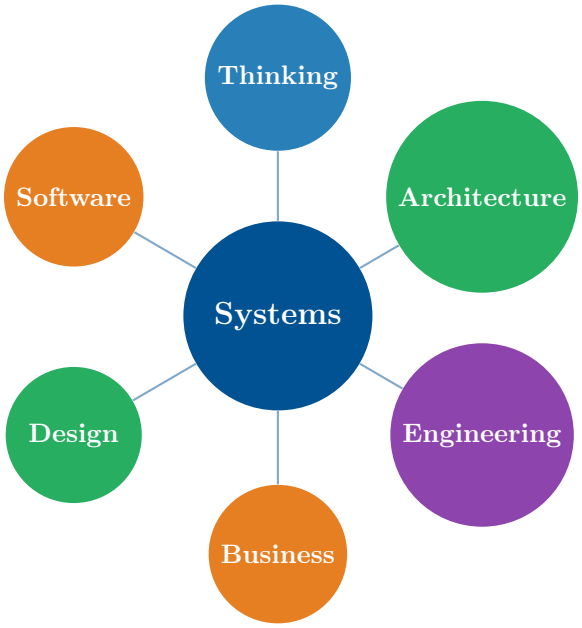
Comprehensive Curriculum in Systems Engineering & Software Architecture



A Multi-Track Professional Development Program
Integrating Theory, Practice, and Implementation

From Foundational Concepts to Advanced Applications

16 Core Texts • 4 Learning Tracks • 52+ Week Program

Contents

Chapter 1

Introduction and Program Overview

1.1 Purpose and Scope

This curriculum provides a structured, comprehensive pathway for developing expertise in systems engineering, software architecture, and business systems design. The program synthesizes insights from sixteen carefully selected texts spanning theoretical foundations, practical implementation strategies, and industry best practices.

The curriculum is designed for software engineers, technical leads, architects, and professionals seeking to develop a holistic understanding of how complex systems—both technical and organizational—are conceived, designed, built, and maintained.

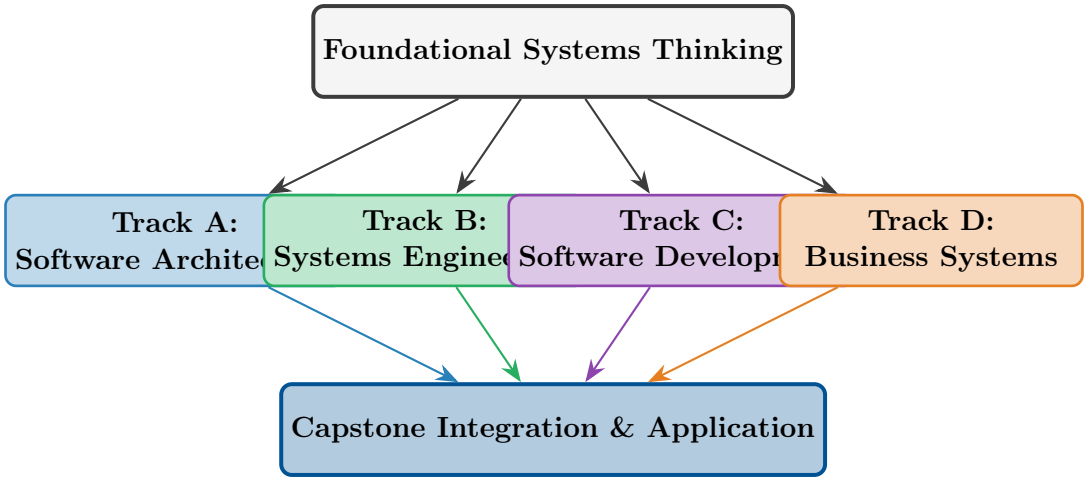
★ Learning Objectives

Upon completion of this curriculum, participants will be able to:

- Apply systems thinking principles to analyze and solve complex problems at multiple scales
- Design and architect software systems that are reliable, scalable, and maintainable
- Understand and apply distributed systems principles in real-world applications
- Build and optimize business processes and organizational systems
- Apply object-oriented design principles effectively in software development
- Navigate the human and organizational aspects of software engineering projects
- Integrate theoretical knowledge with practical implementation skills

1.2 Program Structure

The curriculum is organized into four interconnected learning tracks, each building upon foundational concepts while developing specialized expertise:



1.3 Core Reading List Overview

The curriculum integrates the following sixteen core texts, organized by primary focus area:

1.3.1 Foundational Systems Thinking

Title	Author(s)	Primary Focus
Thinking in Systems: A Primer	Donella H. Meadows	Systems dynamics and mental models
An Introduction to General Systems Thinking	Gerald M. Weinberg	General systems theory

1.3.2 Computer Systems and Software Architecture

Title	Author(s)	Primary Focus
Designing Data-Intensive Applications	Martin Kleppmann	Distributed data systems
Distributed Systems: Principles and Paradigms	Tanenbaum & van Steen	Distributed computing theory
Fundamentals of Software Architecture	Richards & Ford	Architecture styles and patterns
Operating System Concepts	Silberschatz, Galvin, & Gagne	OS internals and design

