ChatGPT

Integrated ADF/SpecKit/VSCode Architecture

We propose a unified development environment built around VS Code devcontainers, Speckit (SpecKit), and the Agentic Delivery Framework (ADF) that spans the entire Scrum-based SDLC. Developers clone their repository and open it in a **VS Code Dev Container** that includes Node (for TypeScript) and Python runtimes out-of-the-box. This containerized setup (local or GitHub Codespace) ensures a consistent toolchain – e.g. a prebuilt TypeScript/Node image can be used as a base [1], and users can extend it with Python, Git, or other tools via the `devcontainer.json` (see the *Create a Dev Container* documentation [1] [2]). The devcontainer enables VS Code (or Codespaces) to run native language servers, linters, Git, and AI extensions seamlessly in an isolated environment. It also allows enterprises to customize the environment per project (choosing language versions, AI models, etc.) by editing the devcontainer spec [1] [2].

From this unified IDE, developers use **Speckit (SpecKit)** for specification-driven planning. Speckit is a Node/TypeScript CLI+TUI tool that guides teams through writing formal specifications and templates [3]. Its internal spec notes that Speckit includes a CLI, TUI, and optional AI agent stub in a pnpm workspace, with templates (blank, next-supabase, etc.) and linting runners built-in [3] [4]. In practice, during backlog grooming or sprint planning, teams use Speckit to create and iterate on spec documents (e.g. user stories, acceptance criteria) within the repo (`docs/specs/…` by default [5]). Speckit can open specs in the editor or show diffs, and it supports generating code scaffolds from templates. Critically, AI assistance is optional – by default Speckit does not call any AI provider unless explicitly enabled (i.e. an "AI propose" command would do nothing unless `ai.enabled=true` in config) [6] [7]. Thus, teams can adopt Speckit's spec-driven process without mandating AI usage, but have the option to invoke copilots or Claude for draft suggestions.

Within VS Code, the **AI Toolkit and Copilot extensions** accelerate development of the actual agent code. VS Code's *AI Toolkit Copilot Tools* provides Agent Code Generation, Model Guidance, Testing/Evaluation generators, and Tracing tools that integrate directly into the IDE [8] [9]. For example, the *Agent Code Gen* tool can generate boilerplate agent logic from a prompt, ensuring that the code follows best practices and even pre-selecting a default framework (e.g. Microsoft Agent SDK) if none is chosen [8]. Likewise, the *Tracing Code Gen* tool helps instrument the code for logging and debugging, with built-in support for both Python and TypeScript frameworks (it lists SDKs for both JS/TS and Python) [9]. In short, developers get Copilot-powered workflows right inside VS Code for writing, testing, and iterating on AI agents. (No matter the language, the AI Toolkit supports multiple stacks: it explicitly lists SDKs like `langchain`, `azure-ai-inference`, `openai`, etc. for both Python and JS/TS [9].)

Proposed Workflow

1. **Specification/Planning**: During sprint planning, the Product Owner or team opens the repo in VS Code. Using the Speckit TUI or commands, they create or edit spec documents under `docs/specs/` (or a custom path) [5]. These specs capture features or bug requirements in Markdown (with front-matter for version, owners, etc.) [10]. If enabled, an AI agent (via `spec propose`) can draft spec content, but by default it's purely human-driven, preserving Scrum's

transparency and collaboration. Speckit's templates (e.g. a project template or blank spec) ensure consistency in how requirements are documented [11].

