To implement the Agentic Governance Framework in Visual Studio Code using your specific stack (GitHub Copilot Student, Zencoder, and Augment Code), you need to configure these tools to act as a coordinated, self-verifying system. Instead of competing, these tools can be layered: Copilot for fast inline completions, Zencoder for Spec-Driven Development (SDD) and codebase mapping, and Augment Code for deep architectural reasoning and CI/CD automation.

Here is your step-by-step implementation process to achieve production-ready, human-level code.

Step 1: Establish the "Constitution" (Context & Rules)

You must translate your coding standards into persistent files that automatically load into the AI's context window. You will configure rule files for all three tools in your project root.

1. GitHub Copilot Instructions:

Create a .github/copilot-instructions.md file. Copilot will automatically append this to every chat request.

- Structure it with clear headings: Tech Stack, Project Structure, and strict Coding Guidelines (e.g., "Always add JSDoc comments," "Never use any in TypeScript").
- For granular control, create path-specific files like .github/instructions/frontend.instructions.md with YAML frontmatter applyTo: "src/web/**/*.tsx" to prevent context pollution.

2. Augment Code Rules:

Create a .augment/rules/ directory. Augment supports advanced YAML frontmatter to control when rules are injected.

- Create files like .augment/rules/security.md and set the frontmatter type: Always for non-negotiable security standards.
- Create .augment/rules/database.md and set type: Auto. Augment will only load this when it detects database-related tasks in your prompt, saving context space.

3. Zencoder "Zen Rules":

Zencoder allows you to map rules to specific file globs.

- Create .zencoder/rules/api-standards.md.
- Add frontmatter:
  YAML
  ```yaml
  ---
  description: "API endpoint standards"
  globs: ["**/api/*.ts"]
  ---
  ```

- List your exact requirements (e.g., "Add input validation," "Include error handling").

Step 2: Build the Semantic Codebase Map

Before writing new code, you must ensure your agents deeply understand your existing architecture to prevent hallucinations.

- **Zencoder Repo Grokking:** Open VS Code, press Cmd+. (Mac) or Ctrl+. (Windows), and select the **Repo-Info Agent**. Type: Generate repo.md file with information from my repo. This analyzes your dependencies, build systems, and architecture, saving a persistent .zencoder/rules/repo.md file that all Zencoder agents will read before acting.
- **Augment Context Engine:** Augment automatically indexes your repository locally and understands relationships across up to 400,000 files. Connect Augment to your external tools (like Jira or GitHub) via the settings panel so it can read issue tickets directly.

Step 3: Implement Spec-Driven Development (Flow Engineering)

Do not prompt the AI to "build a feature." Instead, force it through a structured workflow.

1. **Specify:** Create a Markdown file in a .zencoder/specs/ folder detailing the user story, measurable success criteria, and strict constraints.

2. **Plan:** Use the Zencoder Coding Agent and prompt it: "Based on .zencoder/specs/my-feature.md and our codebase analysis, create a technical implementation plan including architecture and database schema.".

3. **Tasklist Execution:** Use the **Augment Agent**. Point it to the plan and let it generate a "Tasklist". The Augment Agent will map the full sequence (e.g., update middleware -> add logic -> write tests) before it touches any code.

Step 4: Automate the "Immune System" (AI-TDD & Self-Correction)

Leverage the autonomous capabilities of your tools to validate code immediately.

- **Test Generation:** Before implementing a task from your plan, highlight the target file, press Cmd+., and select Zencoder's **Unit Testing Agent**. Prompt it: "Generate unit tests for this specification.".
- **Agentic Repair:** When the implementation fails a test, do not manually copy-paste the error. Use the Augment Agent's **Terminal Execution** feature. Run npm test directly in the terminal; Augment reads the output, spots the failure, and automatically rewrites the code to fix it without your intervention.

Step 5: Hard Governance in CI/CD (The Auggie CLI)

To ensure production readiness, governance must extend beyond VS Code into your automated pipelines.

- Install the Augment CLI via npm install -g @augmentcode/auggie.
- Integrate auggie into your GitHub Actions or CI/CD pipeline.
- Configure an automated PR review step: auggie --print "Analyze this PR against our security and architectural rules. Fail the build if circular dependencies or uncaught exceptions are introduced.".

Summary Workflow Matrix

Phase	Tool Used	Action	Purpose
Governance	Copilot / Augment / Zencoder	Configure .github/copilot-instructions.md, .augment/rules/, and .zencoder/rules/	Enforce standards globally and by file-type.
Context	Zencoder	Run **Repo-Info Agent**	Generate repo.md to map project architecture.
Planning	Zencoder	Write Spec -> Generate Plan	Stop hallucinations by defining exact architecture first.
Execution	Augment Code	Run **Tasklist** & **Terminal Execution**	Multi-file edits and automated test-fixing in the terminal.
Validation	Auggie CLI	Trigger GitHub Action	Automated PR review against custom project rules.