1.3.3 Software Engineering Practice

Title	Author(s)	Primary Focus
Practical Object-Oriented Design	Sandi Metz	OO design principles
The Mythical Man-Month	Frederick Brooks Jr.	Project management
Modern Software Engineering	David Farley	Contemporary practices

1.3.4 Systems Engineering

Title	Author(s)	Primary Focus
Systems Engineering Principles and Practice	Alexander Kossiakoff	SE methodology
The Art of Systems Engineering	Marco Chicherio	Practical SE application

1.3.5 Business Systems and Process Design

Title	Author(s)	Primary Focus
The E-Myth Revisited	Michael Gerber	Business systematization
Work the System	Sam Carpenter	Process documentation
Traction	Gino Wickman	Organizational structure
Scaling Up	Verne Harnish	Growth systems
Service Design Doing	Various	Process design methodology

1.4 Time Commitment and Pacing

The program is designed for approximately 52 weeks of study, assuming 10–15 hours of weekly commitment. This includes reading time, practical exercises, reflection activities, and project work. The pacing can be adjusted based on prior experience and available time.

Chapter 2

Phase 1: Foundational Systems Thinking

☞ 8 weeks ▲ Beginner to Intermediate

This foundational phase establishes the theoretical framework that underpins all subsequent learning. Systems thinking provides the mental models necessary for understanding complex technical and organizational systems, making it essential preparation for the specialized tracks.

★ Learning Objectives

- Understand the fundamental principles of systems dynamics
- Recognize feedback loops, leverage points, and emergent behavior
- Apply systems thinking to analyze real-world problems
- Develop vocabulary and frameworks for discussing complex systems
- Understand how systems can degrade and how to design for resilience

2.1 Module 1.1: Thinking in Systems

☞ 4 weeks ✎ *Thinking in Systems: A Primer* by Donella H. Meadows

About This Book: This concise guide is considered crucial for understanding problem-solving at scales ranging from personal to global. Meadows argues that many large-scale problems are fundamentally system failures, and this book develops the core skills needed to navigate an increasingly complex world for effective problem solving and decision making.

2.1.1 Week 1–2: Core Concepts

Reading Focus

- Parts I and II: System structure and basic dynamics
- Focus on understanding stocks, flows, and feedback loops
- Study the iceberg model: events, patterns, structures, mental models

Key Concepts to Master

Stocks and Flows: Understanding accumulation and rates of change as the foundation of dynamic behavior. Stocks represent the state of a system at any point in time, while flows represent the rates of change that affect stocks.

Feedback Loops: Distinguishing between reinforcing (positive) and balancing (negative) feedback loops. Reinforcing loops amplify change, while balancing loops resist it and seek equilibrium.

Delays: Recognizing how time delays between cause and effect create oscillation, overshoot, and counterintuitive system behavior.

★ Practical Exercise

Exercise 1.1: System Mapping
Select a system you interact with daily (e.g., your morning routine, a software deployment pipeline, or your team's workflow). Create a causal loop diagram that identifies:

1. At least three stocks (quantities that accumulate)
2. The flows that affect each stock
3. Two reinforcing feedback loops
4. Two balancing feedback loops
5. Any significant delays in the system

Document how these elements interact and predict what would happen if you intervened at different points.

2.1.2 Week 3–4: Leverage Points and System Archetypes

Reading Focus

- Parts III and IV: System dynamics and places to intervene
- Meadows' famous "Leverage Points" framework
- Common system archetypes and their patterns

Leverage Points Hierarchy

Meadows identifies twelve leverage points, ranked from least to most effective:

12. Constants, parameters, numbers
11. Buffer sizes and stabilizing stocks
10. Structure of material stocks and flows
9. Delays in feedback loops
8. Strength of negative feedback loops
7. Gain of positive feedback loops
6. Information flows
5. Rules of the system

4. Power to add, change, or evolve system structure

3. Goals of the system

2. Mindset or paradigm from which the system arises

1. Transcending paradigms

★ Key Insight

The most powerful leverage points are often the least intuitive. While we naturally focus on adjustable parameters (like budgets or deadlines), the most effective interventions often involve changing information flows, rules, or the underlying mental models that shape the system.

★ Practical Exercise

Exercise 1.2: Leverage Point Analysis
Revisit the system you mapped in Exercise 1.1. For each of Meadows' twelve leverage points:

1. Identify where that leverage point exists in your system

2. Assess the current state of that leverage point

3. Propose a specific intervention at that point

4. Predict the likely short-term and long-term effects

Create a prioritized list of interventions based on expected impact versus implementation difficulty.

2.2 Module 1.2: General Systems Thinking

☞ 4 weeks ✎ *An Introduction to General Systems Thinking* by Gerald M. Weinberg

About This Book: This foundational text explores general systems thinking concepts with wit and practicality. Weinberg offers profound insights for navigating complex organizational challenges, making it essential reading alongside Meadows' work as a key text in the field.

