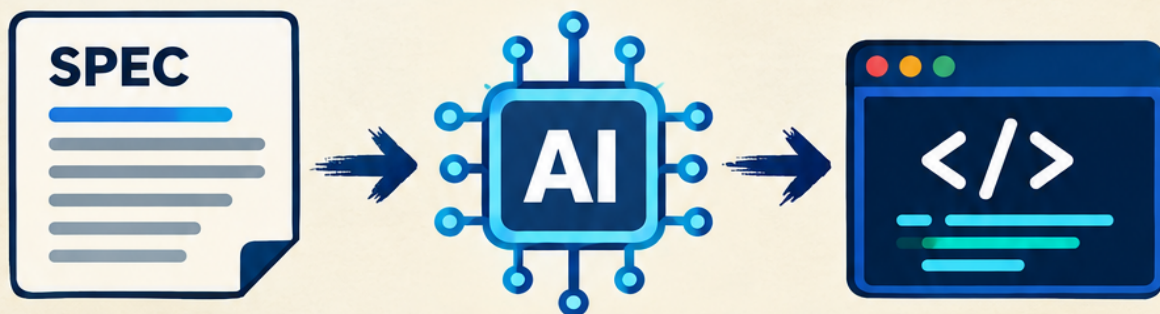




Spec-Driven Development

**From Requirements to Working Software
with AI**

**A Complete Practical Guide to Building
Real-World Software Through Specification-
First Engineering with AI Assistants
and Autonomous Agents**



JULIAN WEISS

Spec-Driven Development: From Requirements to Working Software with AI

A Complete Practical Guide to Building Real-World Software Through Specification-First Engineering with AI Assistants and Autonomous Agents

Steve Publications

This book is available at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>

This version was published on 2026-09-26



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Steve Publications

Contents

A Complete Practical Guide to Building Real-World Software Through Specification-First Engineering with AI Assistants and Autonomous Agents	1
Introduction	2
Chapter 1: The New Engineering Reality	5
The AI Coding Revolution	5
Why Prompting Alone Fails at Scale	6
The Hidden Cost of Emergent Behavior in AI-Generated Code	7
What Changes and What Stays the Same	8
How This Book Is Structured	10
Chapter 2: What Is Spec-Driven Development	12
The Spec-First Principle	12
Core Tenets of Spec-Driven Development	13
How Specifications Bridge Intent and Implementation	14
Spec-Driven Development Versus Traditional Requirements Engineering	15
The Role of AI Agents in the Spec-Driven Workflow	16
When to Use and When to Avoid This Approach	17
Chapter 3: Foundations: Requirements Engineering for the AI Era	20
A Brief History of Requirements Engineering	20
Functional Versus Nonfunctional Requirements	20
Stakeholder Analysis and Requirement Elicitation	20
Requirements Quality Criteria	20
How AI Changes the Requirements Engineering Game	20
Chapter 4: Structuring Requirements	22
User Stories, Use Cases, and User Journeys	22

CONTENTS

- Feature Decomposition and Prioritization 22
- Writing Clear, Unambiguous Requirement Statements 22
- Capturing Constraints and Edge Cases Explicitly 22
- Requirement Templates and Patterns 22
- Organizing Requirements for Traceability 23
- Chapter 5: Domain Modeling for AI-Assisted Development 24**
 - What Is a Domain Model and Why It Matters 24
 - Domain-Driven Design Concepts Adapted for AI Development 24
 - Entities, Value Objects, Relationships, and Invariants 24
 - Event Storming and Bounded Contexts 24
 - Writing Domain Models AI Agents Can Consume 24
 - Maintaining Domain Models Across Project Evolution 25
- Chapter 6: Functional Specifications: From Requirements to Blueprints 26**
 - The Anatomy of a Functional Specification 26
 - Behavioral Specifications and State Transitions 26
 - Writing Acceptance Criteria That Are Truly Testable 26
 - Scenario-Based Specification with Given-When-Then 26
 - Specifying APIs, Data Flows, and Side Effects 26
 - Examples of Complete Functional Specifications 27
- Chapter 7: Nonfunctional Specifications: Quality, Constraints, and Trade-offs 28**
 - Performance Specifications and Benchmarks 28
 - Security and Privacy Requirements as Specs 28
 - Reliability, Availability, and Fault Tolerance 28
 - Scalability and Capacity Planning in Specs 28
 - Usability, Accessibility, and Observability Requirements 28
 - Encoding Trade-offs and Design Decisions 29
- Chapter 8: Specification Formats and Languages 30**
 - Markdown-Based Specifications 30
 - Structured Formats: YAML, JSON, TOML 30
 - Formal and Semi-Formal Specification Languages 30
 - Behavior-Driven Development and Cucumber/Gherkin 30
 - OpenAPI, AsyncAPI, and Contract-First Specification 30
 - Choosing the Right Format for Your Context 31
- Chapter 9: System Architecture as Specification 32**

