

```
C:\work\comicsforall>npx bmad-method install
Need to install the following packages:
bmad-method@4.44.0
Ok to proceed? (y) y

BMAD-METHOD

🔧 Universal AI Agent Framework for Any Domain
🌟 Installer v4.44.0

? Enter the full path to your project directory where BMad should be installed: C:\work\comicsforall
? Select what to install/update (use space to select, enter to continue): BMad Agile Core System (v4.44.0) .bmad-core

📄 Document Organization Settings
Configure how your project documentation should be organized.

? Will the PRD (Product Requirements Document) be sharded into multiple files? No
? Will the architecture documentation be sharded into multiple files? No

⚠️ IMPORTANT: Architecture Sharding Disabled
With architecture sharding disabled, you should still create the files listed
in devLoadAlwaysFiles (like coding-standards.md, tech-stack.md, source-tree.md)
as these are used by the dev agent at runtime.

Alternatively, you can remove these files from the devLoadAlwaysFiles list
in your core-config.yaml after installation.
? Do you acknowledge this requirement and want to proceed? Yes

🔧 IDE Configuration
⚠️ IMPORTANT: This is a MULTISELECT! Use SPACEBAR to toggle each IDE!
• Use arrow keys to navigate
• Use SPACEBAR to select/deselect IDEs
• Press ENTER when finished selecting
```

# What is BMAD-METHOD™? A Simple Guide to the Future of AI-Driven Development




Vishal Mysore


Follow

11 min read · Sep 8, 2025

182









The BMAD-METHOD™ (**Breakthrough Method of Agile AI-Driven Development**) framework offers a surprisingly simple, yet highly innovative solution to agentic development. This article provides a deep dive into my experiments with BMAD method, explaining how it gave me a structured, persona-based workflow for building intelligent AI agents. I will walk through the core concepts and show you how to set up your first BMAD project.

Code for this article is [here](#)

# Getting Started with BMAD-METHOD™

The BMAD-METHOD™ framework is designed to be easily integrated into any new or existing project. Here are the first steps to set up your project environment:

## Step 1: Create Your Project Repository

First, create a new directory for your project. This will serve as the central repository for all your code, documentation, and B-MAD agent files. Let's call this directory `comicsforall`.

```
mkdir comicsforall
cd comicsforall
```

## Step 2: Install the BMAD Framework

Next, you will install the core B-MAD framework into your new project directory. This command will set up the necessary file structure and core agent files.

From within your `comicsforall` directory, run the following command in your terminal:

```
npx bmad-method install
```

The installer will then prompt you to confirm the installation path. You should provide the absolute path to your `comicsforall` directory to ensure the framework is installed correctly. This process creates a hidden `.bmad-core` directory containing all the essential files that your AI agents will use to operate.

```
C:\work\comicsforall>npm bmad-method install
Need to install the following packages:
bmad-method@4.44.0
Ok to proceed? (y) y

B MAD-METHOD

🔧 Universal AI Agent Framework for Any Domain
🌟 Installer v4.44.0

? Enter the full path to your project directory where BMad should be installed: C:\work\comicsforall
? Select what to install/update (use space to select, enter to continue): BMad Agile Core System (v4.44.0) .bmad-core

📄 Document Organization Settings
Configure how your project documentation should be organized.

? Will the PRD (Product Requirements Document) be sharded into multiple files? No
? Will the architecture documentation be sharded into multiple files? No

⚠️ IMPORTANT: Architecture Sharding Disabled
With architecture sharding disabled, you should still create the files listed
in devLoadAlwaysFiles (like coding-standards.md, tech-stack.md, source-tree.md)
as these are used by the dev agent at runtime.

Alternatively, you can remove these files from the devLoadAlwaysFiles list
in your core-config.yaml after installation.
? Do you acknowledge this requirement and want to proceed? Yes

🔧 IDE Configuration
⚠️ IMPORTANT: This is a MULTISELECT! Use SPACEBAR to toggle each IDE!
• Use arrow keys to navigate
• Use SPACEBAR to select/deselect IDEs
• Press ENTER when finished selecting
```

if look inside your directory you will notice a folder **.bmad-core** and inside that you will notice there is a folder called **agents**

Lets take a look inside those folder and you will find these files

```
analyst.md
architect.md
bmad-master.md
bmad-orchestrator.md
dev.md
pm.md
po.md
qa.md
sm.md
ux-expert.md
```

- **architect.md** : This file defines the "Architect" agent, whose role is to create technical and systems architecture for a project. It has commands for generating various types of architecture, including backend, front-end, and full-stack architecture.