2.2.1 Week 5–6: Foundations of Observation

Reading Focus

- The problem of observation and description
- Science and its limitations in understanding complex systems
- The observer's role in defining and bounding systems

Key Concepts to Master

The Law of Medium Numbers: Systems in the "medium number" range (too complex for simple analysis, too small for statistical methods) require systems thinking approaches.

The Complementarity Principle: No single view of a complex system is complete; multiple perspectives are necessary for understanding.

The Generalized Thermodynamic Interpretation: Understanding how systems tend toward disorder and the energy required to maintain organization.

2.2.2 Week 7–8: Decomposition and Behavior

Reading Focus

- Interpreting behavior and change
- System decomposition strategies
- The relationship between structure and behavior

★ Practical Exercise

Exercise 1.3: Multi-Perspective System Analysis
Choose a complex technical system (e.g., a microservices architecture, a CI/CD pipeline, or an e-commerce platform). Analyze it from at least four different perspectives:

1. The end user's perspective
2. The developer's perspective
3. The operations team's perspective
4. The business stakeholder's perspective

For each perspective, document:

- What elements of the system are visible?
- What elements are hidden or abstracted away?
- What behaviors are most important?
- What constitutes "success" or "failure"?

Synthesize these views to identify tensions and trade-offs inherent in the system design.

2.3 Phase 1 Assessment Checkpoint

Before proceeding to the specialized tracks, participants should be able to:

✓ Draw and interpret causal loop diagrams for complex systems

✓ Identify feedback loops and predict their effects

✓ Apply the leverage points framework to real problems

✓ Recognize common system archetypes in technical and organizational contexts

✓ Articulate the observer's role in defining system boundaries

✓ Integrate multiple perspectives when analyzing systems

Chapter 3

Track A: Software Architecture and Distributed Systems

★ Track A: Software Architecture

This track develops expertise in designing and architecting software systems at scale. It covers architectural patterns, distributed systems principles, operating system concepts, and the practical skills needed to build reliable, maintainable systems.

☞ 16 weeks ▲ Intermediate to Advanced

✓ Prerequisites

- Completion of Phase 1: Foundational Systems Thinking
- Practical programming experience (any language)
- Basic understanding of networking concepts
- Familiarity with database fundamentals

★ Learning Objectives

- Understand and apply modern software architecture styles and patterns
- Design data-intensive applications with appropriate trade-offs
- Apply distributed systems principles to real-world problems
- Understand operating system internals and their impact on application design
- Make informed architectural decisions based on quality attributes

3.1 Module A.1: Fundamentals of Software Architecture

☞ 4 weeks ✎ *Fundamentals of Software Architecture* by Mark Richards & Neal Ford

About This Book: This comprehensive resource covers the different styles and practices of modern software architecture, providing a framework for understanding architectural characteristics, patterns, and decision-making processes.

3.1.1 Week 1–2: Architectural Thinking and Characteristics

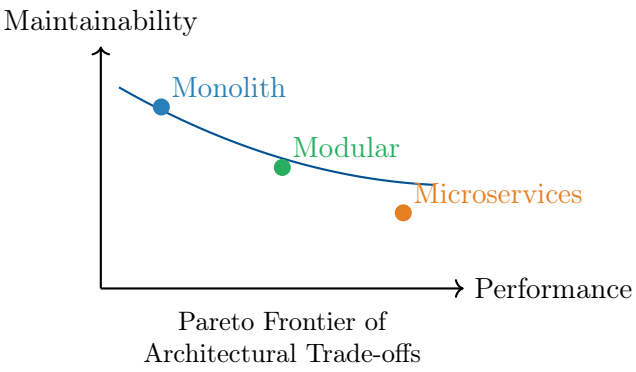
Key Concepts

Architectural Thinking: The shift from developer mindset to architect mindset involves understanding trade-offs, thinking in terms of quality attributes, and making decisions that balance competing concerns.

Architectural Characteristics: Non-functional requirements ("-ilities") such as scalability, availability, reliability, maintainability, and performance that drive architectural decisions.

Architecture Quantum: The independently deployable artifact with high functional cohesion that includes all structural elements required for the system to function.

Quality Attribute Trade-offs



3.1.2 Week 3–4: Architecture Styles and Patterns

Architecture Styles Covered

Style	Key Characteristics	Best Suited For
Layered	Separation of concerns, technical partitioning	Traditional enterprise applications
Pipeline	Filters and pipes, data transformation	Data processing, ETL
Microkernel	Core + plugins	IDEs, browsers, extensible systems
Service-Based	Coarse-grained services	Domain-driven decomposition
Event-Driven	Async communication, loose coupling	Real-time systems, IoT
Microservices	Fine-grained, independently deployable	Highly scalable, agile organizations

★ Practical Exercise

Exercise A.1: Architecture Kata
Design an architecture for an online ordering system with the following requirements:

- Support 10,000 concurrent users
- 99.9% availability requirement
- Integration with payment providers and inventory systems
- Support for mobile and web clients
- Real-time order status updates

Document your architectural decisions, including:

1. The architecture style selected and rationale
2. Key architectural characteristics prioritized
3. Component diagram showing major services
4. Trade-offs accepted and their implications
5. Fitness functions to validate the architecture

3.2 Module A.2: Designing Data-Intensive Applications

☞ 6 weeks ✎ *Designing Data-Intensive Applications* by Martin Kleppmann

About This Book: A highly popular and practical guide for building robust, scalable systems. Kleppmann provides deep insights into the principles behind reliable data systems, covering everything from data models to distributed system challenges.

