ChatGPT

# Agentic AI-First VS Code IDE: A Role-Centric, Lifecycle-Oriented UX

An AI-powered IDE for enterprise delivery must serve diverse stakeholders (Product Owners, Reviewers, QA, DevOps, etc.) by centering **AI agents** as the primary interface and seamlessly spanning every delivery phase (Discover → Spec → Scaffold → Implement → Verify → Operate). We propose a VS Code–based IDE reimagined around **role-specific workspaces**, conversational agents, and interactive project dashboards. This design ensures that non-engineers (e.g. POs, architects, reviewers) can drive workflows without writing code, while engineers still have full transparency and control. Throughout, all actions link back to ADF artifacts for full traceability and audit.

## Role-Driven Workspaces & AI Personas

- **Custom Agent Personas.** The IDE provides a **dropdown of AI agents** tailored to roles or tasks. For example, a "Planning Agent" for Product Owners might summarize requirements and draft an implementation plan, while a "Review Agent" for architects scans code for security issues. VS Code supports defining custom agents via `.agent.md` files [1], allowing each persona to embed role-specific instructions and tools. By selecting an agent (e.g. Composer, Checker, Integrator), the user automatically enters a mode with appropriate permissions and UI layout.

- **Non-Engineer Focus.** Product Owners and stakeholders see a **"Story/Spec Canvas"** rather than raw code. This might include a rendered *Spec Editor* pane with rich-text or wizard-like interfaces (leveraging Speckit for spec templates and linting [2]). A chat panel next to it lets the PO **converse with the AI agent** ("brainstorm user stories", "refine acceptance criteria") in natural language. The agent updates the spec or backlog accordingly, citing spec sections as needed (ADF requires every story to link back to a spec clause [3]). Reviewers might open a **"Review Workspace"** that shows pull-request diffs alongside an AI chat agent highlighting risks (e.g. security code scan results) and summarizing changes.

- **Adaptive Layouts.** The IDE supports **workspace profiles** (e.g. "PO Mode", "Dev Mode", "QA Mode") that rearrange panels. For instance, PO Mode might place the *Spec Editor* and *Kanban board* front-and-center, while Developer Mode shows code editors and test logs. Users can also split screens: e.g. code + chat, or timeline + terminal. This aligns with how VS Code lets you place views in the sidebar, panel, or full editor. Teams can save custom layouts per role or preference, ensuring familiar workflows (keyboard shortcuts and panels) tailored to **agentic vs. code-centric tasks**.

## Full Lifecycle Flows

We break the user journey into ADF phases, each with a distinct UX:

## 1. Discover (Ideation & Planning)

- **AI-Facilitated Backlog Creation.** A **"Discovery Board"** workspace lets POs sketch business goals. An embedded Kanban view (akin to Kaiban's agentic board [4]) shows columns like *Ideas*, *Approved*, *Spec*, etc. Users can drag an "Idea" card to *Spec*, triggering an AI planning workflow.
- **Agentic Brainstorming.** The user types in a natural prompt ("Define features for invoice reporting"), and a **Vision Agent** (Composer-like role [5]) generates a high-level requirements outline. The output fills the Spec editor or story template. All revisions are automatically linked to the relevant spec clause as metadata (ADF mandates citing the spec in each story [3]).

## 2. Spec (Definition & Approval)

- **Interactive Spec Editor.** Specs (e.g. YAML or Markdown) are edited in VS Code with *Speckit* support: linting, live preview, and a Requirements Traceability Matrix (RTM). The UI could show a **split view**: raw spec text on one side, and generated documentation/RTM on the other [2]. An AI **"Spec Agent"** can run on demand to suggest improvements or generate missing sections. For example, the PO asks "Are all user roles covered?" and the agent updates the spec.
- **Live Drafts & Templates.** Built-in and repo-local spec templates (Speckit templates) are selectable via a command palette or dropdown. Custom **wizard panels** could step non-technical users through spec fields (e.g. entering acceptance criteria, security requirements, performance budget).
- **Governance Gates (Pre-commit).** The UI highlights required checks (spec-verify, mode-policy, etc.) and runs them as the user works, surfacing any blocking issues. Results (e.g. coverage %, static-scan findings) appear in a **Verification Panel**, and the links to evidence (e.g. `requirements-trace.json`) are shown for transparency [6]. The agentic interface keeps users aware of ADF compliance (e.g. flagging missing CODEOWNERS or spec references) before code is even written.

