

Table of Contents

Preface	xxiii
Free benefits with your book	xxix
Chapter 1: Getting Started with Xcode 26 and AI-Enhanced Development	1
Technical requirements	2
Setting up your development environment	2
Installing Xcode 26 with Xcodes • 3	
<i>Why use Xcodes?</i> • 3	
<i>Installing Xcodes</i> • 3	
<i>Installing Xcode 26</i> • 4	
Exploring Xcode 26's enhanced features • 5	
Understanding AI in software development	10
A pragmatic approach to understanding LLMs • 10	
Understanding tokens • 11	
<i>From tokens to neural network processing</i> • 12	
<i>Tokenization tools and visualization</i> • 12	
<i>Token calculation across languages and models</i> • 15	
<i>Why token count matters for iOS developers</i> • 15	
<i>Practical token optimization strategies</i> • 16	
LLM integration in Xcode 26	17
Integrating ChatGPT • 18	
Integrating Claude • 21	

Integration approaches • 21

Claude subscription setup • 21

Claude API integration • 22

API key setup and configuration • 22

Xcode 26 configuration (plug and play) • 23

Real-world cost analysis • 26

Usage considerations • 26

Token management best practices • 26

Claude Code • 27

Claude Code's hidden architecture • 28

Prerequisites and installation • 29

Integration strategy recommendations • 32

GitHub Copilot for Xcode 33

System permissions configuration • 36

Understanding the chat interface • 38

Initial analysis attempt • 39

Successful code analysis with GPT-5 • 41

Summary 43

Chapter 2: Swift Concurrency Through AI-Assisted Dialog 45

Technical requirements 46

Getting started with Swift Concurrency 46

The completion handler pattern • 47

AI-assisted migration • 49

Choosing your AI partner • 49

Crafting the transformation prompt • 50

The transformed code • 51

Xcode 26's integrated AI tools • 53

The contextual coding tools • 53

Understanding through explanation • 54

Documentation generation • 55

Rolling back changes • 57	
Structured concurrency • 58	
Transforming individual functions to async/await • 61	
Actor isolation	73
Handling multiple errors independently • 78	
The problem: all-or-nothing error handling • 79	
The solution: return errors separately • 79	
Using the error information • 80	
Testing all error scenarios • 81	
Ensuring concurrency safety	83
Understanding data races • 83	
The problem: a classic data race • 83	
Before Swift 6: manual synchronization • 84	
Swift 6: compile-time detection • 85	
Practical example: Fixing data races with Xcode • 85	
Be mindful of LLM suggestions • 88	
The modern solution: Mutex (SE-0433)	89
Why Mutex is better than actors for this case • 90	
How Mutex works • 91	
Understanding race conditions • 92	
Race condition without data race: the banking example • 92	
The correct solution: atomic check-and-act • 93	
Understanding atomicity • 95	
Common concurrency mistakes: mental models, not syntax • 97	
Summary	99
Chapter 3: SwiftUI, iOS 26 Innovations, and AI-Enhanced Development	101
Technical requirements	102
Understanding MCP	102
Installing and configuring the ios-simulator-mcp server	103
Architecture and dependencies • 104	

Installing IDB • 105	
Installing the MCP server • 106	
Verifying the installation • 107	
<i>Method 1 – Using the /mcp command • 107</i>	
<i>Method 2 – Inspecting the configuration file • 108</i>	
Common issues and their solutions • 109	
<i>IDB connection failures • 109</i>	
<i>Path issues on Apple Silicon • 109</i>	
Selecting and inspecting ios-simulator-mcp • 110	
<i>The complete tool inventory • 110</i>	
<i>Understanding the tool manifest • 111</i>	
Critical – managing sessions and token consumption • 111	
<i>How I hit my quota • 112</i>	
<i>Understanding Claude Code Desktop usage • 112</i>	
<i>Advanced monitoring with ccusage • 113</i>	
<i>Live monitoring for high-risk sessions • 114</i>	
<i>Best practices for sustainable usage • 115</i>	
Detecting accessibility issues with MCP 116	
The scenario – a notes app with hidden accessibility problems • 116	
<i>The application structure • 117</i>	
Running the accessibility audit with Claude and MCP • 118	
<i>Claude’s detailed audit report • 118</i>	
The iterative fix process • 121	
<i>Dynamic Tools – intelligent tool discovery • 123</i>	
<i>Final verification and compliance • 123</i>	
Integrating Claude Code with GitHub 126	
Add the Claude Code GitHub workflow • 132	
Creating your first pull request with Claude • 135	
<i>Intelligent .gitignore management • 137</i>	
<i>Claude reviews your pull request • 140</i>	