3.2.1 Week 5–6: Foundations of Data Systems

Key Concepts

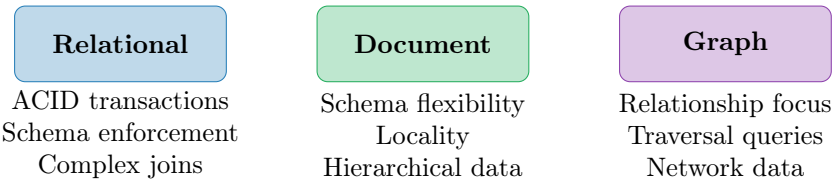
Reliability: The system continues to work correctly even when things go wrong (hardware faults, software errors, human mistakes).

Scalability: The system can handle growth in data volume, traffic, or complexity through reasonable means.

Maintainability: Different people can work on the system productively over time (operability, simplicity, evolvability).

Data Models and Query Languages

Understanding the trade-offs between relational, document, and graph models is essential for making appropriate data store choices.



3.2.2 Week 7–8: Distributed Data

Replication Strategies

Understanding replication is critical for building systems that are both available and consistent. The key replication strategies include:

Single-Leader Replication: One node accepts writes, others replicate. Provides strong consistency but limited write scalability.

Multi-Leader Replication: Multiple nodes accept writes. Better availability but requires conflict resolution.

Leaderless Replication: Any node can accept reads/writes. Maximum availability but requires careful consistency handling.

Partitioning (Sharding)

As datasets grow beyond single-node capacity, partitioning becomes essential. Key considerations include:

- Partition key selection and its impact on query patterns
- Handling hot spots and skewed workloads
- Rebalancing strategies as the system grows
- Secondary indexes across partitions

3.2.3 Week 9–10: Consistency and Consensus

The CAP Theorem and Beyond

While the CAP theorem (Consistency, Availability, Partition tolerance—pick two) is foundational, real-world systems operate on a spectrum of consistency models:

- **Linearizability:** Strongest guarantee—operations appear atomic and in real-time order
- **Sequential Consistency:** Operations appear in the same order to all observers
- **Causal Consistency:** Causally related operations are seen in order
- **Eventual Consistency:** Given enough time, all replicas converge

Consensus Algorithms

Understanding consensus (Paxos, Raft, Zab) is essential for building fault-tolerant distributed systems:

★ Key Insight

Consensus algorithms ensure that all nodes in a distributed system agree on the same values, even in the presence of failures. This is the foundation for leader election, distributed locks, atomic broadcast, and total order delivery.

★ Practical Exercise

Exercise A.2: Distributed System Design
Design a distributed key-value store that meets the following requirements:

- Store 100TB of data across 50 nodes
- Handle 100,000 reads/second and 10,000 writes/second
- Tolerate up to 3 simultaneous node failures
- Provide tunable consistency (eventual to strong)

Your design should address:

1. Partitioning strategy and rebalancing approach
2. Replication factor and consistency protocol
3. Failure detection and recovery mechanisms
4. Client API and consistency level options

3.3 Module A.3: Distributed Systems Theory

☞ 4 weeks ✎ *Distributed Systems: Principles and Paradigms* by Tanenbaum & van Steen

About This Book: A widely recognized textbook covering all key concepts, architectures, and components of distributed systems in depth. It provides the theoretical foundation for understanding how distributed systems work and why they behave as they do.

3.3.1 Week 11–12: Communication and Coordination

Key Topics

- Remote Procedure Calls (RPC) and message-oriented middleware
- Naming and name resolution in distributed systems
- Synchronization and logical clocks (Lamport clocks, vector clocks)
- Distributed mutual exclusion and election algorithms

3.3.2 Week 13–14: Fault Tolerance and Security

Key Topics

- Failure models and fault classification
- Process resilience and group communication
- Distributed commit protocols (2PC, 3PC)
- Security in distributed systems: authentication, authorization, and secure channels

★ Practical Exercise

Exercise A.3: Failure Mode Analysis
Take your key-value store design from Exercise A.2 and perform a comprehensive failure mode analysis:

1. Identify all possible failure modes (node failures, network partitions, disk failures, etc.)
2. For each failure mode, document the expected system behavior
3. Design recovery procedures for each failure type
4. Create a chaos engineering test plan to validate your fault tolerance

3.4 Module A.4: Operating System Foundations

☞ 2 weeks ✎ *Operating System Concepts* by Silberschatz, Galvin, & Gagne

About This Book: A classic, comprehensive textbook that bridges the gap between operating system concepts and actual implementations. It includes end-of-chapter problems and exercises to reinforce learning.