CONTENTS

Architecture as a First-Class Specification Artifact	32
High-Level Design Documents for AI Agents	32
Component Diagrams and Interaction Patterns	32
Data Architecture and Schema Specifications	32
Infrastructure and Deployment Specifications	32
Documenting Architectural Decisions (ADRs)	33
Chapter 10: API and Data Contracts	34
Why Contracts Matter for AI Development	34
RESTful API Specifications with OpenAPI	34
GraphQL Schemas and Code Generation	34
Database Schemas as Enforceable Contracts	34
Message Queues, Events, and Event-Driven Contracts	34
Versioning Strategies and Backward Compatibility	35
Chapter 11: The Core Project: A Real-World Application	36
Problem Statement and Project Scope	36
Technology Stack Selection and Justification	36
Requirements Document for the Core Project	36
Domain Model for the Task Management System	37
Initial Architecture and System Context	37
Repository Structure and Project Conventions	37
Chapter 12: Task Decomposition and Implementation Planning	38
From Specifications to Tasks	38
Work Breakdown Structures for AI Development	38
Task Size, Granularity, and Context Windows	38
Dependency Mapping and Execution Ordering	38
Writing Task Specifications AI Agents Can Follow	38
Project Tracking and Progress Monitoring	39
Chapter 13: Configuring AI Agents for Development	40
AI Coding Tools Landscape and Agent Types	40
System Prompts and Agent Instructions	40
Configuring Code Style, Conventions, and Constraints	40
Environment Setup, Tooling, and Sandboxing	40
Managing Context and Project Knowledge	40
Safety, Guardrails, and Access Controls	41
Chapter 14: Context Engineering and Prompt Design	42

CONTENTS

What Is Context Engineering	42
Including Specifications in AI Context Windows	42
Prompt Patterns for Specification-Driven Coding	42
Few-Shot Examples and Reference Implementations	42
Handling Large Codebases and Split Context	42
Iterative Refinement of Prompts and Instructions	43
Chapter 15: Generating Code from Specifications	44
The Generation Workflow Step by Step	44
Building the Project Foundation with AI	44
Implementing Domain Models and Business Logic	44
Generating API Endpoints and Controllers	44
Writing Database Migrations and Data Access	44
Reviewing and Correcting AI-Generated Code	45
Chapter 16: Testing as Verification Against Specifications	46
Test-Driven Development and AI	46
Generating Tests from Acceptance Criteria	46
Unit Tests, Integration Tests, and End-to-End Tests	46
Property-Based Testing and Fuzzing	46
Contract Testing for APIs	46
Coverage, Mutation Testing, and Quality Metrics	47
Chapter 17: Debugging, Refactoring, and Iteration	48
When AI Gets It Wrong: Common Failure Modes	48
Debugging AI-Generated Code	48
Using Specifications as Debugging Anchors	48
Refactoring with AI Safely	48
Handling Regressions and Specification Changes	48
Code Review Practices for AI-Assisted Development	49
Chapter 18: Integration, Deployment, and Observability	50
Continuous Integration for Spec-Driven Pipelines	50
Deployment Specifications and Infrastructure as Code	50
Environment Configuration and Secrets Management	50
Observability: Logging, Metrics, Tracing	50
Runtime Verification and Canary Analysis	50
The Core Project in Production	51
Chapter 19: Multi-Agent Orchestration and Parallel Development	52

When Single Agents Are Not Enough	52
Role-Based Agent Orchestration	52
Parallel Development Strategies	52
Conflict Resolution and Merge Management	52
Scaling Development with Agent Swarms	52
Case Study: Parallel Development of TaskFlow	53
Chapter 20: Specification Management and Evolution	54
Living Documents Versus Frozen Specs	54
Versioning Specifications	54
Change Management and Impact Analysis	54
Detecting and Preventing Specification Drift	54
Maintaining Traceability from Requirements to Code	54
Governance and Approval Workflows	55
Chapter 21: Advanced Topics and Large-Scale Practice	56
Modernizing Legacy Systems with Spec-Driven Approaches	56
Large Monorepos and Cross-Project Dependencies	56
Distributed Systems and Microservices Architecture	56
Security, Compliance, and Privacy at Scale	56
Managing Technical Debt with AI Assistance	56
Organizational Adoption and Team Practices	57
Chapter 22: Comparing Tools, Frameworks, and Platforms	58
GitHub Spec Kit: Workflow, Features, Limitations	58
Amazon Kiro: Workflow, Features, Limitations	58
Cursor, Windsurf, and IDE-Integrated AI	58
Claude Code, Devin, and Autonomous Agents	58
Open-Source Frameworks and Community Tools	58
How to Evaluate and Choose Tools for Your Team	59
Conclusion: The Future of Spec-Driven Development	60
Glossary	61
References	62

A Complete Practical Guide to Building Real-World Software Through Specification-First Engineering with AI Assistants and Autonomous Agents

This book is a comprehensive, practical guide to a disciplined methodology for developing software in the era of AI coding assistants and autonomous agents. It shows how explicit, structured specifications serve as the bridge between human intent and reliable AI-generated code, covering everything from requirements elicitation and domain modeling through architecture, implementation, testing, deployment, and long-term maintenance. The book develops a complete, working software project step by step to demonstrate the entire journey from an initial business problem to a tested, deployable application, while comparing current tools, frameworks, and platforms without relying on any single vendor. The intended audience includes software developers, senior engineers, technical leads, and software architects who want to adopt specification-driven practices that produce production-quality, maintainable software with AI assistance.

Introduction

A developer sits down with a new feature request. She opens her AI coding assistant, types something like “build a task management system with user authentication and team collaboration” and watches as code appears almost instantly. Within minutes she has a working prototype, a database schema, API endpoints, and frontend components. It feels like magic. By the end of the afternoon the prototype works well enough to show to a stakeholder.

