perplexity

expand the article to include in the comparising the other abstraction introduced by the AAMAD framework were agents personas are defined in .md files and rules to be followed are loaded from .mdc files. Here is the repo as reference: https://github.com/synaptic-ai-consulting/AAMAD

Here's the updated markdown, now including AAMAD's `.md` personas and `.mdc` rules as a first-class abstraction alongside Copilot SKILLs and Claude Code Skills.

From Copilot & Claude Code SKILLs to Agentic Crews

How to Turn Skills and AAMAD Personas into Tasks and Agents with CrewAI

As an AI-native developer, you now work with several powerful abstractions:

- **GitHub Copilot SKILLs** – filesystem-based "mini-agents" that package instructions, tools, and resources for specific workflows.[1] [2]
- **Claude Code Skills** – similar folders that extend Claude Code with reusable workflows, metadata, scripts, and domain resources.[3] [4] [5]
- **AAMAD personas and rules** – `.md` agent persona files plus `.mdc` rules that define how AI coding agents should behave in your repo.[6] [7]
- **Crews (CrewAI)** – coordinated teams of agents and tasks that execute these workflows as part of your backend system.[7] [8]

This guide shows you how to use all three "skills" dialects (Copilot, Claude Code, AAMAD) as specifications for CrewAI agents and tasks.

1. Mental model: skills, personas, agents, and tasks

Skills (Copilot & Claude Code)

In Copilot and Claude Code, a **Skill** is a reusable, named capability:[2] [4] [1] [3]

- A `SKILL.md` file with metadata (name, description, tags) and detailed instructions.
- Optional instruction files (e.g., `FORMS.md`, `REFERENCE.md`) that document workflows or domain rules.[4] [9]
- Optional scripts (linters, validators, converters) and resources (schemas, templates, API docs, examples).[5] [4]

Copilot SKILLs focus on dev workflows inside VS Code and GitHub. Claude Code Skills focus on Claude's VM-like coding environment and can be used via Claude Code, Claude API, and the Agent SDK.[10] [1] [2] [3] [4]

AAMAD personas (.md) and rules (.mdc)

The AAMAD framework adds another, very compatible layer:[6] [7]

- **Agent personas (.md)**
  - Located in `.cursor/agents/` (or similar), each persona file describes a role (e.g., Product Manager, System Architect, QA Engineer) with responsibilities, tone, and workflows.[7] [6]
  - Each persona owns specific epics and artifacts in AAMAD's Define–Build–Deliver phases, following single-responsibility principles.[7]
- **Rules (.mdc)**
  - Cursor rule files in `.cursor/rules/` (.mdc) act as architectural blueprints for your agents.[6]
  - They encode project-wide and feature-specific constraints: architecture decisions, coding standards, security/compliance policies, and allowed patterns.[6]
  - They effectively serve as a "constitution" for AI agents, defining how agents should think, work, and make decisions across phases.[6]

This combination gives you **personas** (who you are) plus **rules** (how you must behave) as reusable, version-controlled context.[7] [6]

Agents and tasks (CrewAI)

- A **CrewAI agent** is a persona with a role, goal, tools, and behavior instructions.[8] [11]
- A **CrewAI task** is a concrete assignment with description, inputs, expected outputs, and a designated agent.[12] [8]

You can combine all of these like this:

Layer / artifact	Purpose in your system
Copilot SKILL / Claude Code Skill	Encodes a reusable capability/workflow
AAMAD persona `.md`	Encodes a durable agent persona for that capability

Layer / artifact	Purpose in your system
AAMAD rules `.mdc`	Encodes governance and patterns for all agents
CrewAI agent	Runtime persona that implements the skill + persona + rules
CrewAI task	Runtime job using the agent, with inputs/outputs

2. Converting skills and personas into CrewAI agents + tasks

You can't drop a SKILL folder or `.md` persona directly into CrewAI, but you can treat them as **specifications** for CrewAI agents and tasks.[11] [2] [4] [12] [7]

Step 1: Read the Skill and persona as a contract

For a given capability, pull together three inputs:

- From **Copilot / Claude Skill** (`SKILL.md`, etc.): goal, scope, inputs, outputs, workflow steps, tools.[9] [2] [4]
- From **persona** `.md`: the role's responsibilities, tone, decision style, and collaboration patterns.[7] [6]
- From **rules** `.mdc`: architectural constraints, coding standards, compliance rules, and allowed patterns.[6]

Together, these give you:

- *What* the agent should do and when.
- *Who* the agent is and how it should communicate.
- *How* it must behave to stay compliant and on-architecture.

