



Robert C. Martin Series

Code That Fits in Your Head

Heuristics for Software Engineering



Mark Seemann

Foreword by Robert C. Martin

Praise for **Code That Fits in Your Head**

“We progress in software by standing on the shoulders of those who came before us. Mark’s vast experience ranges from philosophical and organisational considerations right down to the precise details of writing code. In this book, you’re offered an opportunity to build on that experience. Use it.”

—Adam Ralph, *speaker, tutor, and software simplifier, Particular Software*

“I’ve been reading Mark’s blogs for years and he always manages to entertain while at the same time offering deep technical insights. *Code That Fits in Your Head* follows in that vein, offering a wealth of information to any software developer looking to take their skills to the next level.”

—Adam Tornhill, *founder of CodeScene, author of Software Design X-Rays and Your Code as a Crime Scene*

“My favorite thing about this book is how it uses a single code base as a working example. Rather than having to download separate code samples, you get a single Git repository with the entire application. Its history is hand-crafted to show the evolution of the code alongside the concepts being explained in the book. As you read about a particular principle or technique, you’ll find a direct reference to the commit that demonstrates it in practice. Of course, you’re also free to navigate the history at your own leisure, stopping at any stage to inspect, debug, or even experiment with the code. I’ve never seen this level of interactivity in a book before, and it brings me special joy because it takes advantage of Git’s unique design in a new constructive way.”

—Enrico Campidoglio, *independent consultant, speaker and Pluralsight author*

“Mark Seemann not only has decades of experience architecting and building large software systems, but is also one of the foremost thinkers on how to scale and manage the complex relationship between such systems and the teams that build them.”

—Mike Hadlow, *freelance software consultant and blogger*

Code That Fits in Your Head: Heuristics for Software Engineering

Table of Contents

Cover

Half Title

Title Page

Copyright Page

Contents

Series Editor Foreword

Preface

About the Author

PART I: Acceleration

Chapter 1 Art or Science?

1.1 Building a House

1.1.1 The Problem with Projects

1.1.2 The Problem with Phases

1.1.3 Dependencies

1.2 Growing a Garden

1.2.1 What Makes a Garden Grow?

1.3 Towards Engineering

1.3.1 Software as a Craft

1.3.2 Heuristics

1.3.3 Earlier Notions of Software Engineering

1.3.4 Moving Forward with Software Engineering

1.4 Conclusion

Chapter 2 Checklists

Table of Contents

2.1 An Aid to Memory

2.2 Checklist for a New Code Base

2.2.1 Use Git

2.2.2 Automate the Build

2.2.3 Turn On all Error Messages

2.3 Adding Checks to Existing Code Bases

2.3.1 Gradual Improvement

2.3.2 Hack Your Organisation

2.4 Conclusion

Chapter 3 Tackling Complexity

3.1 Purpose

3.1.1 Sustainability

3.1.2 Value

3.2 Why Programming Is Difficult

3.2.1 The Brain Metaphor

3.2.2 Code Is Read More Than Its Written

3.2.3 Readability

3.2.4 Intellectual Work

3.3 Towards Software Engineering

3.3.1 Relationship to Computer Science

3.3.2 Humane Code

3.4 Conclusion

Chapter 4 Vertical Slice

4.1 Start with Working Software

4.1.1 From Data Ingress to Data Persistence

4.1.2 Minimal Vertical Slice

4.2 Walking Skeleton

4.2.1 Characterisation Test

4.2.2 Arrange Act Assert

4.2.3 Moderation of Static Analysis

4.3 Outside-in

4.3.1 Receive JSON

Table of Contents

- 4.3.2 Post a Reservation
- 4.3.3 Unit Test
- 4.3.4 DTO and Domain Model
- 4.3.5 Fake Object
- 4.3.6 Repository Interface
- 4.3.7 Create in Repository
- 4.3.8 Configure Dependencies

4.4 Complete the Slice

- 4.4.1 Schema
- 4.4.2 SQL Repository
- 4.4.3 Configuration with Database
- 4.4.4 Perform a Smoke Test
- 4.4.5 Boundary Test with Fake Database

4.5 Conclusion

Chapter 5 Encapsulation

5.1 Save the Data

- 5.1.1 The Transformation Priority Premise
- 5.1.2 Parametrised Test
- 5.1.3 Copy DTO to Domain Model

5.2 Validation

- 5.2.1 Bad Dates
- 5.2.2 Red Green Refactor
- 5.2.3 Natural Numbers
- 5.2.4 Postels Law

5.3 Protection of Invariants

- 5.3.1 Always Valid

5.4 Conclusion

Chapter 6 Triangulation

6.1 Short-Term versus Long-Term Memory

- 6.1.1 Legacy Code and Memory

6.2 Capacity

- 6.2.1 Overbooking

Table of Contents

- 6.2.2 The Devils Advocate
- 6.2.3 Existing Reservations
- 6.2.4 Devils Advocate versus Red Green Refactor
- 6.2.5 When Do You Have Enough Tests?

