

DAVID FARLEY



MODERN SOFTWARE ENGINEERING

Doing What Works to
Build Better Software Faster

Foreword by TRISHA GEE



Praise for *Modern Software Engineering*

"*Modern Software Engineering* gets it right and describes the ways skilled practitioners actually engineer software today. The techniques Farley presents are not rigid, prescriptive, or linear, but they are disciplined in exactly the ways software requires: empirical, iterative, feedback-driven, economical, and focused on running code."

—Glenn Vanderburg, Director of Engineering at Nubank

"There are lots of books that will tell you how to follow a particular software engineering practice; this book is different. What Dave does here is set out the very essence of what defines software engineering and how that is distinct from simple craft. He explains why and how in order to master software engineering you must become a master of both learning and of managing complexity, how practices that already exist support that, and how to judge other ideas on their software engineering merits. This is a book for anyone serious about treating software development as a true engineering discipline, whether you are just starting out or have been building software for decades."

—Dave Hounslow, Software Engineer

"These are important topics and it's great to have a compendium that brings them together as one package."

—Michael Nygard, author of *Release IT*, professional programmer,
and software architect

"I've been reading the review copy of Dave Farley's book and it's what we need. It should be required reading for anyone aspiring to be a software engineer or who wants to master the craft. Pragmatic, practical advice on professional engineering. It should be required reading in universities and bootcamps."

—Bryan Finster, Distinguished Engineer and
Value Stream Architect at USAF Platform One

Modern Software Engineering: Doing What Works to Build Better Software Faster

Table of Contents

Cover

Half Title

Title

Copyright

Dedication

Contents

Foreword

Preface

Acknowledgments

About the Author

Part I: What Is Software Engineering?

1 Introduction

EngineeringThe Practical Application of Science

What Is Software Engineering?

Reclaiming Software Engineering

How to Make Progress

The Birth of Software Engineering

Shifting the Paradigm

Summary

2 What Is Engineering?

Production Is Not Our Problem

Design Engineering, Not Production Engineering

Table of Contents

A Working Definition of Engineering
Engineering != Code
Why Does Engineering Matter?
The Limits of Craft
Precision and Scalability
Managing Complexity
Repeatability and Accuracy of Measurement
Engineering, Creativity, and Craft
Why What We Do Is Not Software Engineering
Trade-Offs
The Illusion of Progress
The Journey from Craft to Engineering
Craft Is Not Enough
Time for a Rethink?
Summary

3 Fundamentals of an Engineering Approach

An Industry of Change?
The Importance of Measurement
Applying Stability and Throughput
The Foundations of a Software Engineering Discipline
Experts at Learning
Experts at Managing Complexity
Summary

Part II: Optimize for Learning

4 Working Iteratively

Practical Advantages of Working Iteratively
Iteration as a Defensive Design Strategy
The Lure of the Plan
Practicalities of Working Iteratively

Table of Contents

Summary

5 Feedback

A Practical Example of the Importance of Feedback

Feedback in Coding

Feedback in Integration

Feedback in Design

Feedback in Architecture

Prefer Early Feedback

Feedback in Product Design

Feedback in Organization and Culture

Summary

6 Incrementalism

Importance of Modularity

Organizational Incrementalism

Tools of Incrementalism

Limiting the Impact of Change

Incremental Design

Summary

7 Empiricism

Grounded in Reality

Separating Empirical from Experimental

I Know That Bug!

Avoiding Self-Deception

Inventing a Reality to Suit Our Argument

Guided by Reality

Summary

8 Being Experimental

What Does Being Experimental Mean?

Feedback

Table of Contents

Hypothesis

Measurement

Controlling the Variables

Automated Testing as Experiments

Putting the Experimental Results of Testing into Context

Scope of an Experiment

Summary

Part III: Optimize for Managing Complexity

9 Modularity

Hallmarks of Modularity

Undervaluing the Importance of Good Design

The Importance of Testability

Designing for Testability Improves Modularity

Services and Modularity

Deployability and Modularity

Modularity at Different Scales

Modularity in Human Systems

Summary

10 Cohesion

Modularity and Cohesion: Fundamentals of Design

A Basic Reduction in Cohesion

Context Matters

High-Performance Software

Link to Coupling

Driving High Cohesion with TDD

How to Achieve Cohesive Software

Costs of Poor Cohesion

Cohesion in Human Systems

Summary

Table of Contents

11 Separation of Concerns

- Dependency Injection
- Separating Essential and Accidental Complexity
- Importance of DDD
- Testability
- Ports & Adapters
- When to Adopt Ports & Adapters
- What Is an API?
- Using TDD to Drive Separation of Concerns
- Summary

12 Information Hiding and Abstraction

- Abstraction or Information Hiding
- What Causes Big Balls of Mud?
- Organizational and Cultural Problems
- Technical Problems and Problems of Design
- Fear of Over-Engineering
- Improving Abstraction Through Testing
- Power of Abstraction
- Leaky Abstractions
- Picking Appropriate Abstractions
- Abstractions from the Problem Domain
- Abstract Accidental Complexity
- Isolate Third-Party Systems and Code
- Always Prefer to Hide Information
- Summary

13 Managing Coupling

- Cost of Coupling
- Scaling Up
- Microservices

Table of Contents

- Decoupling May Mean More Code
- Loose Coupling Isn't the Only Kind That Matters
- Prefer Loose Coupling
- How Does This Differ from Separation of Concerns?
- DRY Is Too Simplistic
- Async as a Tool for Loose Coupling
- Designing for Loose Coupling
- Loose Coupling in Human Systems
- Summary

Part IV: Tools to Support Engineering in Software

14 The Tools of an Engineering Discipline

- What Is Software Development?
- Testability as a Tool
- Measurement Points
- Problems with Achieving Testability
- How to Improve Testability
- Deployability
- Speed
- Controlling the Variables
- Continuous Delivery
- General Tools to Support Engineering
- Summary

15 The Modern Software Engineer

- Engineering as a Human Process
- Digitally Disruptive Organizations
- Outcomes vs. Mechanisms
- Durable and Generally Applicable
- Foundations of an Engineering Discipline
- Summary

Table of Contents

Index