Step 2: Define a single-responsibility CrewAI agent

You then create a CrewAI agent that **embodies all three layers**:[8] [11] [7] [6]

- **Role / name** – aligned with the persona and SKILL (e.g., `ComplianceReviewAgent`, `GapAnalysisAgent`).
- **System prompt / backstory** – synthesized from:
  - The Skill's instructions (what to do, algorithms, output schema).[2] [9]
  - The persona `.md` (voice, expertise level, collaboration rules).[7] [6]
  - Key constraints from relevant `.mdc` rules (never bypass these checks, always log in this format, etc.).[6]
- **Tools** – wrappers around scripts and external systems referenced in Skills or rules (e.g., deterministic validators, linters, policy lookup functions).[4] [5] [6]

This moves you from "editor-only" agents (Copilot, Claude, Cursor) to **backend agents** that work the same way, but as part of a crew.

Step 3: Define task templates that match the contracts

Next you define CrewAI tasks whose inputs/outputs mirror the Skill + persona expectations:[12] [8] [7]

- Use the Skill's input/output schema as the task's contract.
- Use the persona's responsibilities to phrase the task description ("As the Compliance Review Agent, your job is to…").
- Use `.mdc` rules to define mandatory steps (e.g., "run deterministic checks before LLM reasoning", "log all high-risk decisions").[6]

Example task:

- **Task:** `run_compliance_review`
- **Agent:** `ComplianceReviewAgent`
- **Inputs:** `draft_contract`, `policy_set`, optional `jurisdiction`
- **Expected output:**
  - Summary of overall compliance posture
  - Structured list of findings (clause, issue, severity, recommended change)
  - Explicit references to rules satisfied/violated

Now your Skill + persona + rules live on as a **concrete, callable task** in your Crew.

3. How the three abstractions line up

You can think of Copilot SKILLs, Claude Code Skills, and AAMAD personas/rules as complementary views on the same underlying capability.

Representation comparison

Aspect	Copilot SKILLs	Claude Code Skills	AAMAD personas + `.mdc`
Core file	`SKILL.md`	`SKILL.md`	Persona `.md` + rule `.mdc`
Main focus	Dev workflows, repo-aware helpers	Dev workflows + shell/VM execution	Project-wide architecture, roles, and governance
Where it lives	Repo or extension folders	Claude Code VM filesystem	`.cursor/agents/` and `.cursor/rules/` in your repo
What it encodes	Goal, inputs, outputs, steps, tools	Goal, inputs, outputs, steps, scripts, resources	Roles, responsibilities, standards, constraints
How AI uses it	Copilot activates SKILL when relevant	Claude loads content in levels + runs scripts	Cursor agents follow personas and rules as a "constitution"[6]

Mapping to CrewAI

Concept	CrewAI mapping
Skill goal + steps	Task description and internal reasoning style
Skill inputs / outputs	Task input schema + expected output structure
Persona `.md`	Agent identity, tone, collaboration and responsibility
Rules `.mdc`	Hard constraints baked into tools, prompts, and validations
Scripts (Copilot/Claude/AAMAD)	CrewAI tools (deterministic helpers)

When you define a CrewAI agent, you are effectively **merging** the Skill and the persona into a single runtime entity, grounded in `.mdc` rules.

4. Example: document pipeline across all layers

Suppose you're building a document workflow with four capabilities:

1. Document ingestion / content input
2. Document drafting / writing
3. Gap analysis
4. Legal & compliance comparison

You might have:

- Copilot SKILLs like `document-ingestion`, `doc-writer`, `gap-analysis`, `compliance-check`.[1] [2]
- Claude Code Skills for PDF ingestion, PRD writing, spec vs implementation review, contract compliance.[3] [5] [4]
- AAMAD personas for Product Manager, System Architect, QA Engineer, and a specialized Compliance persona, plus `.mdc` rules under `.cursor/rules/` that define architecture and compliance constraints.[7] [6]

You can map these to CrewAI like this:[11] [8] [12] [7]

Capability	Skills (Copilot / Claude)	AAMAD persona + rules	CrewAI agent + task
Ingestion / content input	`document-ingestion` SKILL, "doc-ingest" Skill	Ingestion/Backend persona + ingest rules `.mdc`	`IngestionAgent` + `ingest_document` task
Document drafting / writing	`doc-writer`, PRD writer / contract drafter Skills	Product Manager or Authoring persona + style rules	`AuthoringAgent` + `draft_document` task
Gap analysis	`gap-analysis` Skill, "spec vs implementation reviewer"	QA / Review persona + quality rules `.mdc`	`GapAnalysisAgent` + `run_gap_analysis` task

Capability	Skills (Copilot / Claude)	AAMAD persona + rules	CrewAI agent + task
Legal & compliance comparison	`compliance-check` Skill, "compliance reviewer" Skill	Compliance persona + compliance/safety rules `.mdc`	`ComplianceReviewAgent` + `run_compliance_review` task

Inside your editor, Copilot, Claude, and Cursor respect these artifacts; in your backend, CrewAI consumes the same semantics through agents and tasks.