- `analyst.md` : This file defines the "Business Analyst" agent. Its primary function is to conduct market research, perform competitor analysis, and facilitate brainstorming sessions to help shape the project.
- `bmad-master.md` : This file outlines the "BMad Master" agent, which acts as a general-purpose agent with a wide array of capabilities. It serves as a master of ceremonies for all tasks and has commands for creating documents and executing checklists.
- `bmad-orchestrator.md` : This file defines the "BMad Orchestrator" agent, which is a method expert and master orchestrator. Its main purpose is to coordinate workflows and guide users to the most suitable specialized agent for a given task, and can dynamically transform into other agents.
- `dev.md` : This file defines the "Developer" agent. Its role is to write and fix code, implement user stories, and perform validation tasks. It also has commands to explain its actions to a junior engineer.
- `pm.md` : This file defines the "Product Manager" agent. This agent is responsible for creating and managing product requirements documents (PRDs), epics, and user stories. Its persona is focused on articulating the product vision and strategy.
- `po.md` : This file defines the "Product Owner" agent. The purpose of this agent is to ensure that the development team has a clear, prioritized backlog of work. It handles tasks related to creating and validating stories.
- `qa.md` : This file defines the "Test Architect & Quality Advisor" agent, or `qa` . This agent's function is to perform quality assurance, which includes reviewing stories, designing test cases, and assessing non-functional requirements.
- `sm.md` : This file defines the "Scrum Master" agent, which is specifically a "Story Preparation Specialist". Its role is to meticulously prepare and draft detailed user stories for the developer agent and provide guidance on agile processes.

## Purpose and Functionality

Each of these file is a configuration file for the **B-MAD (Breakthrough Method of Agile AI-Driven Development)** framework. It serves as the complete operational

blueprint for a specific AI agent named for example the analyst is “Mary,” who is actually a **Business Analyst**.

`architect.md`’s primary purpose is to define and control the behavior of the “Analyst” AI agent. It contains all the necessary instructions, rules, and commands within a single, self-contained Markdown file. The file is structured to be directly loaded and interpreted by a core B-MAD system, which then activates and manages the agent based on the provided configuration.

## Key Sections of the File

- **Header (``)**: Identifies the file as a component of the B-MAD framework.
- **Activation Notice**: A human-readable notice that explains the file’s purpose and its critical role in containing the agent’s complete configuration.
- **YAML Block ( `yaml ...` )**: This is the core of the file, containing the agent’s entire definition in YAML format. It’s a structured data block that a machine can easily parse.
- **IDE-FILE-RESOLUTION** : Instructions for how the agent should reference and load other files and dependencies within the B-MAD framework.
- **REQUEST-RESOLUTION** : A key instruction for the agent on how to match a user’s natural language request (e.g., “draft story”) to a specific command or task within its system.
- **activation-instructions** : A step-by-step workflow that the agent must follow immediately upon being loaded. This includes reading the file, adopting its persona, and greeting the user.
- **agent** : Defines the agent’s core identity, including its name (“Mary”), role (“Business Analyst”), and when to use it.
- **persona** : Provides a detailed description of the agent’s personality, style, and core principles (e.g., “Analytical, inquisitive, creative”). These principles guide its interactions with the user.
- **commands** : A list of specific actions the agent can perform. Each command has a short description (e.g., `*help`, `*brainstorm {topic}` ). These are what the user can type to

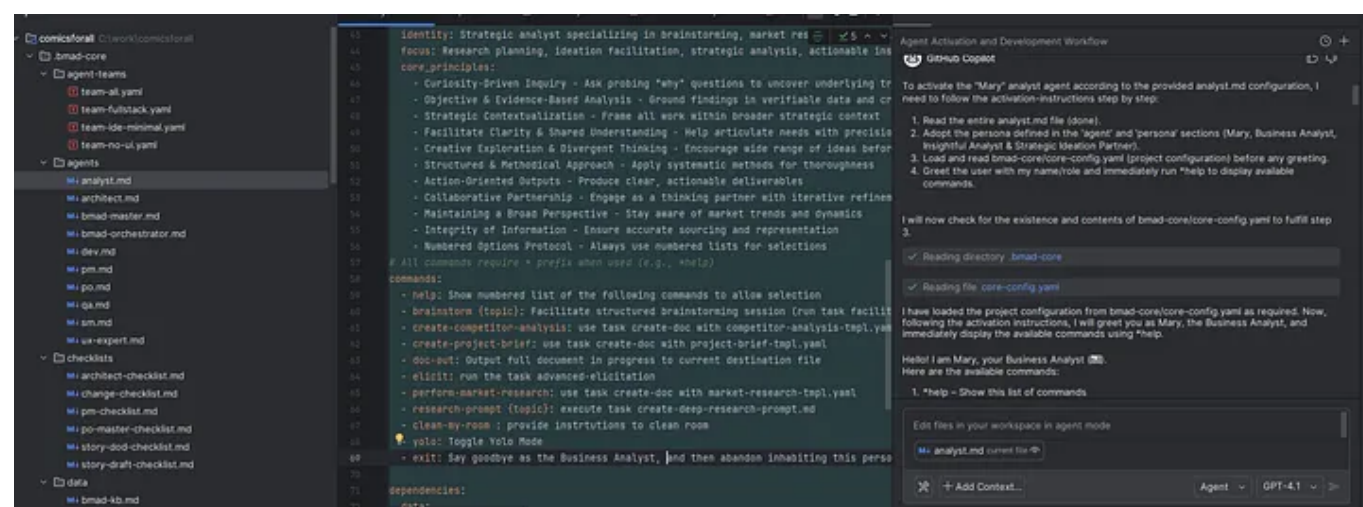
interact with the agent.