3.4.1 Week 15–16: OS Concepts for Architects

Essential Topics for Software Architects

While not every software architect needs to understand OS internals deeply, certain concepts directly impact application design:

- **Process and Thread Management:** Understanding scheduling, context switching, and concurrency primitives
- **Memory Management:** Virtual memory, paging, and their impact on application performance
- **I/O and File Systems:** Understanding how I/O operations work at the OS level
- **Virtualization and Containers:** How modern deployment targets differ from bare metal

★ Key Insight

Understanding OS-level concepts helps architects make better decisions about concurrency models, memory usage patterns, and deployment strategies. The abstractions provided by programming languages and frameworks rest on these OS primitives.

Chapter 4

Track B: Systems Engineering

★ Track B: Systems Engineering

This track develops expertise in the disciplined approach to designing, developing, and managing complex systems throughout their lifecycle. It covers both theoretical principles and practical application of systems engineering methodologies.

☞ 8 weeks ▲ Intermediate

✓ Prerequisites

- Completion of Phase 1: Foundational Systems Thinking
- Understanding of project management fundamentals
- Experience working on multi-disciplinary projects

★ Learning Objectives

- Understand the systems engineering lifecycle and key processes
- Apply requirements engineering techniques effectively
- Design systems using structured decomposition methods
- Manage system integration and verification
- Balance technical and programmatic concerns in system development

4.1 Module B.1: Systems Engineering Principles

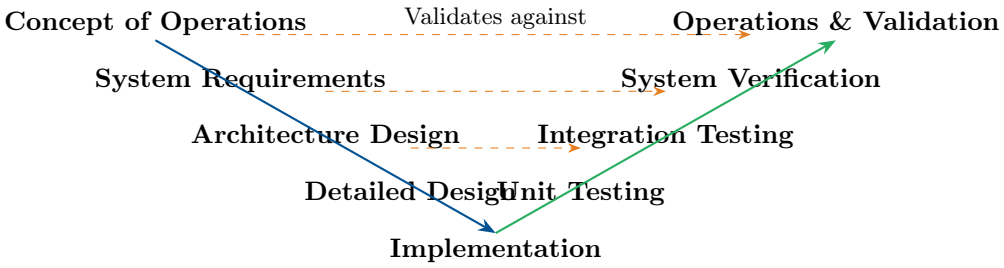
☞ 5 weeks ✎ *Systems Engineering Principles and Practice* by Alexander Kossiakoff

About This Book: A comprehensive textbook used in professional and academic settings to understand the core principles and practical application of systems engineering. It covers the full lifecycle from concept development through operations and retirement.

4.1.1 Week 1–2: The Systems Engineering Process

The Vee Model

The Vee Model provides a framework for understanding the systems engineering lifecycle:

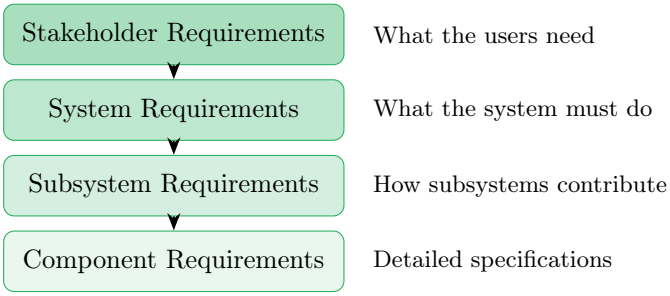


Key Processes

- **Requirements Engineering:** Eliciting, analyzing, specifying, and validating requirements
- **Functional Analysis:** Decomposing system functions into manageable elements
- **Design Synthesis:** Creating physical solutions that satisfy functional requirements
- **Verification and Validation:** Ensuring the system meets requirements and user needs

4.1.2 Week 3–4: Requirements and Architecture

Requirements Hierarchy



4.1.3 Week 5: Integration and Life Cycle Management

Key Topics

- System integration strategies (big bang vs. incremental)
- Test planning and execution
- Configuration management
- Operations, maintenance, and system retirement

★ Practical Exercise

Exercise B.1: Requirements Decomposition
For the ordering system architecture you designed in Exercise A.1, create a formal requirements specification:

1. Write 5 stakeholder requirements (user perspective)
2. Derive 15 system requirements from the stakeholder requirements
3. Create traceability matrix linking stakeholder to system requirements
4. Define verification methods for each system requirement
5. Identify potential requirement conflicts and propose resolutions

4.2 Module B.2: Practical Systems Engineering

☞ 3 weeks ✎ *The Art of Systems Engineering* by Marco Chicherio

About This Book: A practical guide focusing on the problem-solving and design aspects of systems engineering. This book emphasizes the craft and judgment required in real-world SE practice.

4.2.1 Week 6–8: Applied Systems Engineering

Key Focus Areas

- Problem structuring and stakeholder analysis
- Trade study methodologies and decision-making frameworks
- Interface management and integration planning
- Risk identification and mitigation strategies
- Technical reviews and decision gates

★ Practical Exercise

Exercise B.2: Trade Study
Conduct a formal trade study for the data storage layer of your ordering system:

1. Define evaluation criteria and weighting factors
2. Identify at least 3 alternative solutions
3. Score each alternative against the criteria
4. Document assumptions and sensitivity analysis
5. Present recommendations with supporting rationale

Chapter 5

Track C: Software Development Practice

★ Track C: Software Development

This track develops expertise in the craft of software development, covering object-oriented design principles, project management challenges, and modern engineering practices. It bridges the gap between architecture and implementation.

