The research confirms that **Zencoder** and **GitHub Copilot** use different mechanisms for "governance," but they can be unified into a single workflow.

To solve this "once and always," you cannot rely on developers remembering to follow rules. You must **productize the process** by creating a set of "Actionable Tickets" that you can literally copy-paste into Jira, Linear, or GitHub Issues.

Here is your **Implementation Sprint: The Agentic Governance Framework**.

Epic: Agentic Governance Rollout

Goal: Transition from "Vibe Coding" (chaotic, human-dependent) to "Agentic Engineering" (structured, automated, verified).
Definition of Done: A repository where AI agents automatically follow architectural rules, tests are mandatory before implementation, and bad code is physically blocked from commits.

Ticket 1: Establish the "Constitution" (Single Source of Truth)

Priority: Critical / P0
Assignee: Tech Lead / Architect
Description:
AI agents (Zencoder & Copilot) currently lack shared context. We need a single "Constitution" that enforces our tech stack and security rules across both tools.
Implementation Steps:

1. Create a root directory: .governance/.
2. Create the master file: .governance/CONSTITUTION.md.
3. **Content to Insert:**

   Engineering Constitution

   1. **Stack:** Next.js 14 (App Router), Tailwind, TypeScript (Strict).
   2. **Security:** No hardcoded secrets. Zod validation for ALL inputs.
   3. **Testing:** Tests MUST be written before code (Red-Green-Refactor).
   4. **Style:** Functional components only. No any types.
4. **Sync to Zencoder:**
   - Create .zencoder/rules/00-core.md.
   - Add Frontmatter to force it into context:
     YAML
     ```
     ---
     description: "Core Project Constitution - ALWAYS APPLY"
     alwaysApply: true
     ```

```
---
(Import content from.governance/CONSTITUTION.md)
```

5. **Sync to Copilot:**
   - Create .github/copilot-instructions.md.
   - Add: *"You must strictly adhere to the engineering constitution below..."* followed by the content.

Acceptance Criteria:

- [ ] Asking Zencoder "What tech stack do we use?" returns the correct stack defined in .governance.
- [ ] Asking Copilot "Generate a component" produces code that matches the strict style guide (e.g., no any types).

Ticket 2: Configure Zencoder "Flow" Agents

Priority: High / P1
Assignee: Senior Dev
Description:
Stop using the generic "Ask" agent for everything. We need specialized agents that force "Flow Engineering" (Plan -> Test -> Build).
Implementation Steps:

1. **Run Repo-Info:** Run the Zencoder command /repo-info to generate the .zencoder/rules/repo.md context map.
2. **Create "The Architect" Agent:**
   - Go to Zencoder > Custom Agents > Add.
   - **Name:** Architect
   - **Prompt:** *"You are a System Architect. Your goal is to PLAN, not code. Read .zencoder/rules/00-core.md. When given a feature request, output a Markdown file specs/FEATURE_NAME.md detailing the files to create, data schemas, and edge cases. Do not generate implementation code."*
3. **Create "The TDD Builder" Agent:**
   - **Name:** TDD Builder
   - **Prompt:** *"You are a Test-Driven Engineer. 1. Read the spec provided. 2. Create a FAILING test file first. 3. Wait for user confirmation. 4. Write the minimum code to pass the test. 5. Refactor."*

Acceptance Criteria:

- [ ] The Architect agent refuses to write code and instead writes a Plan/Spec file.
- [ ] The TDD Builder agent creates a .test.ts file before creating the .ts implementation file.

Ticket 3: Automate the "Red-Green" Loop (VS Code Tasks)

Priority: Medium / P2
Assignee: Dev
Description:
Manually running tests and copying error messages to the AI is slow. We need a "One-Click" loop to feed errors back to Zencoder/Copilot.
Implementation Steps:

1. Update .vscode/tasks.json with the "Test & Copy" task.
2. **Config Snippet:**
   JSON
   ```json
   {
     "label": "🔄 AI Loop: Test & Copy",
     "type": "shell",
     "command": "npm test -- --reporter=json > test-output.json |
   ```

| (cat test-output.json | pbcopy && echo '❌ Copied errors to clipboard!')",
"presentation": { "reveal": "always", "panel": "dedicated" }
}

(Note: Use clip.exe for Windows)
Acceptance Criteria:

- [ ] Running the task runs the project tests.
- [ ] If tests fail, the JSON error log is automatically in the user's clipboard, ready to paste into Zencoder.

Ticket 4: "Hard Governance" Gatekeeper (Husky)

Priority: Critical / P0
Assignee: DevOps / Lead
Description:
AI agents hallucinate and get lazy (e.g., leaving TODO or any). We must physically prevent this code from entering the repo.
Implementation Steps:

1. Install Husky: npm install husky --save-dev && npx husky install.
2. Create hook: .husky/pre-commit.
3. **Script:**
   Bash
   ```bash
   #!/bin/sh
   ```

```
. "$(dirname "$0")/_/husky.sh"



# 1. Block Lazy AI
if grep -r "TODO" src/ |

| grep -r "FIXME" src/; then
echo "🤖 Governance Protocol: Lazy AI detected (TODO/FIXME). Rejecting."
exit 1
fi



# 2. Block Hallucinated Types
if grep -r ": any" src/; then
  echo "🤖 Governance Protocol: Strict Type Violation ('any'). Rejecting."
  exit 1
fi

# 3. Verify Logic
npm test
```

Acceptance Criteria:

- [ ] Try to commit a file with // TODO. The commit MUST fail.
- [ ] Try to commit a file with something: any. The commit MUST fail.
- [ ] Try to commit code that breaks tests. The commit MUST fail.

Ticket 5: "One Time" Project Setup Script (The Automation)

Priority: Low / P3 (Optional but recommended)
Assignee: DevOps
Description:
To ensure this works "Always" for every new project, create a scaffolding script.
Implementation Steps:

1. Create a script init-agentic.sh.
2. The script should:
   - Generate the .zencoder and .github folder structures.

    - Write the default CONSTITUTION.md files.
    - Install Husky and write the hooks.
    - Add recommended VS Code extensions (Zencoder, Copilot) to .vscode/extensions.json.

Acceptance Criteria:

- [ ] Running npm run init-agentic in a blank folder sets up the entire governance framework automatically.

How to Execute This Plan

1. **Sprint 1 (Days 1-2):** Complete **Ticket 1 (Constitution)** and **Ticket 4 (Husky)**. This stops the bleeding immediately. No bad code gets in.
2. **Sprint 1 (Days 3-4):** Complete **Ticket 2 (Zencoder Agents)**. This changes the workflow from "chatting" to "engineering."
3. **Sprint 1 (Day 5):** Complete **Ticket 3 (VS Code Tasks)**. This speeds up the TDD loop.

This transforms your VS Code from a simple text editor into a **Governed AI Factory**. The "Constitution" creates the rules, the "Agents" follow the process, and "Husky" acts as the quality assurance guard.