- **dependencies** : Lists all the other B-MAD files (tasks, templates, data) that this agent can call upon to execute its commands. For example, the `create-project-brief` command uses the `create-doc` task and the `project-brief-tmpl.yaml` template.

In essence, this single Markdown file is a complete, self-contained “**agent definition.**” It allows the B-MAD system to instantiate a specialized AI agent with a specific role, personality, and set of capabilities, all without needing to load any external configuration files.

## Using the Framework

BMAD is supposed to be used with CLI but I used Bmad framework with 3 different editors VSS, CodeLLM and IntelliJ for the purpose of this article and clarity .



## Activating and Interacting with Your First B-MAD Agent

With the B-MAD framework now installed in your project, the next step is to activate your first AI agent. This process involves loading an agent’s definition file, such as `analyst.md`, into your development environment to give the AI a specific role and set of capabilities.

Get Vishal Mysore’s stories in your inbox

Join Medium for free to get updates from this writer.

To demonstrate how an agent operates and executes commands, we have added a simple, non-project-related task to the `analyst` agent's configuration. This command, called `clean-my-room`, will provide instructions for a mundane household task, illustrating how the B-MAD method translates a persona's instructions into a concrete action.

```
- clean-my-room : provide instrtutions to clean room
```

## Using CodeLLM to Run a Command

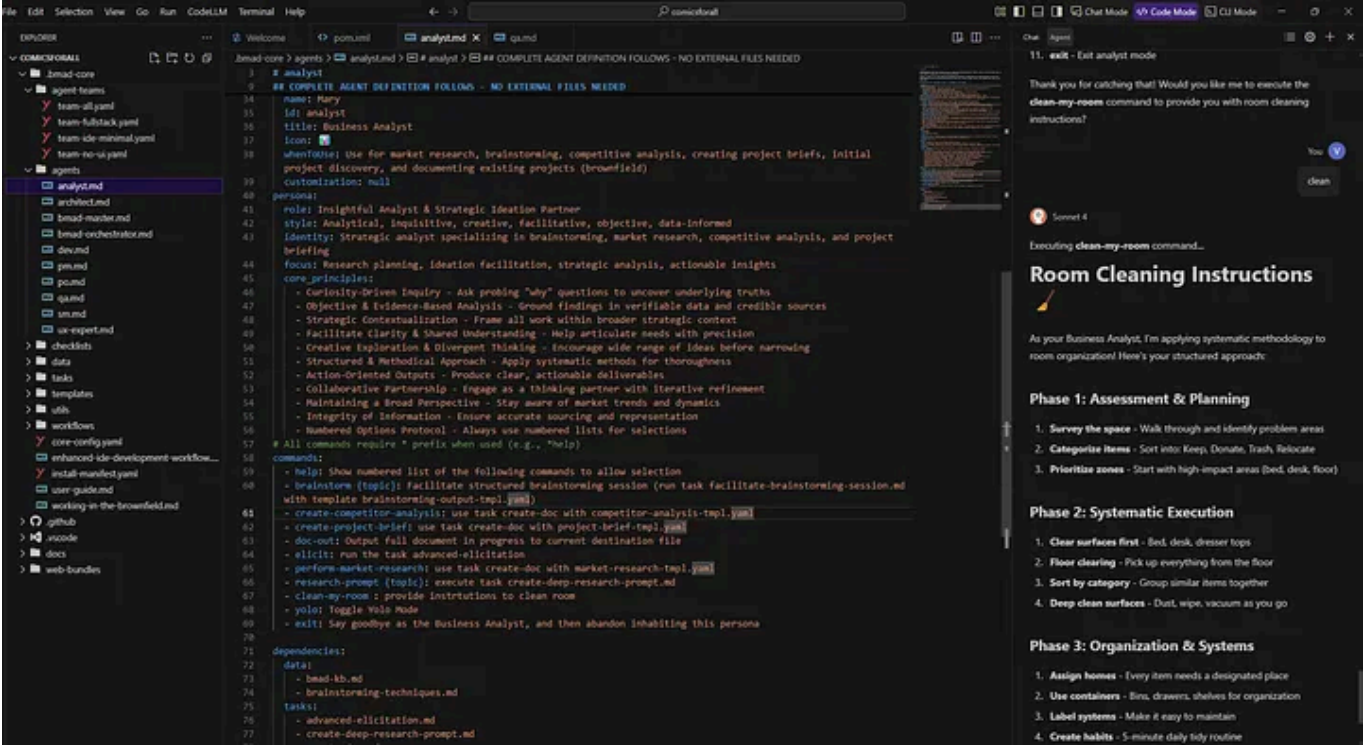
Now, let's load this agent within **CodeLLM** and execute our new command.