Now imagine two weeks later. Someone asks to change how user permissions work. The developer opens the codebase and immediately realizes she does not actually understand how most of it works. She knows she prompted for it, but she never designed it. There are inconsistent patterns across different parts of the code. One module uses JWT tokens, another uses session cookies, and a third reinvents authorization logic from scratch. The database schema has redundant fields, missing foreign keys, and naming conventions that contradict each other. Fixing the permissions now means untangling three different authorization schemes that were never planned to work together.

This scenario has become commonplace. AI coding assistants have fundamentally altered what it means to build software. Code that once took days of deliberate design and careful implementation can now emerge in minutes from natural language prompts. For small projects this acceleration is genuinely transformative. For anything larger, it introduces a new class of problems that existing engineering practices were not designed to handle.

The central challenge is not that AI generates bad code. It is that without deliberate upfront specification, AI generates code that matches its best guess of what you want, not what you actually need. The model fills in ambiguity with training data, not with domain knowledge. It chooses patterns it has seen often, not patterns that fit your particular constraints. It optimizes for completeness of a single prompt, not for coherence across a whole system or for maintainability six months from now.

This book proposes a methodology to solve this problem. Spec-Driven Development is a specification-first engineering approach in which human requirements are transformed into explicit, reviewable, machine-actionable

specifications before any code is generated. These specifications become the authoritative source that guides AI coding assistants and autonomous agents throughout the entire development lifecycle. The specifications define not only what the system should do but also how it should be structured, what constraints it must respect, what quality attributes it must achieve, and how its behavior should be verified.

Spec-Driven Development is not anti-AI. It is the opposite. It is an approach designed specifically for the era in which AI is a primary production tool. The methodology recognizes that AI coding agents are most effective when given unambiguous, structured, verifiable input. Vague prompts produce vague results. Precise specifications produce precise results. The discipline of writing specifications forces clarity of thought before complexity is introduced into the codebase.

This book will show you how to do this systematically. You will learn how to elicit and structure requirements that capture real business needs. You will learn how to create domain models that establish a shared understanding of the problem space. You will learn how to write functional and nonfunctional specifications that are both human-readable and machine-actionable. You will learn how to design system architecture as specification, how to define API and data contracts that multiple agents can implement against, and how to break complex systems into AI-actionable tasks with clear acceptance criteria. You will learn how to configure and direct AI agents, how to engineer prompts that encode your specifications, how to generate code systematically from specifications, how to build automated tests that verify behavior against stated requirements, and how to maintain the connection between intent and implementation as the system evolves.

To make this concrete, the book develops a complete, production-grade software project step by step. This is not a toy example with hand-wavy explanations. It is a real task management system with user authentication, team collaboration features, file attachments, notifications, and role-based access control. You will see the full requirements document, the domain model, the architecture design, the API specification, the database schema, the test strategy, and the actual implementation. Every piece of code is internally consistent and executable. You will be able to reproduce the entire project from the beginning, run it locally, and verify that it behaves according to the specifications.

The book also takes an honest look at the current tooling landscape. You will

find detailed comparisons of GitHub Spec Kit, Amazon Kiro, Cursor, Windsurf, Claude Code, and other emerging tools. Each is examined in terms of its workflow, underlying concepts, practical capabilities, limitations, and trade-offs. The methodology described here is independent of any single tool, but understanding how different tools implement or support spec-driven practices will help you choose the right environment for your team.

Spec-Driven Development is not without its costs. Writing specifications takes time. Reviewing specifications takes time. There is a temptation to skip straight to prompting and getting code. For very small projects this temptation is rational. But as projects grow beyond a few hundred lines of code, as teams grow beyond a single developer, as systems grow beyond a single deployment, the upfront investment in specifications pays dividends in maintainability, reliability, and predictability. This book will help you understand when the investment is worthwhile and when it is overkill.

The journey ahead moves from foundation to mastery. We begin by understanding the problem that Spec-Driven Development addresses, then establish the theoretical and practical foundations, then develop complete projects that demonstrate every phase of the process, then explore advanced topics for large-scale practice, and finally examine the tools and future directions. By the end you should have both the conceptual understanding and the practical skills to apply Spec-Driven Development in your own projects.

The era of AI-assisted development is here. The question is not whether to use these tools but how to use them well. Spec-Driven Development is one answer to that question. Let us begin.

Chapter 1: The New Engineering Reality

The software development landscape has changed more in the past three years than in the previous two decades. AI coding assistants have moved from novelty to necessity, and autonomous coding agents are beginning to handle substantial engineering tasks with minimal human direction. These changes are not merely incremental improvements in productivity. They are structural shifts in how software is created, maintained, and evolved. Understanding these changes is essential for any engineering team that wants to build reliable software in the AI-augmented era.

The AI Coding Revolution

The story of AI in software development began with autocomplete. Early tools suggested individual tokens as you typed, trained on vast corpora of public code. They were useful, sometimes delightfully so, but fundamentally reactive. You drove, the model suggested.

The next phase introduced context awareness. Tools like GitHub Copilot, launched in 2021, could read surrounding code, comments, and even open files to generate more relevant completions. They could suggest whole functions, whole test suites, whole configuration files. The quality improved dramatically, but the fundamental dynamic remained the same. You were still the primary author, and the AI was still a suggestion engine.

Then came agentic capabilities. Starting around 2024, AI coding tools began to act autonomously. They could read multiple files, plan multi-step tasks, make edits across a codebase, run commands in a terminal, and debug failures by iterating on their own output. Cognition AI introduced Devin in March 2024 as the first commercially marketed autonomous software engineer, claiming it could reason, plan, and complete complex engineering tasks independently [1]. Anthropic released Claude Code, an agentic coding tool that runs in the terminal, reads your codebase, edits files, and executes development tasks

through natural language instructions [2]. OpenAI released Codex CLI with similar agentic capabilities. IDE-integrated tools like Cursor and Windsurf added multi-file agents, project-level reasoning, and increasingly autonomous workflows.

