Spec Driven Development from Idea to Production

How structured context turns coding agents into reliable delivery systems

Dmitry Valetin
Founder of Spexus.ai
dmitry.valetin@gmail.com
https://www.linkedin.com/in/dvaletin/

Agenda

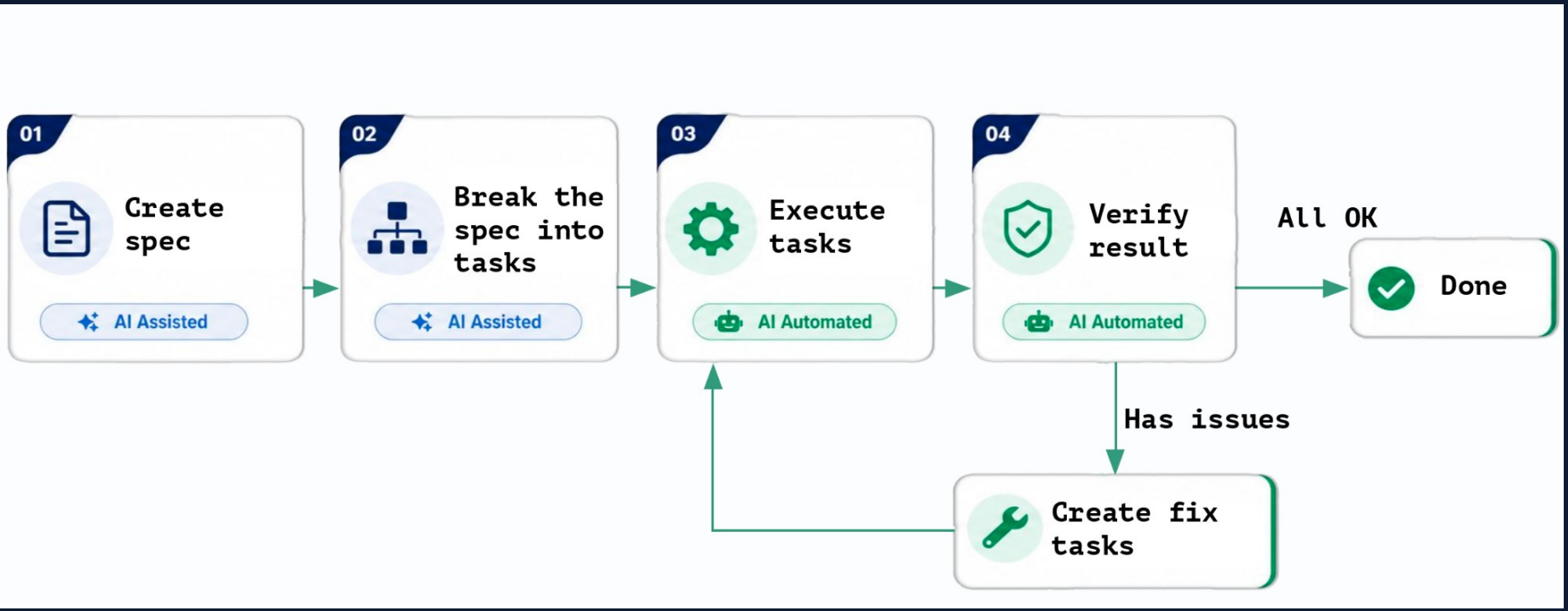
01 SDD workflow inside the SDLC

02 Context management for LLMs

03 SDD as a context-transfer system for coding agents

04 Real example: MCP JSON-RPC API Handler

SDD Workflow in the SDLC


01
Create spec
AI Assisted
02
Break the spec into tasks
AI Assisted
03
Execute tasks
AI Automated
04
Verify result
AI Automated
All OK
Done
Has issues
Create fix tasks

Context Management

Why context matters for LLMs

No built-in memory
LLMs do not know your product, users, architecture, or delivery constraints by default.