Leveraging LLMs with MCP for accessibility reviews – with a critical eye	144
Understanding the semantic difference between Label, Value, and Hint • 144	
Detecting missing business logic • 145	
Rich text made easy – TextEditor improvements in iOS 26	146
Summary	150
Chapter 4: Test-Driven Development with AI and Claude Code	151
Technical requirements	152
What is TDD?	153
Using Claude Code for TDD	153
Adding memory to Claude Code • 154	
Initializing Claude Code • 154	
Understanding the generated CLAUDE.md file • 156	
Refining CLAUDE.md with TDD intent • 157	
Writing the first test: an empty shopping cart	159
Triangulation	162
Triangulation in practice • 163	
<i>Adding the first product • 163</i>	
<i>Resisting early generalization • 165</i>	
<i>Validating generalization with multiple items • 167</i>	
Using Claude Code in planning mode for test refactoring	169
Why planning mode matters • 169	
Expectation: detecting redundancy • 169	
Final tests after triangulation • 174	
TPP	177
Why TPP matters • 177	
Understanding TPP transformations • 177	
<i>From nothing to something • 178</i>	
<i>From constants to logic • 178</i>	
<i>From simple to structured code • 178</i>	
<i>From scalars to collections • 178</i>	

From inline code to abstraction • 178	
Why order matters • 179	
Encoding TPP into CLAUDE.md • 179	
Applying transformations • 180	
Applying Transformation #1: $\{\}$ $\rightarrow$ constant • 181	
Applying Transformation #2: constant $\rightarrow$ variable • 184	
Applying Transformation #3: Introducing an explicit quantity parameter • 185	
Forcing boundary validation through an explicit prompt • 187	
Test Data Builders	193
Test setup obscures intent • 194	
Claude Code's response • 196	
CartBuilder versus ItemBuilder • 197	
Implementing CartBuilder • 198	
Testing network code with async/await and URLProtocol	199
Summary	202
 Chapter 5: AI-Powered Code Architecture and Legacy System Modernization	 205
Technical requirements	206
Introducing Clean Architecture	207
Clean Architecture as a set of principles • 207	
Layers are a means, not a goal • 208	
A simple three-layer view • 209	
Presentation layer • 209	
Domain layer • 209	
Data layer • 209	
Putting the domain at the center • 210	
Data transfer objects and domain models • 211	
Repositories as the boundary between data and domain • 217	
The subtle boundary of use cases • 219	
Organizing use cases around business domains • 221	

<i>A concrete use case example</i> • 222	
<i>A brief reminder of the reactive model</i> • 224	
From ad hoc prompts to structured guidance	226
Introducing agent skills • 226	
Creating the refactoring skill folder • 228	
Triggering the migration with Claude Code • 234	
Migration summary and final report • 239	
Residual effects of the migration on Package.swift • 243	
Defining an architectural skill for cleaning Package.swift	245
Scope of the package-swift-cleanup Skill.md • 245	
Launching the Package.swift cleanup with Claude Code • 247	
Dependency injection as an architectural principle	250
Why dependency injection becomes critical in modular architectures • 250	
Introducing Swinject in context • 250	
Scopes in practice: Legacy code example • 253	
The composition root pattern • 254	
Migrating from Swinject to Factory • 256	
<i>From runtime wiring to declarative composition</i> • 256	
<i>A concrete comparison: BasketRepository in Swinject and Factory</i> • 260	
Entering the modularization discussion	262
The role of the abstractions package • 263	
Modularizing with Swift Package Manager • 264	
Modularizing with Tuist • 268	
<i>Module cache and Xcode cache</i> • 269	
<i>Positioning Tuist in the modularization landscape</i> • 269	
Migrating from SPM to Tuist using an agent skill • 270	
<i>Install Tuist</i> • 270	
<i>Examine the existing SPM architecture</i> • 270	
<i>Introduce Tuist alongside SPM</i> • 271	
<i>Generating the Xcode project</i> • 272	
Summary	274