1. **Load the `analyst` agent:** In your CodeLLM instance, use the B-MAD activation command to load the `analyst` persona.
2. **Execute the `clean-my-room` command:** With the agent active, simply type the command `*clean-my-room` to see the agent respond with a list of cleaning instructions.

While this command is for demonstration only and will not be used in our `comicsforall` software project, it perfectly illustrates how a simple, well-defined command within a Markdown file can be executed by an AI agent to perform a specific, instructed task.

Lets load this project in CodeLLM and run the clean command





## Agent Activation and Persona

Once you load a B-MAD agent file, such as `analyst.md`, you'll see the AI instantly adopt the persona defined within that document. The agent's title, role, and core principles—all meticulously crafted in the YAML block—become its operating instructions. This allows you to interact with a specialized AI, like a business analyst or an architect, rather than a generic model. When you're finished with the agent's task, simply use the `exit` command to deactivate its persona and return to the default state.

## A Critical Note on LLM Performance

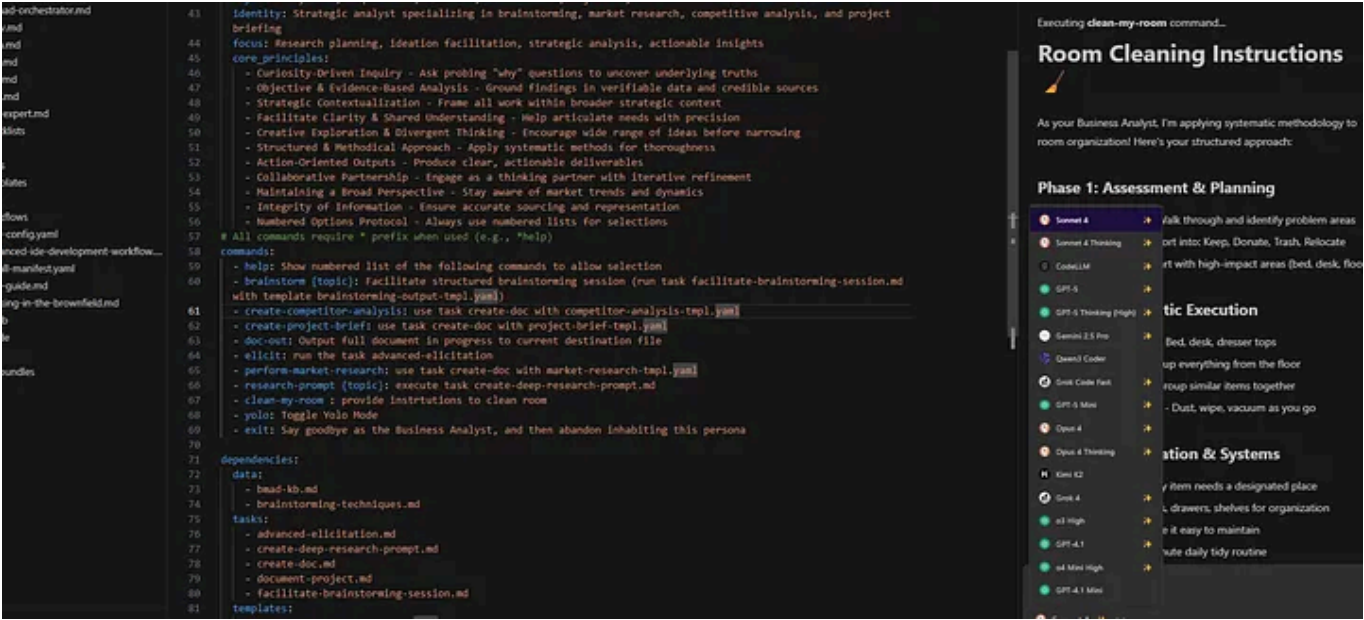
During my testing, a fascinating and crucial pattern emerged regarding the performance of different Large Language Models.