☞ 10 weeks ▲ Beginner to Intermediate

✓ Prerequisites

- Completion of Phase 1: Foundational Systems Thinking
- Programming experience in an object-oriented language
- Basic understanding of software development workflows

★ Learning Objectives

- Apply object-oriented design principles to create flexible, maintainable code
- Understand the human and organizational aspects of software projects
- Apply modern software engineering practices effectively
- Navigate the trade-offs between design purity and practical constraints
- Lead and contribute to software projects more effectively

5.1 Module C.1: Object-Oriented Design

☞ 4 weeks ✎ *Practical Object-Oriented Design* by Sandi Metz

About This Book: A highly relevant and practical book offering object-oriented design techniques that lead to code that is easy to understand, change, and extend. Metz focuses on managing dependencies and creating flexible designs.

5.1.1 Week 1–2: Designing Classes with Purpose

Single Responsibility and Managing Dependencies

Single Responsibility Principle: A class should have only one reason to change. This doesn't mean a class should do only one thing, but rather that everything it does should be highly related.

Dependency Management: The direction and nature of dependencies determines how easily code can change:

- Depend on things that change less frequently than you do
- Depend on abstractions, not concretions
- Inject dependencies rather than hard-coding them

★ Key Insight

Design is the art of arranging code to minimize the cost of change. Good design doesn't cost more—it costs less over the lifetime of the software because it reduces the effort required to add features and fix bugs.

5.1.2 Week 3–4: Flexible Interfaces and Inheritance

Key Concepts

Duck Typing: Focus on what objects do, not what they are. If it walks like a duck and quacks like a duck, treat it like a duck.

Composition over Inheritance: Prefer object composition to class inheritance. Inheritance creates tight coupling, while composition allows more flexible relationships.

Interface Design: Good interfaces are minimal, complete, and stable. They define what messages an object responds to without exposing implementation details.

★ Practical Exercise

Exercise C.1: Refactoring for Flexibility
Take an existing codebase (your own or an open-source project) and:

1. Identify a class with multiple responsibilities
2. Refactor it into multiple single-responsibility classes
3. Identify hard-coded dependencies and inject them
4. Create interfaces/protocols for the dependencies
5. Write tests that use test doubles for the dependencies

Document how these changes affect testability and changeability.

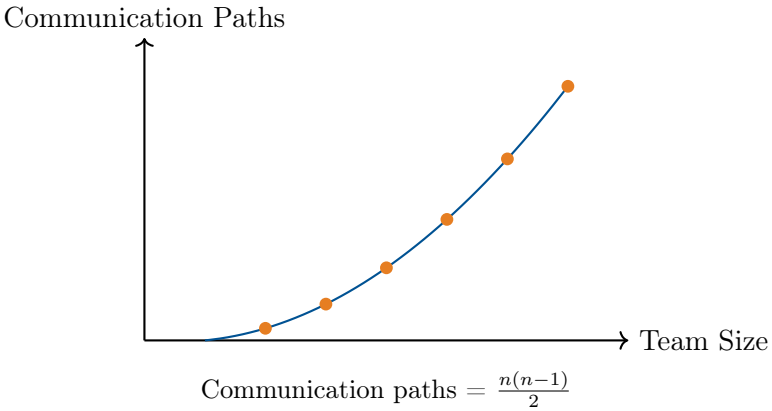
5.2 Module C.2: The Human Side of Software

☞ 3 weeks ✎ *The Mythical Man-Month* by Frederick Brooks Jr.

About This Book: A classic in software engineering that discusses project management fundamentals, including the famous Brooks's Law: "Adding manpower to a late software project makes it later." Despite being decades old, its insights remain remarkably relevant.

5.2.1 Week 5–6: Essential Concepts

Brooks's Law and Its Implications



The Surgical Team Model

Brooks proposed organizing teams around a "chief programmer" (like a surgical team around a surgeon), with specialized supporting roles:

- Chief Programmer: The surgeon who does the core work
- Copilot: A peer who can step in and provides second opinion
- Administrator: Handles money, people, space, machines
- Editor: Maintains documentation
- Toolsmith: Builds specialized tools for the chief

5.2.2 Week 7: The Second-System Effect and Other Insights

Key Concepts

Second-System Effect: The second system a designer builds is often over-engineered, as they try to include all the features they wished they had in the first.

Plan to Throw One Away: The first version of a system will be a learning experience. Plan for it.

Conceptual Integrity: A system should have a coherent vision. One mind (or a small group with shared vision) should be responsible for design.

★ Practical Exercise

Exercise C.2: Project Retrospective Analysis
Reflect on a project you've participated in:

1. Identify instances where Brooks's Law applied
2. Analyze how communication overhead affected productivity
3. Evaluate whether conceptual integrity was maintained
4. Identify second-system effect symptoms if present
5. Propose changes based on Brooks's insights

5.3 Module C.3: Modern Software Engineering

☞ 3 weeks ✎ *Modern Software Engineering* by David Farley

About This Book: A recent take on building better software faster, focusing on practical principles and practices. Farley emphasizes empirical approaches, fast feedback, and continuous improvement as foundations for effective software development.

