Here is the first book to emphasize fundamental principles rather than specific software engineering tools and techniques.

Fundamentals of Software Engineering provides selective, in-depth coverage of the fundamentals of software engineering, stressing principles, methods, and rigorous formal and informal approaches. It stresses the importance of rigor in the practice of software engineering and emphasizes the important principles that can be applied independently of the life cycle model.

Numerous clear examples and exercises showing the application of principles to practical real-world problems are also included and make this an excellent self-study guide.

CONTENT HIGHLIGHTS

* uses small examples to illustrate principles and large case studies to show application and combination of principles in more realistic situations.

* emphasizes formality, design for change, and incrementality, using case studies to compare and contrast the different formalisms

* draws analogies to other engineering disciplines

* covers design, specification, verification and validation, the software process, management, and environments.

CONTENTS

PREFACE xi

1 SOFTWARE ENGINEERING: A PREVIEW 1

1.1 The Role of Software Engineering in System Design 2
1.2 A History of Software Engineering 3
1.3 The Role of the Software Engineer 5
1.4 The Software Life Cycle 6
1.5 The Relationship of Software Engineering to Other Areas of Computer Science 8
  1.5.1 Programming Languages 8
  1.5.2 Operating Systems 10
  1.5.3 Data Bases 10
  1.5.4 Artificial Intelligence 11
  1.5.5 Theoretical Models 12
1.6 The Relationship of Software Engineering to Other Disciplines 13
  1.6.1 Management Science 13
  1.6.2 Systems Engineering 13
## 5 SOFTWARE SPECIFICATION

5.1 The Uses of Specifications 152
5.2 Specification Qualities 154
5.3 Classification of Specification Styles 157
5.4 Verification of Specifications 159
5.5 Operational Specifications 160
  5.5.1 Data Flow Diagrams: Specifying Functions of Information Systems 161
  5.5.2 Finite State Maclu'nes: Describing Control Flow 167
  5.5.3 Petri Nets: Specifying Asynchronous Systems 174
5.6 Descriptive Specifications 199
  5.6.1 Entity-relationship Diagrams 199
  5.6.2 Logic Specifications 202
  5.6.3 Algebraic Specifications 219
5.7 Building and Using Specifications in Practice 226
  5.7.1 Requirements for Specification Notations 227
  5.7.2 Building Modular Specifications 230
  5.7.3 Specifications for the End User 241
5.8 Concluding Remarks 242

## 6 SOFTWARE VERIFICATION

6.1 Goals and Requirements of Verification 256
  6.1.1 Everything Must Be Verified 257
  6.1.2 The Results of Verification May Not Be Binary 257
  6.1.3 Verification May Be Objective or Subjective 258
  6.1.4 Even Implicit Qualities Must Be Verified 259
6.2 Approaches to Verification 260
6.3 Testing 260
  6.3.1 Goals for Testing 262
  6.3.2 Theoretical Foundations of Testing 264
  6.3.3 Empirical Testing Principles 266
  6.3.4 Testing in the Small 269
  6.3.5 Testing in the Large 288
  6.3.6 Separate Concerns in the Testing Activity 293
  6.3.7 Testing Concurrent and Real-time Systems 295
6.4 Analysis 297
  6.4.1 Informal Analysis Techniques 298
  6.4.2 Correctness Proofs 301
6.5 Symbolic Execution 318
  6.5.1 Basic Concepts of Symbolic Execution 320
  6.5.2 Programs with Arrays 323
  6.5.3 The Use of Symbolic Execution in Testing 325
  6.5.4 Symbolic Execution of Concurrent Programs 327
6.6 Debugging 329
6.7 Verifying Other Software Properties 334
  6.7.1 Verifying Performance 334
  6.7.2 Verifying Reliability 335
  6.7.3 Source-code Metrics 339
6.8 Concluding Remarks 344

## 7 THE SOFTWARE PRODUCTION PROCESS

7.1 Software Production Process Models 358
  7.1.1 Waterfall Model 360
  7.1.2 Evolutionary Model 374
    7.1.3 Transformation Model 377
## 7. Case Studies