Unsafe confidence
Missing context produces plausible code, wrong abstractions, and weak verification.

Useful context
Intent, constraints, interfaces, and definition of done must be explicit.

Operational payoff
Better context reduces token waste, shortens review loops, and improves results.

SDD as Context Transfer

Product context for a coding agent

Agents do not start with product memory.

SDD transfers intent through a structured chain instead of hoping context appears inside a prompt.

- Epic = intent of the increment and expected business change
- User Story = a concrete slice of user value
- Requirement = a guardrail and implementation constraint
- Acceptance Criterion = a verifiable success point

Technology context and system design

Standards

Reusable rules shared across epics or scoped to a single epic.

System design

Architecture, boundaries, data flow, and implementation hints.

Delivery contract

Where code should live, which patterns to use, and how to run tests.

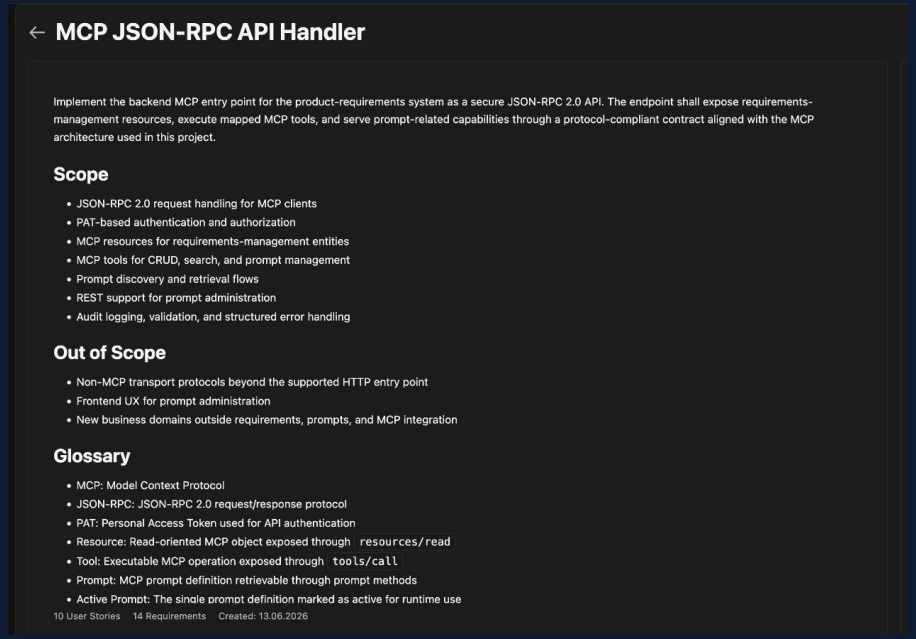
Together with product context, this makes agent output explainable, reviewable, and easier to verify.

CASE: Create MCP Server API

Epic as intent + technology context

Epic answers:

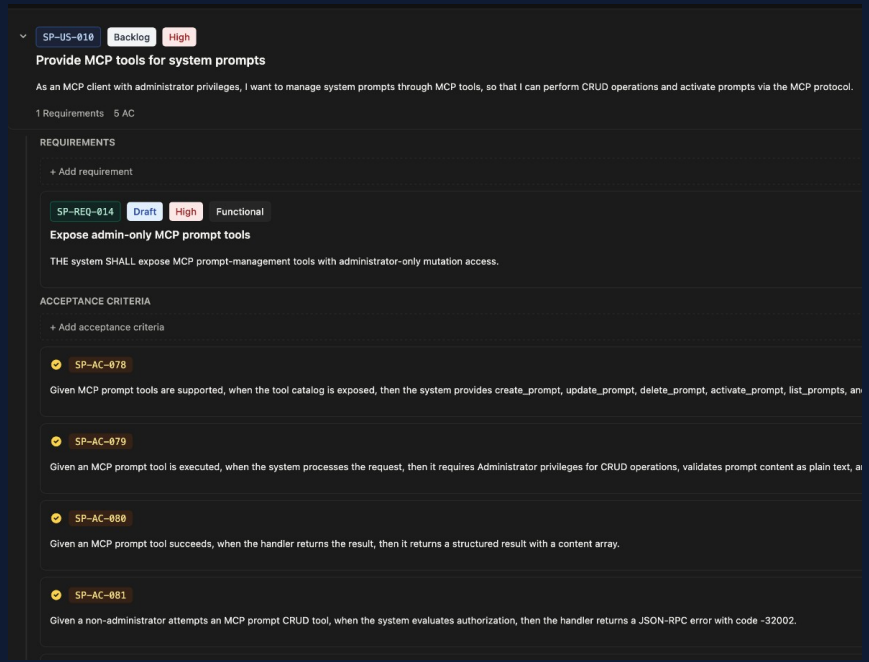
- **why this increment exists**
- **what system change is in scope**
- **which terminology and technical direction the team shares**



Product context: stories, requirements, acceptance criteria

Traceability layer:

- user stories define the user-facing slice
- requirements constrain implementation choices
- acceptance criteria define how success is verified



Task slicing, implementation, verification

Task slicing

Split the epic into traceable tasks. Each task becomes an agent prompt.

Coding

Run the coding orchestrator to implement tasks one by one.

Verification

Run a verification **sub-agent(*)** to review the result against the spec.

Fix gaps

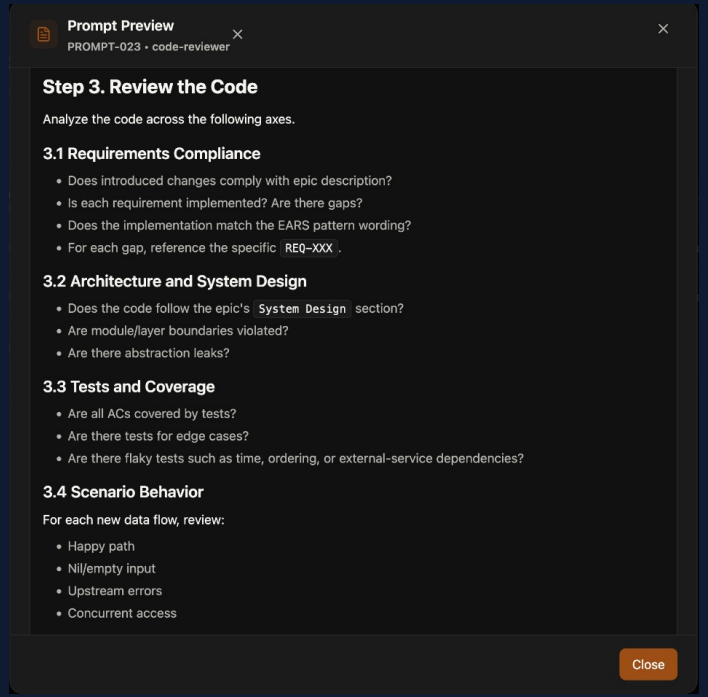
Each gap becomes a fix task. Rerun orchestration until the gaps are closed.

** A sub-agent does not share context with the orchestrator, which reduces the risk of hallucinations and context poisoning.*

Verification as a first-class contract

The definition of done becomes explicit:

- **each AC should map to a test or verification step**
- **review asks whether the implementation satisfies the spec**
- **evidence stays grounded in spec, not intuition**



What’s next?

Build

Build the product with CI/CD

Test

Run E2E, integration, and unit tests
Run the linter

Deploy

Deploy to production with CI/CD

This closes the loop from specification to production release.

Grazie mille. Ci sono domande?

Dmitry Valetin

Founder of Spexus.ai

dmitry.valetin@gmail.com

https://www.linkedin.com/in/dvaletin/