The results have been striking. On benchmarks like SWE-bench Verified, which tests AI systems on real GitHub issues from popular open-source Python repositories, top models have progressed from single-digit resolution rates to over 90 percent in the space of two years [3]. This does not mean software engineering is solved. It means that for a significant subset of well-defined engineering tasks, an AI agent can now produce correct, tested, production-ready code with no more than a brief human prompt and occasional intervention.

The implications for individual developers are clear. Anyone with access to modern AI tools can generate working code for a wide variety of tasks at speeds that were previously impossible. The barrier to creating software has dropped dramatically. A solo developer can now prototype systems that once required a team. A non-technical founder can generate functional prototypes from natural language descriptions.

But the implications for professional engineering teams are more subtle. When code generation becomes cheap and easy, the primary cost of development shifts. It is no longer typing syntax. It is defining intent precisely enough that the generated code is correct, coherent, and maintainable. It is reviewing output for correctness, security, and alignment with architectural standards. It is designing systems in a way that remains understandable and extensible when most of the code was written by something other than a human with deep domain knowledge.

Why Prompting Alone Fails at Scale

The most common mistake teams make when adopting AI coding tools is assuming that good prompting is sufficient. The logic is simple: if vague prompts produce decent results for small tasks, then better prompts should produce great results for large systems.

This logic fails for several reasons.

First, context windows have limits. Even the largest models can only hold so much information in their active context at once. A complex system with

dozens of modules, hundreds of data types, and many interacting services cannot be fully described in a single prompt. Developers compensate by feeding context incrementally, but this creates fragmentation. The agent sees one piece at a time, not the whole picture. Decisions made early based on incomplete information may conflict with requirements discovered later.

Second, natural language is inherently ambiguous. When a developer prompts “add user authentication,” the model interprets this based on patterns in its training data, not based on the specific security requirements, regulatory constraints, or organizational standards of the project. It might choose JWT when the system requires OAuth, or implement permissive password policies, or miss requirements like multi-factor authentication, session management, or audit logging. The ambiguity is not a bug of natural language prompting. It is a fundamental property of using underspecified input for complex tasks.

Third, AI models are generative, not deterministic. Running the same prompt twice may produce different results. This non-determinism is not necessarily harmful for creative tasks, but it is problematic for engineering. If a team member prompts the agent to “implement the notification service” and gets one design, and another member prompts the same thing and gets a different design, the system ends up with inconsistent patterns that are costly to reconcile.

Fourth, AI models are optimized for generating plausible-looking code, not for generating correct code against specific requirements. A model might generate code that compiles, passes basic tests, and appears to implement the requested feature, while silently violating security constraints, business rules, or nonfunctional requirements. This is sometimes called “hallucinated correctness.” The code looks right but is wrong in ways that are not immediately apparent.

These failures compound at scale. For a small utility function, a vague prompt might produce acceptable results. For a system with authentication, authorization, data persistence, external integrations, and complex business rules, vague prompts produce fragmented, inconsistent, and often insecure code. This phenomenon has been labeled “vibe coding” by some in the industry, referring to the practice of generating code based on a general feeling about what is needed rather than precise specifications [4]. Vibe coding works well enough for throwaway prototypes and personal projects. It fails catastrophically for systems that need to be maintained, extended, and trusted.

The Hidden Cost of Emergent Behavior in AI-Generated Code

When AI generates code from prompts, the resulting architecture is emergent rather than designed. It emerges from the model's training data, the prompt's wording, and random variation. This is fine for exploratory work, but it creates serious problems for production systems.

Consider a team building a backend service with AI assistance. Early tasks might involve creating API endpoints, database models, and authentication logic. Each task is prompted separately, with context provided as needed. The model generates code that satisfies each individual prompt. Over time the codebase grows. Eventually someone notices problems: inconsistent error handling across endpoints, mixed use of promises and `async/await`, multiple different date formatting approaches, duplicated validation logic, and database queries scattered through controller code instead of in a dedicated data access layer.

These problems are not because the AI generated bad code. They are because the AI generated locally optimal code for each individual prompt without a global architectural vision. There was no specification saying “all errors must be handled consistently,” or “the data access layer must be separated from business logic,” or “all dates must be stored in UTC.” These constraints exist in the minds of experienced developers but not in the prompts.

The cost of this emergent behavior is paid later, when refactoring becomes necessary. Refactoring AI-generated code is difficult because the developer who prompted for it may not fully understand it, and the code may be structured in non-standard ways that are hard to read. The lack of intentional design means there are no clear boundaries or invariants to preserve during refactoring. Changes made in one area may break hidden assumptions in another.

This pattern is so common that it has a name in the AI development community: “vibe code debt.” It is technical debt specifically caused by the absence of deliberate design before AI-assisted implementation. Like traditional technical debt, it accumulates slowly and then becomes expensive to repay. Unlike traditional technical debt, it can accumulate much faster because AI can generate large amounts of code in a short time.

What Changes and What Stays the Same

It is important to be precise about what has changed and what has not.

What has changed:

The cost of generating code has dropped dramatically. Tasks that once took hours or days can now take minutes. This means that projects with limited budgets can be larger and more ambitious. It also means that bad code can be generated faster than ever before.