## 3. Scaffold (Project Generation)

- **Template-based Setup.** Upon spec approval, the user triggers scaffolding (via command or AI prompt). For example, "Initialize new secure-web-app project." An AI **Scaffold Agent** (Conductor role) drives the *Speckit init/build* process, choosing frameworks and creating folder structure. Progress is shown in an interactive **Taskview panel**, where each step (install libs, setup CI configs) is a checklist item.
- **Visualization of Architecture.** The UI could graphically display the generated architecture (using Mermaid or draw.io), with components derived from the spec (e.g. service APIs, data models). This helps POs and architects verify the initial design.
- **Traceable Artifacts.** All scaffolded files (README, initial code) include annotations linking back to spec items. Speckit's automated docs (RTM.md, evidence metadata) are committed to the repo, and the UI can display a **Traceability Navigator** showing how requirements flow into code modules [2].

## 4. Implement (Coding & Testing)

- **Code Editor + AI Chat.** The main coding environment remains VS Code, but with tight AI integration. An inline **AI chat pane** (like Copilot Chat) is always available. Users can highlight code or tests and ask the agent questions (e.g. "Why did this test fail?" or "Generate function to parse X"). Agents like Performer/Checker from Symphony can be invoked via commands (e.g. `/review`, `/fix`) to either generate code or point out issues [7].

- **Continuous Guidance.** A persistent **Coach/Insight view** monitors your edits: it shows metrics (Requirement Coverage, Tool Precision, Backtracking) and suggests best practices (based on Speckit hints). If the user's edits drift from spec, the AI highlights the divergence. For example, as code is written, the AI might warn "This change exceeds scope by 20% (Edit Locality check) – consider breaking into a new story," aligning with ADF's `mode-policy` gate.
- **Embedded REPL & Previews.** The layout can include live previews of UI components or API docs. For front-end stories, one panel might run a local dev server to show the UI, guided by the UX Designer agent. For APIs, a swagger-like preview can open. This ties into rapid feedback in the "Implement" phase.

### 5. Verify (Review & Gate Checks)

- **Story Preview Workspace.** Before merging, the IDE offers a **"Story Preview" tab** summarizing the changes: updated documentation, screenshots, demo links, and gate results. Product Owners and stakeholders can navigate the Story Preview and **accept or comment**. As mandated by ADF, the PO must explicitly "accept" this preview to unblock merge [8]. The UI provides a big " Accept Preview" button once all conditions are met.
- **Automated Review Agents.** AI Reviewers automatically run on the code: the Security Specialist agent flags vulnerabilities in one pane, the Perf agent checks performance budgets, etc. Results feed into the pull request so that a human reviewer sees both the automated report and AI commentary side by side.
- **Audit Trail Panel.** A dedicated **Evidence Inspector** shows all gate logs, SARIF reports, and SBOM files as a timeline or list. Each item in the Evidence Bundle is clickable (e.g. "spec-verify passed ✓" opens the schema linter output). This makes audit artifacts directly explorable in the IDE [6]. Handoff packets for multi-agent work (if any) are likewise viewable as Markdown or diagrams, ensuring continuity per ADF SSP rules.

### 6. Operate (Deploy & Monitor)

- **Deployment Dashboard.** After merge, a new **Operations Workspace** is available. It shows the deployment status (e.g. CI/CD pipeline stages, environment statuses). An AI Ops agent (DevOps role) can answer queries like "When was last deploy?", "Scale up the service cluster," or "Show service latency graphs." Real-time logs or metrics from monitoring tools are embedded via panels (e.g. Grafana iframe, AWS/Azure dashboards).
- **Pulse & Metrics View.** The IDE integrates the ADF **Delivery Pulse**: a global view of team metrics (story lead time, trust score, SOPC compliance). A periodic refresh shows how many stories shipped this increment and agent success rates, reinforcing governance.
- **Incident Assist.** If alerts fire, a **ChatOps Panel** with an AI agent helps investigate (search logs, suggest fixes) while recording a remediation plan. All such actions are appended to the project's Evidence Bundle for traceability.

## UI Metaphors & Layout Innovations

- **Integrated Kanban/Timeline.** Beyond sidebars, the IDE includes a **full-screen Kanban or timeline view** of the SSP. Stories flow across columns (Spec, Code, Review, Done) and AI agents appear as "avatars" on tasks, indicating who is working (or which agent is active) [4]. Clicking a card opens a dialog with chat history, requirements, and handoff notes. This visual workflow ensures users see progress at a glance.