2. **Development in VS Code**: Developers code in the same VS Code container. The environment includes TypeScript (Node 24+) and Python toolchains [6]. IDE extensions (GitLens, language servers, etc.) work as usual. Critically, AI CLI tools like GitHub Copilot CLI or Claude Code can also be installed in the container to run in the terminal or as extensions. For example, a developer might invoke Copilot CLI (`gh copilot`) to generate a feature branch, pull in context from code and Speckit docs, implement code, test it, and open a PR—all from the command line, as Copilot CLI can handle multi-step workflows. Within the editor, developers may use inline Copilot/chat or the Copilot Agent Mode to automate code tasks. The VS Code AI Toolkit (Agent Code Gen, etc.) further assists by generating code snippets, picking models, and adding tests or tracing based on prompts [8] [9].
3. **Local Testing and Commits**: As code and specs evolve, the devcontainer ensures tests can run locally (Node/Python tests) and Git commands are available. Code formatting, linting, and security checks (e.g. ESLint, pylint, SAST tools) all run in the container. Commits should follow a convention (e.g. semantic-release setup) and reference Speckit spec IDs or JIRA ticket numbers as needed. The devcontainer can include pre-commit hooks or linters by default.
4. **CI/CD Integration**: The centralized GitHub repository (or GitHub Enterprise) hosts the code. We leverage GitHub Actions for CI/CD, triggered on PRs or merges. Actions can lint code, run tests, and publish artifacts. Crucially, we can integrate Speckit into CI: e.g., an Action could run `speckit validate` to ensure specs are up-to-date, or even call an agent to review changes against the spec. If the Agentic Delivery Framework includes its own pipeline steps, those can be implemented as Actions or composite actions. Artifacts (binaries, containers) can be stored in the enterprise artifact repository or artifact registry.
5. **Backlog/Issue Tracking**: We continue using Jira for overall backlog and sprint management, but connect it with GitHub for traceability. By installing the **GitHub–Jira integration**, every branch, commit, and PR can link to a Jira issue [12]. For example, developers include the Jira ticket ID in commits and PRs; the GitHub for Jira app then displays GitHub activity in the Jira issue. This way, Scrum events (standups, sprint reviews) use Jira as the single source for tracking, but detailed work artifacts live in GitHub.
6. **Agentic Orchestration (Cloud)**: While the day-to-day workflow is local/VSCode-centric, the architecture should allow cloud agentic orchestration as needed. For example, a scheduled CI job or GitHub Action could spin up AI agents (e.g. via OpenAI or Microsoft AI SDKs) to perform release tasks or cross-repo analytics. The JFrog concept of an *agentic repository* suggests that the artifact repo itself could be "smart" – e.g. automatically tagging releases, notifying stakeholders, or even reverting broken builds [13]. In practice, we could use Azure AI Foundry or custom AI Services to run workflows (meeting, code review, release notes) triggered by GitHub events. This layer remains configurable: enterprises can opt in to specific automations (like automated code tests, or release-generation bots) while still keeping manual gates if desired.

Enterprise Adoption Considerations

This architecture reuses as much of the GitHub ecosystem as possible, as requested. We rely on GitHub (or GH Enterprise) for source control, Actions for CI/CD, and GitHub Apps for security and code scanning. The AI/agent pieces are similarly pluggable: Speckit's AI features can be toggled on/off in config, and our devcontainer can include multiple AI CLI tools (Copilot CLI, Claude CLI, etc.) so teams can choose their providers/models. The devcontainer spec itself is versioned in the repo and can be customized (we provide a base Node+TS setup and let teams extend it in their own `.devcontainer/devcontainer.json`). The workflow is **language-agnostic** at its core: while our examples use TS and Python (which we support out-of-the-box), nothing prevents adding other runtimes via the same devcontainer (the VS Code Tools support many languages [9] [1]).

Throughout the cycle, Scrum's roles and ceremonies remain: the Product Owner writes/specifies requirements (captured in Speckit docs and Jira), the team codes in VS Code (with AI-assisted help but still reviewed by humans), and the Scrum Master/DevOps engineer sets up the automated pipelines. The *guarantee* is that ADF's principles (scrum-first, agent-safe, as indicated by the tagline) hold: agents and AI tools assist but do not override Scrum's transparency and human decision-making. For instance, Speckit ensures spec changes are visible to the team [10], and any agentic actions (e.g. an AI closing a PR) would require pull request approval per policy.

In summary, this integrated architecture uses **VS Code DevContainers** for a consistent dev environment [1] [2], **Speckit** for formal spec-driven planning [3] [14], and **VS Code's AI Toolkit/Copilot** for smart coding assistance [8] [9]. It ties into enterprise tools (GitHub/GHE, Jira, artifact repos) for traceability and CI. By providing extensible devcontainer configs and making AI features optional yet available, it lets enterprises adopt the Scrum-oriented agentic delivery process with minimal friction, while enabling advanced workflows (like Copilot CLI automation or cloud-based AI pipelines) when desired.

Sources: This architecture is informed by the SpecKit design (Speckit) [3] [14], VS Code Dev Container documentation [1] [2], and VS Code AI Toolkit guidelines [8] [9]. The agentic repository paradigm from JFrog [13] motivates embedding intelligence into our CI/CD (e.g. smart artifact management).

---

[1] [2] Create a Dev Container
https://code.visualstudio.com/docs/devcontainers/create-dev-container

[3] [4] [5] [6] [7] [10] [11] [14] speckit-spec.md
https://github.com/adf-labs/speckit-internal/blob/e1d025e1168327e3ca886134edfd3ddd2fd77fb0/docs/internal/specs/speckit-spec.md

[8] [9] Use AI Toolkit Copilot tools for AI agent development
https://code.visualstudio.com/docs/intelligentapps/copilot-tools

[12] Integrate Jira with GitHub | Atlassian Support
https://support.atlassian.com/jira-cloud-administration/docs/integrate-jira-software-with-github/

[13] Agentic Repository: A New Paradigm for Software Delivery
https://jfrog.com/learn/devops/agentic-repository/