Technical requirements	278
Apple Silicon and local AI capabilities	279
Availability of Apple Foundation Models	281
Configuring LLMs: temperature, Top-K, and Top-P	287
Temperature • 287	
Top-K sampling • 288	
Top-P (nucleus) sampling • 288	
Applying these concepts to Apple Foundation Models • 288	
Executing a prompt using a chat-based interaction	289
Guiding model behavior with instructions • 294	
Streaming responses with Foundation Models • 296	
Error handling and context window management • 297	
Context window exhaustion • 297	
Structured responses with @Generable and @Guide	300
Declaring the output schema • 300	
Adding semantic constraints and intent • 301	
Extending Foundation Models with tools	305
Implementing a tool with Foundation Models • 307	
Returning structured tool output with GeneratedContent • 311	
Consuming tools through the language model session • 312	
Fine-tuning Foundation Models	314
Adapter-based fine-tuning workflow • 316	
Launching adapter training • 320	
Exporting the adapter and Python version constraints • 322	
Consuming a Foundation Model adapter in an application • 323	
Summary	328

Technical requirements	332
Understanding MCP architecture	333
MCP Host • 333	
MCP Client • 334	
MCP Server • 334	
<i>Beyond tools: Server features in MCP • 338</i>	
Communication modes in MCP: Stdio vs streamable HTTP (SSE) • 340	
<i>Stdio-based communication • 341</i>	
<i>Streamable HTTP (server-sent events) • 341</i>	
<i>JSON-RPC message structure • 342</i>	
Implementing a stdio-based MCP server in Swift	344
Adding the Swift MCP SDK dependency • 346	
Understanding the WeatherTool implementation • 347	
Implementing the tool execution method • 351	
Bootstrapping the MCP Server • 356	
Testing the MCP Server locally • 364	
Deploying a Swift-based MCP Server to npm	365
Compiling the Swift binary • 365	
The package.json manifest • 368	
The launcher script (run.js) • 369	
Authenticating with npm • 370	
Publishing the package to the npm registry • 371	
Looking beyond the Weather Server	374
Summary	376

Chapter 8: AI-Assisted Reviews, RFCs, and Governance	379
Technical requirements	381
The role of an iOS architect in the age of AI	382
Documenting architectural decisions with ADRs	383
The structure of ADR • 384	
RFC and architectural collaboration • 386	
Analyzing ADR-001: authentication architecture • 387	
Analyzing the codebase with SonarQube Cloud	389
Connecting a repository to SonarQube Cloud • 390	
Enabling AI code assurance in SonarQube Cloud • 394	
Configuring SonarQube analysis through CI/CD • 396	
Configuring the SonarQube project • 402	
Alternatives to sonar-project.properties • 404	
Investigating the detected security issues • 405	
Reviewing the SonarQube analysis results • 407	
Enabling AI CodeFix in SonarQube Cloud • 411	
Using Claude code agent teams to address SonarQube findings • 413	
Using an AI developer team to address SonarQube findings	413
Installing and configuring tmux for agent teams • 414	
Authentication security remediation • 416	
Creating the AI developer team • 421	
Installing the SonarQube MCP server • 428	
MCP-guided AI remediation workflow • 431	
AI-assisted security auditing with Claude Code Security	438
Summary	441
Chapter 9: Agentic Development in Xcode	443
Technical requirements	444
Configuring Codex in Xcode	445
Using Claude agent	448

Extending agents with custom skills	450
The Agents directory • 451	
The ClaudeAgentConfig directory • 451	
The codex directory and custom skills • 453	
Integrating Apple's MCP server	456
Integrating Gemini CLI with Xcode 26.3	462
Planning mode in practice	465
Summary	472
 Chapter 10: Unlock Your Exclusive Benefits	 475
 Unlock this Book's Free Benefits in 3 Easy Steps	 476
 Chapter 11: Glossary and References	 479
 Glossary	 479
References	480
Books • 480	
Articles and conference papers • 481	
 Other Books You May Enjoy	 485
 Index	 489