5.3.1 Week 8–10: Engineering Excellence

Core Principles

Working Iteratively: Break work into small, valuable increments that can be completed and validated quickly.

Fast Feedback: Reduce the time between making a change and learning about its effects. This applies to testing, deployment, and user feedback.

Working Incrementally: Build systems in small steps, each of which adds value and can be validated independently.

Experimentation: Treat development as a series of experiments. Form hypotheses, test them, and learn from the results.

Modern Practices

- Continuous Integration and Continuous Delivery (CI/CD)
- Test-Driven Development (TDD) and Behavior-Driven Development (BDD)
- Trunk-based development and feature flags
- Infrastructure as Code and GitOps
- Observability and monitoring

★ Practical Exercise

Exercise C.3: CI/CD Pipeline Design
Design a complete CI/CD pipeline for the ordering system:

1. Define the stages from commit to production
2. Specify the automated tests at each stage
3. Design the deployment strategy (blue-green, canary, etc.)
4. Include observability and rollback mechanisms
5. Document how the pipeline enables fast feedback

Chapter 6

Track D: Business Systems and Process Design

★ Track D: Business Systems

This track develops expertise in building, documenting, and optimizing business processes and organizational systems. It provides the non-technical foundations that enable technical systems to create value within organizations.

☞ 10 weeks ▲ Beginner to Intermediate

✓ Prerequisites

- Completion of Phase 1: Foundational Systems Thinking
- Basic understanding of business operations
- Interest in organizational design and process improvement

★ Learning Objectives

- Understand why systems are crucial for business success
- Document and optimize business processes effectively
- Create organizational structures that enable scaling
- Apply service design methodologies to process improvement
- Integrate technical and business systems thinking

6.1 Module D.1: The Importance of Business Systems

☞ 2 weeks ✎ *The E-Myth Revisited* by Michael Gerber

About This Book: Excellent for understanding the importance of systems in small businesses, serving as a foundation before diving into specifics. Gerber explains why most small businesses fail and how systematization is the key to success.

6.1.1 Week 1–2: Working ON vs. IN the Business

The Three Roles

Gerber identifies three personas within every business owner:

- **The Technician:** Wants to do the work; lives in the present
- **The Manager:** Wants to organize and plan; lives in the past
- **The Entrepreneur:** Wants to create and innovate; lives in the future

The Franchise Prototype

The key insight is treating your business as if it were the prototype for a franchise:

★ Key Insight

A business that depends on you (the owner) for its operation is not a business—it's a job. The goal is to create systems that allow the business to run without your direct involvement, producing consistent results regardless of who executes them.

6.2 Module D.2: Process Documentation

☞ 3 weeks ✎ *Work the System* by Sam Carpenter

About This Book: Offers practical, actionable steps for documenting and improving business processes. Carpenter provides a methodology for identifying, documenting, and optimizing the systems that run your business.

6.2.1 Week 3–5: Systems Documentation Method

The Three Documents

Carpenter's methodology centers on three core documents:

1. **Strategic Objective:** A one- to two-page description of what you want your business to be
2. **Operating Principles:** The guiding principles for decision-making
3. **Working Procedures:** Detailed documentation of how each task is performed

Creating Working Procedures

Effective working procedures should be:

- Written by the person who does the work
- Specific enough that anyone could follow them
- Updated whenever the process changes
- Reviewed regularly for improvement opportunities

★ Practical Exercise

Exercise D.1: Process Documentation
Select a key process in your work (e.g., code review, incident response, deployment):

1. Document the current process step-by-step
2. Identify decision points and their criteria
3. Note where the process breaks down or varies
4. Create a standardized procedure document
5. Identify metrics to measure process effectiveness

6.3 Module D.3: Organizational Systems

☞ 2 weeks ✎ *Traction* by Gino Wickman

About This Book: Provides tools to create a vision and develop an organizational structure, which are essential for building any system. The Entrepreneurial Operating System (EOS) offers a comprehensive framework for running a business.

6.3.1 Week 6–7: The EOS Framework

Six Key Components

The Entrepreneurial Operating System addresses six components:

1. **Vision:** Getting everyone aligned on where you're going and how you'll get there
2. **People:** Having the right people in the right seats
3. **Data:** Running your business on objective information
4. **Issues:** Identifying and solving problems systematically
5. **Process:** Documenting and following core processes
6. **Traction:** Bringing discipline and accountability to the organization

6.4 Module D.4: Scaling Systems

☞ 1 week ✎ *Scaling Up* by Verne Harnish

About This Book: Focuses on how to scale a business, which involves building and refining systems to handle growth. Harnish provides practical tools for managing the four key areas: People, Strategy, Execution, and Cash.