- **Contextual Workflows.** Users can launch tasks via natural-language commands or palette entries (e.g. "Generate API endpoint from spec," "Run security scan"). The UI then automatically opens the relevant panels and agents. For example, invoking **"Start Code Review"** opens a diff view and the Checker agent chat together.
- **Drag-and-Drop Composition.** In Spec or planning mode, users can drag pre-defined "policy check" or "test plan" cards onto features, automatically wiring in AI prompts to cover those concerns. This lets POs configure gating without manual YAML edits.
- **Peek/Augment.** Similar to "Peek Definition," one could **peek AI notes** on any function or requirement. Hovering on a spec item might show the last AI comment or rationale ("[AI Agent]: This requirement traces to security control SC-21 [6] ").

## Integration with Speckit & ADF Tooling

- The IDE embeds **Speckit commands**: e.g. a "Generate RTM" button runs `speckit gen`, and the RTM appears in a split editor. Speckit's verification step (`speckit verify`) can run on save, highlighting spec drift.
- **Templates and Compliance.** When selecting templates (Speckit's `init --mode secure` presets), the UI shows recommended compliance overlays (HIPAA, etc.) as toggles. The AI agent explains implications as the user picks options.
- **Policy Gates in UI.** The IDE surface enforces mode/preset guards: it can gray out disallowed operations (e.g. "You cannot commit secrets to this branch" pop-up if policy triggers [6] ). Agents act as compliance guardians, prompting users to confirm or fix issues.
- **ChatOps for CI/CD.** The AI chat can initiate CI jobs or rollbacks. For instance, the PO asks the bot to run a "staging preview" build; the agent triggers the workflow and reports back with a URL.

## Traceability, Audit & Governance

Throughout every view, **traceability is central**. Every AI suggestion, code change, or configuration action is logged with context: which story, which agent or user, and which spec item it came from. For example, when the AI generates code, it automatically commits the rationale ("Generated by Composer Agent on [date], citing spec §3.2"). All logs and handoffs feed into the ADF evidence bundle.

Key elements to ensure compliance:

- **Spec linkage:** The workspace enforces that stories and commits include references to ADF spec clauses [3] . The UI might even auto-insert a spec ID when creating a branch or PR.
- **Audit Trail Pane:** A global panel lists every major event (lease start/stop, gate pass/fail, approvals) in chronological order. Inspecting an event shows linked artifacts (commits, documents, chat transcripts).
- **Governance Assistants:** A dedicated *"Governance Agent"* watches for policy violations (change to protected file, bypass of a gate) and issues alerts. It also assists with kill-switch scenarios by guiding the user through the CAPA runbook steps (as per ADF) if needed.

## References

This design synthesizes ADF and Speckit principles. Speckit promotes **spec-driven development** with auto-generated docs and traceability [2] . ADF v0.6.0 emphasizes linking every change to a spec clause and gating (e.g. the PO must accept the Story Preview before merge [8]). We borrow the *multi-agent* concept from frameworks like Symphony [5], where each agent has a specialized role in the lifecycle. Custom VS Code agents are leveraged to embody these roles [1]. Kanban-style boards have proven useful for agentic workflows [4] and are integrated here to maintain the sequential SSP semantics [9]. Throughout, **evidence bundles** record all outputs and decisions [6], ensuring the AI-driven process remains auditable and governed.

---

[1] Custom agents in VS Code
https://code.visualstudio.com/docs/copilot/customization/custom-agents

[2] README.md
https://github.com/adf-labs/speckit-internal/blob/e1d025e1168327e3ca886134edfd3ddd2fd77fb0/README.md

[3] [8] [9] adf-spec-v0.6.0.md
https://github.com/adf-labs/agentic-delivery-framework-internal/blob/410e450a33e492e64fd49206388153268cfeff16/docs/specs/adf-spec-v0.6.0.md

[4] GitHub - kaiban-ai/kaiban-board: A Kanban board-like UI to manage agentic workflows. Designed for seamless integration with KaibanJS to visualize and manage AI agent tasks in real time.
https://github.com/kaiban-ai/kaiban-board

[5] [7] agentic.state-of-art.md
https://github.com/K1ntus/MIDAS/blob/853ffa69f9ea0ae830971a650d613c8354148ca6/docs/agentic.state-of-art.md

[6] github.md
https://github.com/adf-labs/agentic-delivery-framework-internal/blob/410e450a33e492e64fd49206388153268cfeff16/docs/profiles/github.md