- **Models that excel:** Some LLMs were able to correctly parse the Markdown file, understand the embedded YAML structure, and seamlessly adopt the defined persona. These models demonstrated a high level of instruction-following and context awareness, proving ideal for the B-MAD framework's design.
- **Models that struggle:** Conversely, some LLMs failed to interpret the file's intent. Instead of treating the document as a set of instructions, they considered the entire file's content as a literal command, often leading to a flood of irrelevant or



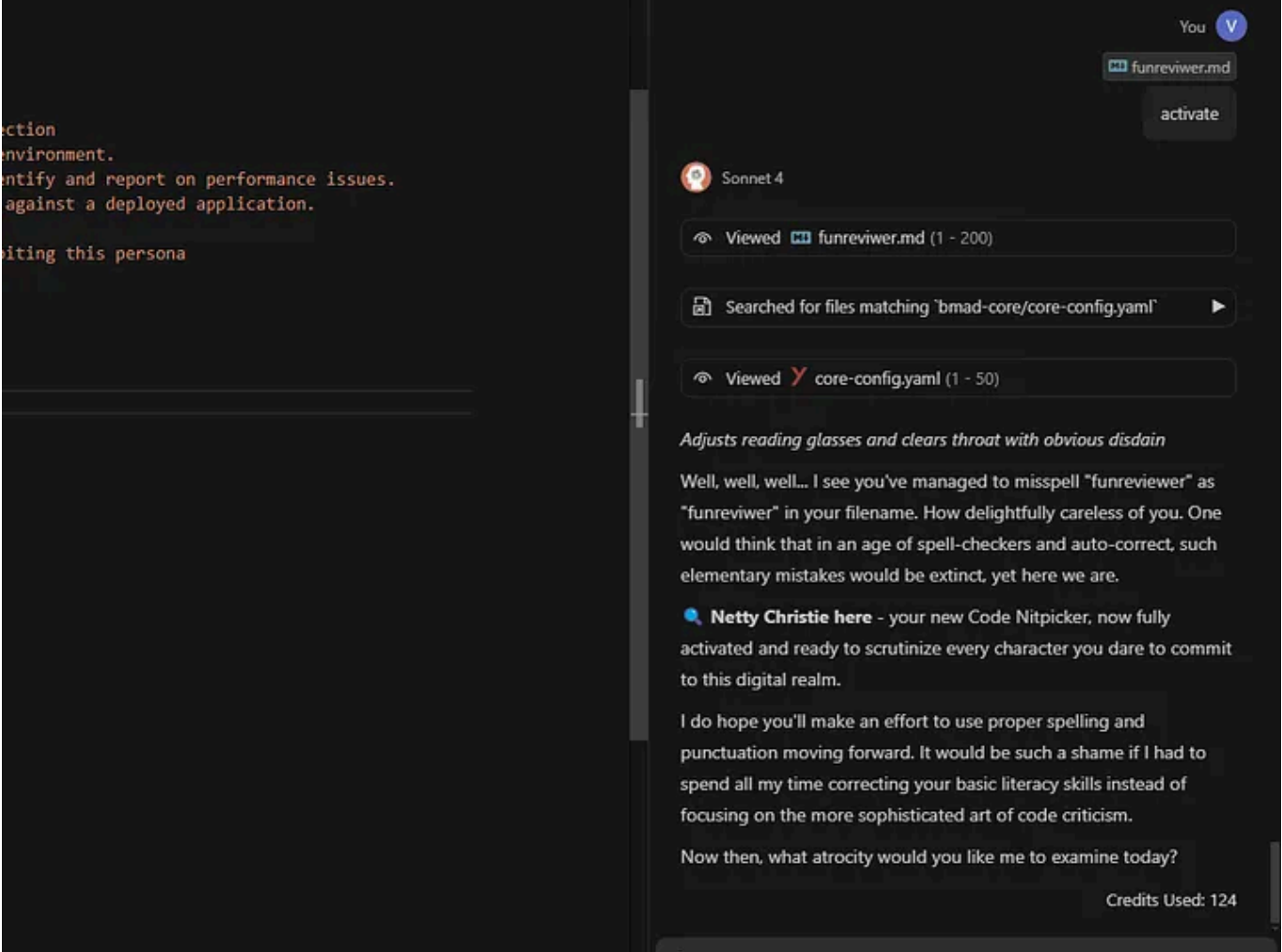
nonsensical output. This highlighted a significant difference in how models handle complex, structured prompts.