5. Orchestrating everything as an Application Crew

Once you have agents and tasks for each capability, you can compose them into an **Application Crew** that mirrors AAMAD's multi-phase approach:[8] [7]

1. **Define / ingest**
   - `IngestionAgent` runs `ingest_document`.
   - Outputs normalized chunks, metadata, and maybe a summary artifact.
2. **Build / draft**
   - `AuthoringAgent` runs `draft_document` using RAG over ingestion outputs and following authoring persona + `.mdc` rules (style, structure, forbidden patterns).[7] [6]
3. **Analyze**
   - `GapAnalysisAgent` runs `run_gap_analysis` to compare the draft against templates/standards.
4. **Govern / comply**
   - `ComplianceReviewAgent` runs `run_compliance_review`, heavily constrained by `.mdc` compliance rules and deterministic tools.[6]

You can orchestrate this Crew sequentially for high-stakes contexts, or run some tasks in parallel (e.g., gap analysis and compliance review on the same draft) for efficiency.[8] [7]

In AAMAD terms, you are effectively turning the **Application Crew** from the playbook into a running CrewAI system, while still leveraging your editor-side Skills and Cursor rules to guide development.[8] [7] [6]

6. Practical checklist to unify Skills, personas, rules, and crews

When you want to bring all of this together in your own project:[9] [2] [4] [11] [12] [7] [6]

1. **Pick a narrow capability**
   - e.g., gap analysis or compliance review.
2. **Collect its artifacts**
   - Relevant Copilot / Claude Skills (`SKILL.md` + scripts).
   - The corresponding AAMAD persona `.md`.
   - Any `.mdc` rules that govern this area.

3. **Define a CrewAI agent**
    - Merge the Skill instructions, persona traits, and core rules into the agent's system prompt and toolset.
4. **Define one or more CrewAI tasks**
    - Use the Skill's inputs/outputs as the contract and the persona's responsibilities to phrase the task.
5. **Wrap scripts and checks as tools**
    - Turn existing scripts (from Skills or AAMAD repo) into CrewAI tools that run deterministically before/after LLM reasoning.
6. **Respect `.mdc` rules at runtime**
    - Enforce key rules via tools, validations, and prompt constraints, not just as documentation.
7. **Compose a Crew**
    - Combine multiple Skill-backed agents into an end-to-end Crew that mirrors your AAMAD phases and Application Crew design.

Working this way, you get a consistent story from **editor to backend**:

- Skills and personas define capabilities and behavior.
- `.mdc` rules provide governance as code.
- CrewAI executes those capabilities as orchestrated, auditable multi-agent workflows.

Is it useful for you if I next draft a short, concrete example showing one AAMAD persona + `.mdc` rules + Claude Skill mapped into a single CrewAI agent definition in pseudo-code?

⁂

1. https://github.com/github/awesome-copilot/blob/main/docs/README.skills.md
2. https://github.com/github/awesome-copilot/blob/main/skills/make-skill-template/SKILL.md
3. https://code.claude.com/docs/en/skills
4. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview
5. https://websearchapi.ai/blog/how-to-create-claude-code-skills
6. https://www.linkedin.com/pulse/three-layers-framework-agentic-code-quality-carmelo-iaria-fp2uf
7. THE-AGENTIC-ARCHITECT-PLAYBOOK.pdf
8. CrewAI-Lightning-Lesson-Series-1.md
9. https://resources.anthropic.com/hubfs/The-Complete-Guide-to-Building-Skill-for-Claude.pdf
10. https://platform.claude.com/docs/en/build-with-claude/skills-guide
11. https://docs.crewai.com/en/concepts/agents
12. https://docs.crewai.com/en/concepts/tasks
13. https://synaptic-ai-consulting.github.io/learn-aamad/

14. https://github.com/Bagana-AI-Learn
15. https://www.linkedin.com/posts/carmeloiaria_enterpriseai-agenticai-multiagentsystems-activity-7383114842636795904-shEK
16. https://github.com/Synaptic-Labs-AI
17. https://github.com/synaptics-synap
18. https://forum.cursor.com/t/cursor-rules-mdc-clarification/104879
19. https://www.linkedin.com/posts/addyosmani_ai-programming-softwareengineering-activity-7389186792706506752-BWnR
20. https://www.emergentmind.com/topics/agents-md-files
21. https://www.reddit.com/r/cursor/comments/1qjekug/consensus_on_using_actual_cursor_rules_mdc_vs/
22. https://github.com/Kylee7/Synaptic
23. https://www.aihero.dev/a-complete-guide-to-agents-md
24. https://www.youtube.com/watch?v=ABozvKmctkc
25. https://github.com/topics/ai-consulting
26. https://www.youtube.com/watch?v=W_RXEmy4N18