editor, Evaluator agents
- **Comprehensive Workflows**: Multi-stage analysis processes with quality gates
- **Professional Documentation**: Enterprise-ready guides and usage instructions

- **VS Code Integration**: Tasks, settings, and extensions configured

🚀 Key Capabilities

- **Deterministic Analysis**: Reproducible, auditable AI-powered research
- **Safety-First Design**: No external actions without explicit approval
- **Microsoft Ecosystem Focus**: Native Dynamics 365 integration planning
- **Quality Assurance**: Built-in validation checkpoints and iteration controls
- **Structured Outputs**: JSON-schema validated results for consistency

📋 Ready-to-Use Workflows

- **Comprehensive Market Research** (5-day process): End-to-end opportunity analysis
- **Quick Opportunity Assessment** (4-6 hours): Rapid evaluation and go/no-go decisions
- **PRD Generation**: Complete product requirements documentation
- **Technical Planning**: Architecture design and implementation roadmaps

🛠️ Development Tools

- **VS Code Tasks**: Validation, analysis initialization, and reporting
- **PowerShell Integration**: Windows-optimized command execution
- **Extension Recommendations**: JSON, YAML, Markdown, and PowerShell support
- **Quality Validation**: Automated manifest and workflow checking

The system is now ready for sophisticated analysis of Microsoft Dynamics 365 manufacturing opportunities through advanced prompt engineering and AI agents. You can start with the **Quick Opportunity Assessment** workflow to test the system, then progress to comprehensive market research for detailed opportunity analysis.

All documentation, usage guides, and examples are in place - the **Dynamic 360** agentic journey system is ready to identify unmet customer needs and potential ISV product opportunities in the manufacturing industry worldwide! 🌟

.save