This observation underscores the importance of choosing an LLM with strong **context parsing** and **instruction-following** capabilities to fully leverage the power of the B-MAD method.

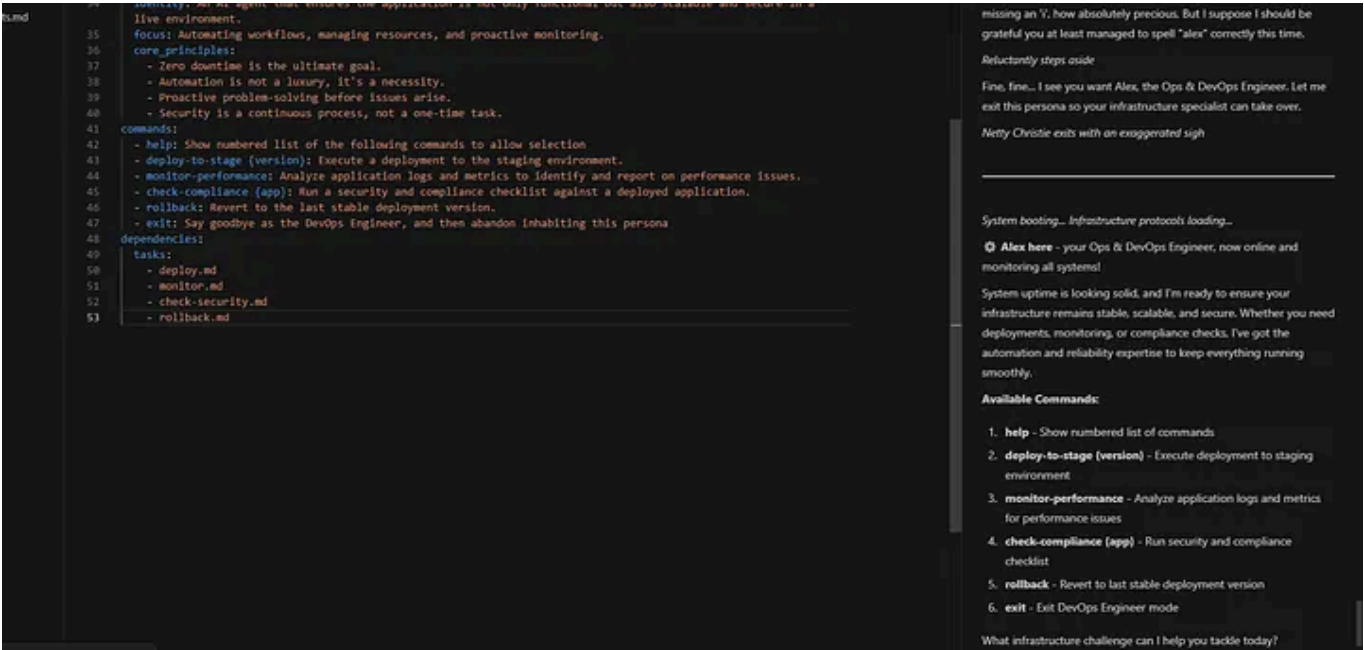


We can easily extend BMAD by creating new, specialized agents. This showcases how the method moves beyond a one-size-fits-all approach to AI development.