6.3 Conclusion

Chapter 7 Decomposition

7.1 Code Rot

- 7.1.1 Thresholds
- 7.1.2 Cyclomatic Complexity
- 7.1.3 The 80/24 Rule

7.2 Code That Fits in Your Brain

- 7.2.1 Hex Flower
- 7.2.2 Cohesion
- 7.2.3 Feature Envy
- 7.2.4 Lost in Translation
- 7.2.5 Parse, Dont Validate
- 7.2.6 Fractal Architecture
- 7.2.7 Count the Variables

7.3 Conclusion

Chapter 8 API Design

8.1 Principles of API Design

- 8.1.1 Affordance
- 8.1.2 Poka-Yoke
- 8.1.3 Write for Readers
- 8.1.4 Favour Well-Named Code over Comments
- 8.1.5 X Out Names
- 8.1.6 Command Query Separation
- 8.1.7 Hierarchy of Communication

8.2 API Design Example

- 8.2.1 Maître D
- 8.2.2 Interacting with an Encapsulated Object
- 8.2.3 Implementation Details

Table of Contents

8.3 Conclusion

Chapter 9 Teamwork

9.1 Git

9.1.1 Commit Messages

9.1.2 Continuous Integration

9.1.3 Small Commits

9.2 Collective Code Ownership

9.2.1 Pair Programming

9.2.2 Mob Programming

9.2.3 Code Review Latency

9.2.4 Rejecting a Change Set

9.2.5 Code Reviews

9.2.6 Pull Requests

9.3 Conclusion

PART II: Sustainability

Chapter 10 Augmenting Code

10.1 Feature Flags

10.1.1 Calendar Flag

10.2 The Strangler Pattern

10.2.1 Method-Level Strangler

10.2.2 Class-Level Strangler

10.3 Versioning

10.3.1 Advance Warning

10.4 Conclusion

Chapter 11 Editing Unit Tests

11.1 Refactoring Unit Tests

11.1.1 Changing the Safety Net

11.1.2 Adding New Test Code

11.1.3 Separate Refactoring of Test and Production Code

11.2 See Tests Fail

11.3 Conclusion

Table of Contents

Chapter 12 Troubleshooting

12.1 Understanding

12.1.1 Scientific Method

12.1.2 Simplify

12.1.3 Rubber Ducking

12.2 Defects

12.2.1 Reproduce Defects as Tests

12.2.2 Slow Tests

12.2.3 Non-deterministic Defects

12.3 Bisection

12.3.1 Bisection with Git

12.4 Conclusion

Chapter 13 Separation of Concerns

13.1 Composition

13.1.1 Nested Composition

13.1.2 Sequential Composition

13.1.3 Referential Transparency

13.2 Cross-Cutting Concerns

13.2.1 Logging

13.2.2 Decorator

13.2.3 What to Log

13.3 Conclusion

Chapter 14 Rhythm

14.1 Personal Rhythm

14.1.1 Time-Boxing

14.1.2 Take Breaks

14.1.3 Use Time Deliberately

14.1.4 Touch Type

14.2 Team Rhythm

14.2.1 Regularly Update Dependencies

14.2.2 Schedule Other Things

14.2.3 Conways Law

Table of Contents

14.3 Conclusion

Chapter 15 The Usual Suspects

15.1 Performance

15.1.1 Legacy

15.1.2 Legibility

15.2 Security

15.2.1 STRIDE

15.2.2 Spoofing

15.2.3 Tampering

15.2.4 Repudiation

15.2.5 Information Disclosure

15.2.6 Denial of Service

15.2.7 Elevation of Privilege

15.3 Other Techniques

15.3.1 Property-Based Testing

15.3.2 Behavioural Code Analysis

15.4 Conclusion

Chapter 16 Tour

16.1 Navigation

16.1.1 Seeing the Big Picture

16.1.2 File Organisation

16.1.3 Finding Details

16.2 Architecture

16.2.1 Monolith

16.2.2 Cycles

16.3 Usage

16.3.1 Learning from Tests

16.3.2 Listen to Your Tests

16.4 Conclusion

Appendix A: List of Practices

A.1 The 50/72 Rule

Table of Contents

- A.2 The 80/24 Rule
- A.3 Arrange Act Assert
- A.4 Bisection
- A.5 Checklist for A New Code Base
- A.6 Command Query Separation
- A.7 Count the Variables
- A.8 Cyclomatic Complexity
- A.9 Decorators for Cross-Cutting Concerns
- A.10 Devils Advocate
- A.11 Feature Flag
- A.12 Functional Core, Imperative Shell
- A.13 Hierarchy of Communication
- A.14 Justify Exceptions from the Rule
- A.15 Parse, Dont Validate
- A.16 Postels Law
- A.17 Red Green Refactor
- A.18 Regularly Update Dependencies
- A.19 Reproduce Defects as Tests
- A.20 Review Code
- A.21 Semantic Versioning
- A.22 Separate Refactoring of Test and Production Code
- A.23 Slice
- A.24 Strangler
- A.25 Threat-Model
- A.26 Transformation Priority Premise
- A.27 X-driven Development
- A.28 X Out Names

Table of Contents

Bibliography

Index