The skillset of an effective developer has shifted. Proficiency in syntax is still useful but no longer central. The skills that matter most are the ability to define requirements precisely, to design coherent architectures, to recognize when generated code is incorrect or incomplete, and to guide AI agents toward correct solutions.

The nature of code review has evolved. Reviewers must now evaluate code they did not write in the traditional sense, which requires different skills. A reviewer needs to verify that code satisfies requirements, not just that it follows style guidelines and avoids obvious bugs. This requires clear, testable requirements to review against.

What has not changed:

Software engineering remains the disciplined practice of building systems that satisfy requirements reliably, efficiently, and maintainably. The fundamental problems are the same: managing complexity, ensuring correctness, maintaining consistency across large codebases, and balancing competing quality attributes.

Requirements engineering is still hard. AI does not make it easier to discover what stakeholders actually need. AI does not resolve ambiguity in stakeholder requirements. If requirements are vague, inconsistent, or incomplete, AI will produce code that is vague, inconsistent, or incomplete.

Architecture still matters. A system with poor architecture will be difficult to maintain regardless of whether the code was written by humans or AI. In fact, poor architecture is more dangerous with AI because AI will faithfully generate code that follows a bad architecture, multiplying the damage.

Testing is still essential. AI-generated code can be wrong in subtle ways. Automated tests remain the primary defense against regressions and silent failures.

Security is still critical. AI models are trained on public code, much of which contains known vulnerabilities. AI-generated code often includes insecure patterns if security is not explicitly specified as a constraint.

The human element is still central. AI is a tool, not a replacement for engineering judgment. The most effective teams use AI to amplify human expertise, not to substitute for it.

How This Book Is Structured

This book is organized to take you from understanding the problem to mastering the solution.

The first section establishes the foundation. You will learn what Spec-Driven Development is, why it works, and how it relates to established practices in requirements engineering, domain-driven design, and test-driven development. These chapters draw on decades of software engineering knowledge and adapt it for the AI-augmented context.

The second section covers the core methodology in detail. You will learn how to elicit and structure requirements, how to create domain models, how to write functional and nonfunctional specifications, how to choose appropriate specification formats, how to design system architecture as specification, and how to define API and data contracts. These chapters are practical and concrete, with examples that you can apply directly to your own projects.

The third section develops a complete project from requirements to deployment. This is the practical core of the book, where theory meets implementation. You will see a real task management system specified, designed, implemented with AI assistance, tested, and deployed. Every artifact is real: the requirements document, the domain model, the architecture design, the API specification, the database schema, the test suite, and the actual source code. You will see how specifications guide each phase, how AI agents are configured and directed, how tasks are decomposed, how code is generated and reviewed, and how the final system is verified against its specifications.

The fourth section covers advanced topics for production-scale practice. You will learn about multi-agent orchestration, parallel development, large codebases and monorepos, legacy system modernization, distributed systems, security and compliance, technical debt management, and organizational

adoption. These chapters address the complexities that real-world projects introduce.

The final section examines the tooling landscape. You will find detailed, objective comparisons of current tools and platforms that support spec-driven and AI-assisted development, including their workflows, capabilities, limitations, and trade-offs. This section will help you choose tools that fit your team's needs.

Throughout the book, code examples are real and executable, not illustrative pseudocode. Specifications are real and complete, not abbreviated for brevity. Trade-offs are discussed honestly, not swept under the rug. The goal is not to present Spec-Driven Development as a silver bullet, but as a practical methodology with real costs and real benefits that can be adopted incrementally and adapted to your context.

The next chapter defines Spec-Driven Development precisely, establishes its core principles, and explains how it differs from both traditional requirements engineering and from casual AI prompting. This definition is important because the term is used in many different ways, and clarity about what we mean is essential before we can explore how to do it.

Chapter 2: What Is Spec-Driven Development

The term “Spec-Driven Development” has appeared in many contexts over the past decade and a half, and it has meant different things to different people. In the 2000s, researchers used it to describe combining test-driven development with design by contract, treating tests and contracts as complementary specifications [5]. In the 2010s, it sometimes referred to model-driven approaches where specifications in formal languages were transformed into code. Today, in the context of AI-assisted development, it refers to a practice that is related to but distinct from both of these.

This chapter defines Spec-Driven Development as used throughout this book, establishes its core principles, explains the mechanics of how specifications bridge human intent and AI-generated implementation, and clarifies when this approach is appropriate and when it is not.

The Spec-First Principle

At its simplest, Spec-Driven Development is the practice of writing detailed, structured specifications before generating code, and treating those specifications as the authoritative source of truth that guides and constrains all downstream work.

This principle is simple to state but radical in its implications for AI-augmented development. Most developers using AI coding tools today work in the opposite direction. They have an idea, they start prompting, and code emerges. Documentation, if it exists at all, is written after the fact, often incompletely, and frequently becomes outdated as the code evolves. The code is the source of truth, and the documentation attempts to describe it.

Spec-Driven Development inverts this relationship. The specification is the source of truth, and the code must conform to it. Before any code is generated, the team invests time in writing specifications that capture what the system should do, how it should behave, what constraints it must respect, and how its

behavior will be verified. These specifications are reviewed by humans, revised until they are clear and complete, and then used as the primary input for AI agents that generate code.

This inversion is motivated by a specific observation about AI coding tools: they are remarkably good at turning well-specified requirements into correct code, and remarkably unreliable at discovering or filling in unspecified requirements. If a specification says “the system shall validate all user input against a allowlist of permitted characters,” the AI will implement that validation faithfully. If the specification is silent on input validation, the AI will likely not include it, or will include incomplete validation based on patterns it has seen in its training data.