**7.1** Telephone Switching System 384
**7.2** Budget Control System 389

## 7. Organizing the Process

**7.3.1** Software Methodologies 394
**7.3.2** Configuration Management 403
**7.3.3** Software Standards 408

## 7. Concluding Remarks

Further Exercises 409
Hints and Sketchy Solutions 411

Bibliographic Notes 411

---

## 8. MANAGEMENT OF SOFTWARE ENGINEERING

**8.1** Management Functions 417

**8.2** Project Planning

**8.2.1** Software Productivity 420
**8.2.2** People and Productivity 426
**8.2.3** Cost Estimation 427

**8.3** Project Control

**8.3.1** Work Breakdown Structures 434
**8.3.2** Gantt Chants 435
**8.3.3** PERT Chants 437
**8.3.4** Dealing with Deviations from the Plan 440

**8.4** Organization

**8.4.1** Centralized-control Team Organization 441
**8.4.2** Decentralized-control Team Organization 444
**8.4.3** Mixed-control Team Organization 445
**8.4.4** An Assessment of Team Organizations 446

**8.5** Risk Management 446

**8.5.1** Typical Management Risks in Software Engineering 447

**8.6** Concluding Remarks

Further Exercises 451
Hints and Sketchy Solutions 452

Bibliographic Notes 452

---

## 9. SOFTWARE ENGINEERING TOOLS AND ENVIRONMENTS

**9.1** Historical Evolution of Tools and Environments 456

**9.2** Classification of Software Tools and Environments 458

**9.3** Representative Tools

**9.3.1** Editors 463
**9.3.2** Linkers 464
**9.3.3** Interpreters 464
**9.3.4** Code Generators 465
**9.3.5** Debuggers 466
**9.3.6** Tools Used in Software Testing 467
**9.3.7** Static Analyzers 468
**9.3.8** Userinterface Management Tools 470
**9.3.9** Configuration Management Tools 474
**9.3.10** Management Tools 477
**9.3.11** Software Engineering Infrastructures 478

**9.4** The Role of Programming Language in the Environment

**9.4.1** Procedural Versus Nonprocedural Languages 480
**9.4.2** Features of Programming in the Small 483
**9.4.3** Programming Languages and Modularity 484
**9.4.4** Object-oriented Programming Languages 487
**9.4.5** Programming Languages and the Handling of Anomalies 491
**9.4.6** Programming Languages and Concurrency 493
**9.4.7** Programming Languages and Verification 496
**9.4.8** Programming Languages and Software Design 497

**9.5** Some Sample Tools and Environments

**9.5.1** Teamwork 498
**9.5.2** The UNIR Environment 500
**9.5.3** Language-centered Environments: Smalltalk80 and KEE 504
Contribute to awsaavedra/fundamentals-of-software-engineering development by creating an account on GitHub.

Objective: This repository aims to give beginners a strong and comprehensive background in fundamental concepts and practices for software engineering. 

- **language** # Language, Object Oriented Programming, OOD, etc.
- **data-structures** # Numerous useful data structures
- **algorithms** # Sorting, shortest path, etc.
- **design-patterns** # General ways to design code for a reoccurring problem
- **version-control** # Just learn git, plenty of free resources
- **terminal** # Learn Bash (cd, ls, etc.)
1. INTRODUCTION
1.1. Evolution—From an Art Form to an Engineering Discipline
1.1.1. Evolution of an Art into an Engineering Discipline
1.1.2. Evolution Pattern for Engineering Disciplines
1.1.3. A Solution to the Software Crisis

Software architects build axioms as well, but the software world is, well, softer than mathematics: fundamental things continue to change at a rapid pace in the software world. The software development ecosystem exists in a constant state of dynamic equilibrium: while it exists in a balanced state at any given point in time, it exhibits dynamic behavior over the long term. In fact, the authors believe that we must question fundamental axioms on a regular basis, in light of improved engineering practices, operational ecosystems, software development processes—everything that makes up the messy, dynamic equilibrium where architects and developers work each day. Read more.