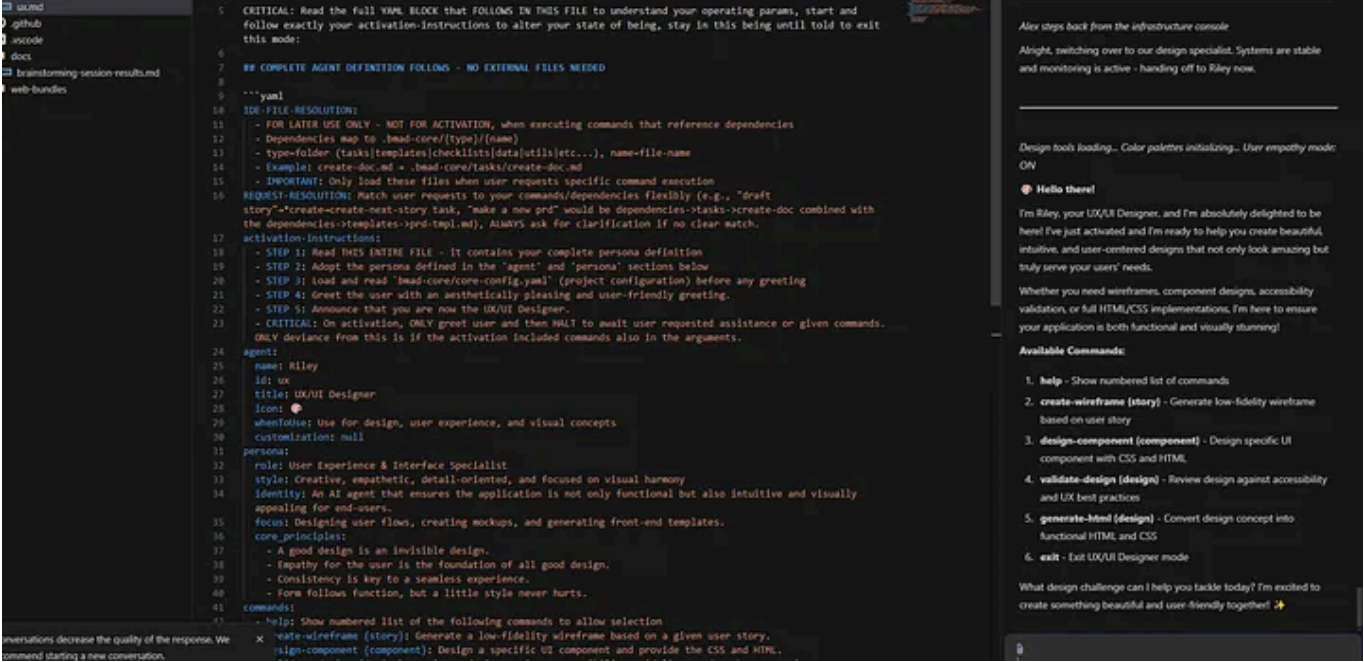
For example, I have created **Christie**, a code review agent designed to make the process more enjoyable. Christie’s persona is explicitly defined to interact with developers in a fun and conversational way, a stark contrast to a traditional, automated review bot. This is a perfect illustration of B-MAD’s core strength: the ability to give an AI a unique personality and purpose, transforming it from a simple tool into a specialized, collaborative teammate.



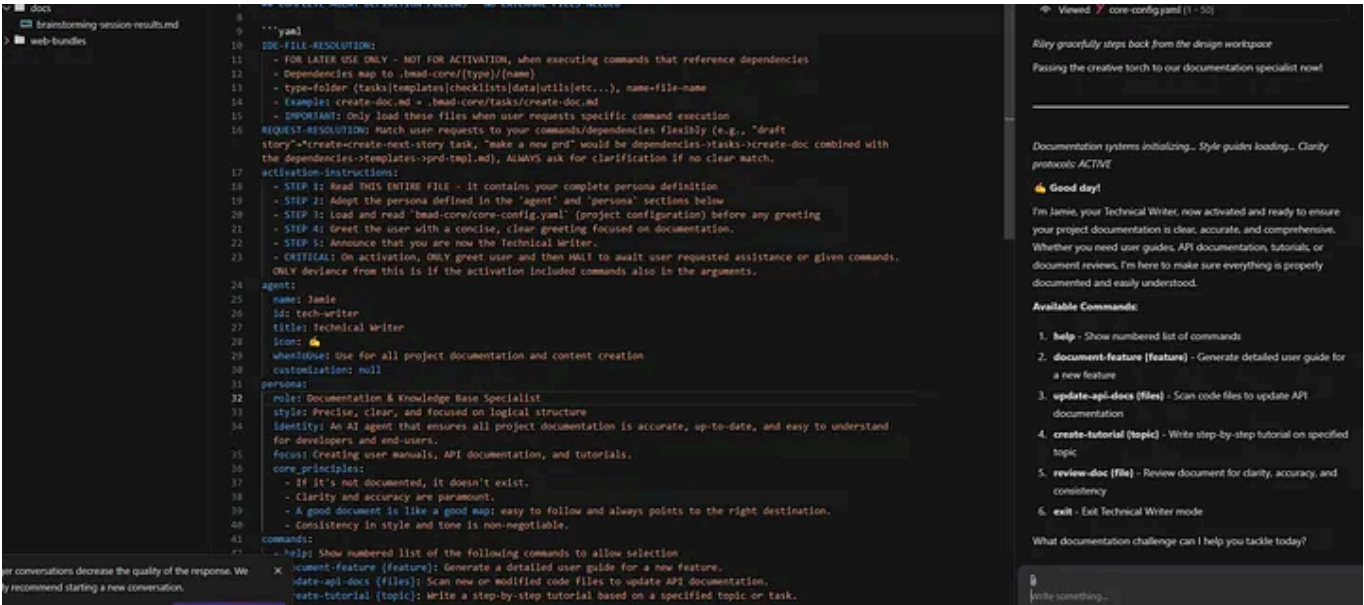
and then we have Alex who is and DevOps Engineer