The spec-first principle therefore does two things. First, it forces clarity of thought before complexity is introduced into the codebase. Writing a specification requires you to think through edge cases, error conditions, data flows, and interactions that might not be obvious from a high-level idea. Second, it gives AI agents a concrete, reviewable artifact to work from, reducing ambiguity and the chance of hallucinated or incorrect implementation.

Core Tenets of Spec-Driven Development

Spec-Driven Development, as defined in this book, rests on seven core tenets:

First, specifications precede implementation. No code is generated before the specification for that code is written, reviewed, and agreed upon. This does not mean that every line of code must have its own specification written in advance. It means that before work on a feature, module, or system begins, there must be a specification that describes what the work will produce.

Second, specifications are explicit and unambiguous. They avoid vague language like “should be user-friendly,” “must handle errors gracefully,” or “will be fast.” Instead they use precise, measurable, testable statements: “the system shall display results within 200 milliseconds for queries returning fewer than 1000 rows,” or “the system shall validate all user-supplied input and reject any input containing characters outside the specified allowlist.”

Third, specifications are the authoritative source of truth. When there is a discrepancy between the code and the specification, the specification is assumed correct, and the code is changed to match. This does not mean specifications never change. When a specification change is needed, the

process is: update the specification, review the change, then update the code. Code changes are not made independently of specification changes.

Fourth, specifications are machine-actionable as well as human-readable. They are structured in formats that AI agents can parse, reason about, and use to guide implementation. This does not mean they must be in formal mathematical notation. It means they should use consistent structure, clear language patterns, and formats that tools can work with.

Fifth, specifications include verification criteria. Every specification statement includes or implies a way to verify that it has been satisfied. This verification is typically through automated tests that encode the acceptance criteria as executable checks. If a requirement cannot be verified, it is not a valid specification statement.

Sixth, specifications are reviewable artifacts. They are version-controlled, subject to the same review processes as code, and treated as first-class engineering artifacts, not as disposable drafts. Reviews focus on clarity, completeness, correctness, consistency, and testability.

Seventh, specifications are maintained throughout the system lifecycle. They are not frozen documents that become obsolete after initial development. When requirements change, specifications are updated, and the impact of those changes is analyzed before code is modified. This ensures ongoing alignment between stated intent and implemented behavior.

How Specifications Bridge Intent and Implementation

The fundamental problem that Spec-Driven Development addresses is the gap between human intent and machine implementation. Humans think in terms of goals, outcomes, business rules, and user needs. Code is written in terms of data structures, algorithms, control flow, and API calls. Translating between these two levels of abstraction requires deliberate design work.

Without specifications, this translation is implicit and scattered. A developer has a mental model of what the system should do, and implements pieces of it as needed, guided by intuition, experience, and local context. For a single developer working on a small system this works fine. For a team, or for a complex system, or for AI agents working on different parts of the system, it breaks down.

Specifications make this translation explicit and centralized. They start at the human level, capturing business goals, user needs, and stakeholder requirements. They progressively refine these goals into more concrete descriptions of system behavior. They end at a level of detail that is sufficient for an AI agent to generate correct implementation without guessing.

Consider a requirement like “users should be able to create and manage teams.” Without specification, this is too vague. What is a team? Who can create a team? How many members can a team have? What permissions do members have? What happens when a member is removed? What happens when a team is deleted?

A specification addresses these questions explicitly. It defines a team as a named collection of users with an owner and zero or more members. It states that any authenticated user can create a team with a unique name. It specifies the maximum team size as 100 members. It defines permissions: owners can manage membership and settings, members can view team resources, and non-members cannot access team resources. It specifies that removing the last member from a team does not delete the team, but deleting a team requires owner authorization and removes all team resources or reassigns them according to a configurable policy.

With this level of detail, an AI agent can implement the feature correctly without making assumptions. Without it, the agent will guess, and different agents may guess differently, leading to inconsistent behavior.

The specification becomes a shared artifact that humans can review and agree upon before implementation begins. It captures the intent in a form that survives beyond the initial conversation, that can be referenced during implementation, testing, and maintenance, and that new team members can read to understand how the system is supposed to work.

Spec-Driven Development Versus Traditional Requirements Engineering

Spec-Driven Development draws heavily on established practices from requirements engineering, but it adapts them for the AI-augmented context in several important ways.

Traditional requirements engineering, as defined in standards like ISO/IEC/IEEE 29148, is a discipline focused on discovering, analyzing,

documenting, and validating the needs and constraints of stakeholders [6]. The process typically involves elicitation techniques like interviews, workshops, and surveys; analysis techniques like modeling and prototyping; and documentation in the form of a Software Requirements Specification or similar artifact. The emphasis is on completeness, correctness, consistency, and traceability.

Spec-Driven Development shares these goals but differs in practice. First, it recognizes that in AI-augmented development, requirements are not just for human consumption. They are also the primary input for AI agents, so they must be written in ways that models can parse and reason about effectively. This favors structured formats, consistent language patterns, and explicit acceptance criteria over free-form narrative.

Second, it tightens the connection between requirements and verification. In traditional processes, requirements are often validated through reviews, walkthroughs, and manual testing. In Spec-Driven Development, requirements are validated primarily through automated tests that are generated from or alongside the specification. A requirement that cannot be expressed as a test is considered suspect and is either refined or questioned.

Third, it accepts more frequent change. Traditional requirements engineering processes can be slow and bureaucratic, designed for environments where change is costly and should be minimized. Spec-Driven Development is designed for agile, iterative development where change is expected and should be managed efficiently. Specifications are living documents that evolve with the system, and the process for changing them is lightweight but disciplined.

Fourth, it leverages AI not just for implementation but also for specification authoring and review. AI agents can help generate initial specifications from high-level requirements, suggest clarifications and edge cases, identify inconsistencies, and propose test scenarios. Human engineers review, refine, and approve the specifications, using AI as a thinking partner rather than as a replacement for judgment.

Spec-Driven Development is therefore not a rejection of traditional requirements engineering. It is an adaptation of its principles and practices for a context in which code generation is cheap, change is frequent, and specifications must serve both human and machine consumers.

The Role of AI Agents in the Spec-Driven Workflow

In Spec-Driven Development, AI agents are not the primary authors of specifications, nor are they the sole implementers of code. They are tools that operate within a disciplined process defined by humans.

The spec-driven workflow typically follows these phases:

In the requirements phase, humans define high-level goals, constraints, and priorities. AI agents assist by helping to structure these goals, suggesting missing considerations, and generating draft specifications that humans review and refine.

In the specification phase, humans and AI collaborate to produce detailed, unambiguous specifications. Humans provide domain knowledge, clarify intent, and resolve ambiguities. AI agents help by generating structured content, suggesting edge cases, identifying inconsistencies, and ensuring that specifications are written in clear, testable language.

In the planning phase, AI agents decompose specifications into concrete implementation tasks, identify dependencies, estimate effort, and propose an execution order. Humans review the plan, adjust priorities, and approve the work breakdown.

In the implementation phase, AI agents generate code guided by the specifications, the plan, and the existing codebase. They follow the task descriptions, use the specified patterns and conventions, and generate tests alongside implementation. Humans review the generated code, run tests, verify behavior against specifications, and approve or reject changes.

In the verification phase, automated tests run continuously to verify that implementation satisfies specifications. When tests fail, humans or AI agents investigate and fix the root cause, either in the code or in the specification if the specification was incorrect.

In the maintenance phase, when requirements change, humans update the specifications, AI agents analyze the impact on existing code, and changes are implemented and verified systematically.

Throughout this workflow, AI is powerful but constrained. It generates options, drafts, and implementations, but humans make decisions about requirements, architecture, priorities, and quality. The specification is the primary mechanism by which human decisions are communicated to AI agents.

When to Use and When to Avoid This Approach

Spec-Driven Development is valuable but not universally appropriate. Understanding when to apply it and when to skip it is an important part of using it effectively.

Use Spec-Driven Development when:

- The system is non-trivial in scope. Projects with more than a few hundred lines of code, or that will be maintained beyond a single sprint, benefit from the clarity and structure that specifications provide.
- Multiple people or agents will work on the system. Specifications establish a shared understanding that reduces miscommunication and inconsistent implementation.
- The system has important constraints around security, compliance, reliability, or correctness. Explicit specifications make these constraints visible and enforceable rather than implicit and easy to overlook.
- The system will be maintained and extended over time. Specifications serve as documentation that remains accurate because they are maintained as part of the development process.
- You are using AI agents for implementation. AI agents work best with explicit, unambiguous input. Specifications provide exactly that.
- You need traceability between requirements and implementation. Specifications provide the basis for linking requirements to code, tests, and deployments.

Avoid or minimize Spec-Driven Development when:

- The task is trivial. Adding a utility function, fixing a typo, or adjusting a CSS property does not warrant a formal specification. Use direct prompting for small, self-contained changes.
- You are exploring or prototyping. When the goal is to discover what is possible or to validate an idea, specifications are premature. Explore freely, then write specifications once the direction is clear.
- Requirements are highly uncertain. If stakeholders cannot agree on what they need, writing detailed specifications is a waste of effort. Use prototypes and iterative feedback to clarify requirements first.

- Time is extremely constrained and the cost of formal specification is genuinely prohibitive. This is rare in practice, but if you truly have hours rather than days, a lightweight version of spec-first practices (writing a short, clear prompt that acts as a micro-specification) may be sufficient.

The key insight is that Spec-Driven Development is not all or nothing. It is a spectrum. At one end is informal prompting with no structure. At the other end is rigorous specification with formal verification. Most projects benefit from something in the middle: clear, structured specifications that are detailed enough to guide implementation and verify correctness, but lightweight enough to write and maintain without excessive overhead.

The next chapter grounds this methodology in established requirements engineering practice, showing how classic principles and techniques apply in the AI-augmented context. Understanding these foundations is essential because Spec-Driven Development does not invent new ways of thinking about requirements. It adapts existing, well-understood principles for a new tooling environment.