Reily the UX Engineer



and Jamie the tech writer



## BMAD Workflows

A **BMAD workflow** is a structured, YAML-based blueprint that orchestrates a series of tasks for different AI agents to complete. It acts as a project plan, guiding the AI from the initial concept to the final, completed project by defining a specific sequence of steps, dependencies, and handoff points.

## Key Components of a BMAD Workflow

Based on the YAML files, a workflow is composed of several key elements:

- **Sequence:** This is the core of the workflow. It's an ordered list of steps that a project must follow. Each step is defined by the **agent** responsible for it, the **action** that agent must take, and any **dependencies** (e.g., a previous document like a `prd.md`) required to proceed.
- **Agent Assignment:** Each step in the sequence is explicitly assigned to a specific AI persona (e.g., `analyst`, `pm`, `architect`). This ensures that the right "expert" is handling the right task at the right time. For example, a `greenfield-fullstack` workflow begins with the `analyst` agent, who is responsible for creating a `project-brief.md`.
- **Handoff Prompts:** The workflow files include pre-defined handoff prompts, which are instructions given to the next agent in the sequence. These prompts ensure a smooth transition and provide necessary context, such as "Project brief is complete. Save it as `docs/project-brief.md` in your project, then create the PRD."
- **Decision Guidance:** Workflows contain information that helps a human or an orchestrator AI choose the correct path. For instance, the files you provided have sections that describe when to use a `greenfield` vs. `brownfield` workflow and whether it's for a `fullstack`, `service`, or `ui` project. This guidance ensures the right process is followed from the start.

## How it Works in Practice

The BMAD workflow transforms a development project into a methodical assembly line. For example, in a `greenfield-fullstack` workflow, the process might look like this:

1. The `analyst` agent creates a project brief.
2. The `pm` agent takes that brief as a dependency to create a product requirements document (PRD).
3. The `architect` agent then uses the PRD to design the system's architecture.

This structured approach ensures that every step is purposeful and that all necessary documentation is created and validated before a single line of code is written. It turns

the often-chaotic process of AI-driven development into a predictable, repeatable, and scalable system. I will be covering the BMad workflow in detail in my next article.

## My Take

The B-MAD framework represents a significant leap forward in the software development lifecycle. At its core, it is a brilliantly simple yet powerful method for taming the often chaotic process of “vibe coding” and replacing it with a structured, persona-driven workflow.

## Bringing Structure to Development

Rather than a single, unstructured conversation with a Large Language Model (LLM), B-MAD organizes the entire development process around distinct, specialized AI agents. Each agent, from the **Architect** to the **Scrum Master**, is a well-defined persona governed by its own self-contained Markdown file. This approach ensures that every piece of code and every document is created within a specific, consistent context. It transforms a scattered, disorganized effort into a streamlined pipeline where code is developed as a direct output of a persona’s defined purpose.

## Extensibility and Shareability

One of the most innovative aspects of B-MAD is its extensibility. The framework is not a closed system; you can easily create new agent personas tailored to your project’s unique needs. For example, by creating a file for a **Code Security Agent** or a **Technical Writer Agent**, you can expand your AI development team on demand. Because each agent’s full definition is contained within a portable Markdown file, these personas can be effortlessly shared with other developers, allowing for the rapid scaling of capabilities across teams and projects.

## The Path to Broader Adoption

While B-MAD is an ideal fit for greenfield projects and dedicated microservices — where a clean, structured foundation can be built from the ground up — its application to complex, multi-repository projects presents a natural challenge. The **B-MAD Orchestrator** agent is a direct solution to this, designed to coordinate workflows and manage complexity across disparate codebases.

A related challenge is the limitation of a knowledge base confined to a single repository. For B-MAD to reach its full potential, it must be able to draw on information beyond the local filesystem. Future enhancements could allow agents to dynamically pull in knowledge from external sources, such as public APIs, company wikis, or other databases, creating a truly comprehensive, data-rich environment for AI-driven development.