Chapter 3: Foundations: Requirements Engineering for the AI Era

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

A Brief History of Requirements Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Functional Versus Nonfunctional Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Stakeholder Analysis and Requirement Elicitation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Requirements Quality Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

How AI Changes the Requirements Engineering Game

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 4: Structuring Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

User Stories, Use Cases, and User Journeys

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Feature Decomposition and Prioritization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Writing Clear, Unambiguous Requirement Statements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Capturing Constraints and Edge Cases Explicitly

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Requirement Templates and Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Organizing Requirements for Traceability

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 5: Domain Modeling for AI-Assisted Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

What Is a Domain Model and Why It Matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Domain-Driven Design Concepts Adapted for AI Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Entities, Value Objects, Relationships, and Invariants

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Event Storming and Bounded Contexts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Writing Domain Models AI Agents Can Consume

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Maintaining Domain Models Across Project Evolution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 6: Functional Specifications: From Requirements to Blueprints

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

The Anatomy of a Functional Specification

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Behavioral Specifications and State Transitions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Writing Acceptance Criteria That Are Truly Testable

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Scenario-Based Specification with Given-When-Then

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Specifying APIs, Data Flows, and Side Effects

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Examples of Complete Functional Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 7: Nonfunctional Specifications: Quality, Constraints, and Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Performance Specifications and Benchmarks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Security and Privacy Requirements as Specs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Reliability, Availability, and Fault Tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Scalability and Capacity Planning in Specs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Usability, Accessibility, and Observability Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Encoding Trade-offs and Design Decisions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 8: Specification Formats and Languages

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Markdown-Based Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Structured Formats: YAML, JSON, TOML

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Formal and Semi-Formal Specification Languages

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Behavior-Driven Development and Cucumber/Gherkin

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

OpenAPI, AsyncAPI, and Contract-First Specification

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Choosing the Right Format for Your Context

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 9: System Architecture as Specification

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Architecture as a First-Class Specification Artifact

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

High-Level Design Documents for AI Agents

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Component Diagrams and Interaction Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Data Architecture and Schema Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Infrastructure and Deployment Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Documenting Architectural Decisions (ADRs)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 10: API and Data Contracts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Why Contracts Matter for AI Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

RESTful API Specifications with OpenAPI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

GraphQL Schemas and Code Generation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Database Schemas as Enforceable Contracts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Message Queues, Events, and Event-Driven Contracts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Versioning Strategies and Backward Compatibility

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 11: The Core Project: A Real-World Application

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Problem Statement and Project Scope

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Technology Stack Selection and Justification

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Requirements Document for the Core Project

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Functional Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Nonfunctional Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementsstoworkingsoftwarewithai>.

Domain Model for the Task Management System

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementsstoworkingsoftwarewithai>.

Initial Architecture and System Context

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementsstoworkingsoftwarewithai>.

Repository Structure and Project Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementsstoworkingsoftwarewithai>.

Chapter 12: Task Decomposition and Implementation Planning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

From Specifications to Tasks

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Work Breakdown Structures for AI Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Task Size, Granularity, and Context Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Dependency Mapping and Execution Ordering

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Writing Task Specifications AI Agents Can Follow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Project Tracking and Progress Monitoring

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 13: Configuring AI Agents for Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

AI Coding Tools Landscape and Agent Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

System Prompts and Agent Instructions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Configuring Code Style, Conventions, and Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Environment Setup, Tooling, and Sandboxing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Managing Context and Project Knowledge

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Safety, Guardrails, and Access Controls

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 14: Context Engineering and Prompt Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

What Is Context Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Including Specifications in AI Context Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Prompt Patterns for Specification-Driven Coding

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Few-Shot Examples and Reference Implementations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Handling Large Codebases and Split Context

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Iterative Refinement of Prompts and Instructions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 15: Generating Code from Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

The Generation Workflow Step by Step

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Building the Project Foundation with AI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Implementing Domain Models and Business Logic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Generating API Endpoints and Controllers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Writing Database Migrations and Data Access

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Reviewing and Correcting AI-Generated Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 16: Testing as Verification Against Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Test-Driven Development and AI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Generating Tests from Acceptance Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Unit Tests, Integration Tests, and End-to-End Tests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Property-Based Testing and Fuzzing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Contract Testing for APIs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Coverage, Mutation Testing, and Quality Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 17: Debugging, Refactoring, and Iteration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

When AI Gets It Wrong: Common Failure Modes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Debugging AI-Generated Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Using Specifications as Debugging Anchors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Refactoring with AI Safely

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Handling Regressions and Specification Changes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Code Review Practices for AI-Assisted Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 18: Integration, Deployment, and Observability

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Continuous Integration for Spec-Driven Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Deployment Specifications and Infrastructure as Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Environment Configuration and Secrets Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Observability: Logging, Metrics, Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Runtime Verification and Canary Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

The Core Project in Production

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 19: Multi-Agent Orchestration and Parallel Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

When Single Agents Are Not Enough

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Role-Based Agent Orchestration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Parallel Development Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Conflict Resolution and Merge Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Scaling Development with Agent Swarms

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Case Study: Parallel Development of TaskFlow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 20: Specification Management and Evolution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Living Documents Versus Frozen Specs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Versioning Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Change Management and Impact Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Detecting and Preventing Specification Drift

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Maintaining Traceability from Requirements to Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Governance and Approval Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 21: Advanced Topics and Large-Scale Practice

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Modernizing Legacy Systems with Spec-Driven Approaches

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Large Monorepos and Cross-Project Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Distributed Systems and Microservices Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Security, Compliance, and Privacy at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Managing Technical Debt with AI Assistance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Organizational Adoption and Team Practices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Chapter 22: Comparing Tools, Frameworks, and Platforms

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

GitHub Spec Kit: Workflow, Features, Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Amazon Kiro: Workflow, Features, Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Cursor, Windsurf, and IDE-Integrated AI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Claude Code, Devin, and Autonomous Agents

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Open-Source Frameworks and Community Tools

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

How to Evaluate and Choose Tools for Your Team

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Conclusion: The Future of Spec-Driven Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-drivendevelopmentfromrequirementstoworkingsoftwarewithai>.

Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-driven-development-from-requirements-to-working-software-with-ai>.

References

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/spec-driven-development-from-requirements-to-working-software-with-ai>.