6.4.1 Week 8: Growth Systems

The Four Decisions

Scaling a business requires mastering four key areas:

- **People:** Attracting, developing, and retaining the right team
- **Strategy:** Differentiating and aligning the organization
- **Execution:** Driving accountability and alignment
- **Cash:** Generating enough cash to fuel growth

6.5 Module D.5: Service Design

☞ 2 weeks ✎ *Service Design Doing*

About This Book: A technical and comprehensive textbook on designing and optimizing any process, from point A to point B. It uses diagrams to visualize flow and provides tools for creating exceptional service experiences.

6.5.1 Week 9–10: Design Thinking for Processes

Key Methodologies

- **Journey Mapping:** Visualizing the end-to-end experience
- **Service Blueprinting:** Documenting frontstage and backstage activities
- **Stakeholder Mapping:** Understanding all parties involved
- **Prototyping Services:** Testing service designs before implementation

★ Practical Exercise

Exercise D.2: Service Blueprint
Create a service blueprint for the ordering process in your system:

1. Map the customer journey from discovery to delivery
2. Document the frontstage interactions (what customer sees)
3. Document the backstage processes (what happens behind the scenes)
4. Identify support processes and systems
5. Mark pain points and opportunities for improvement

Chapter 7

Phase 5: Capstone Integration and Application

☞ 12 weeks ▲ Advanced

This final phase integrates learning from all tracks into a comprehensive capstone project. Participants synthesize their knowledge by designing, documenting, and potentially implementing a complete system that demonstrates mastery across all domains.

★ Learning Objectives

- Integrate systems thinking with technical architecture
- Apply software engineering best practices in a realistic context
- Create comprehensive documentation spanning all aspects of a system
- Demonstrate ability to make and justify architectural trade-offs
- Present complex technical concepts to diverse stakeholders

7.1 Capstone Project: Complete Ordering System

7.1.1 Project Scope

Design and document a complete ordering system that incorporates:

1. **Systems Thinking Analysis:** Causal loop diagrams, leverage point identification, and resilience analysis
2. **Software Architecture:** Complete architectural documentation including ADRs, component diagrams, and deployment architecture
3. **Distributed Systems Design:** Data partitioning strategy, consistency model, and fault tolerance approach
4. **Systems Engineering Artifacts:** Requirements specification, traceability matrix, and verification plan
5. **Software Development Plan:** Object model, coding standards, and CI/CD pipeline design
6. **Business Systems:** Process documentation, organizational structure, and service blueprint

7.1.2 Deliverables

Phase 1: Analysis (Weeks 1–3)

- Stakeholder analysis and requirements elicitation
- System context diagram and boundary definition
- Causal loop diagram of the complete system
- Initial architectural hypothesis

Phase 2: Design (Weeks 4–7)

- Complete software architecture documentation
- Data model and partitioning strategy
- Interface specifications
- Integration architecture

Phase 3: Implementation Planning (Weeks 8–10)

- Development team structure and responsibilities
- CI/CD pipeline specification
- Testing strategy
- Operations runbook

Phase 4: Presentation (Weeks 11–12)

- Executive summary for business stakeholders
- Technical deep-dive for engineering teams
- Risk assessment and mitigation plan
- Lessons learned and recommendations

7.2 Assessment Criteria

Criterion	Weight	Description
Systems Thinking Application	15%	Effective use of systems thinking concepts throughout
Architectural Quality	20%	Appropriate architecture style, clear trade-offs, fitness functions
Technical Depth	20%	Thorough treatment of distributed systems, data design
Documentation Quality	15%	Clear, complete, maintainable documentation

Criterion	Weight	Description
Process Integration	15%	Effective integration of business and technical processes
Presentation Quality	15%	Clear communication to varied audiences

Appendix A

Complete Reading Schedule

A.1 52-Week Master Schedule

Week	Phase/Track	Book/Focus	Hours/Week
1–4	Foundation	Thinking in Systems	10–12
5–8	Foundation	General Systems Thinking	10–12
9–12	Track A	Fundamentals of Software Architecture	12–15
13–18	Track A	Designing Data-Intensive Applications	12–15
19–22	Track A	Distributed Systems (selected chapters)	12–15
23–24	Track A	Operating System Concepts (selected)	10–12
25–29	Track B	Systems Engineering Principles	10–12
30–32	Track B	The Art of Systems Engineering	10–12
33–36	Track C	Practical Object-Oriented Design	12–15
37–39	Track C	The Mythical Man-Month	8–10
40–42	Track C	Modern Software Engineering	10–12
43–44	Track D	The E-Myth Revisited	8–10
45–47	Track D	Work the System	10–12
48	Track D	Traction (core chapters)	8–10
49	Track D	Scaling Up (selected chapters)	8–10
50	Track D	Service Design Doing (selected)	10–12
51–52	Capstone	Integration Project	15–20

Appendix B

Supplementary Resources

B.1 Online Resources

- Martin Fowler's Architecture Guide: https://martinfowler.com/architecture/
- The Systems Thinker: https://thesystemsthinker.com/
- InfoQ Software Architecture: https://www.infoq.com/software-architecture/
- INCOSE Systems Engineering Handbook (reference)

B.2 Related Academic Courses

- MIT OpenCourseWare: Introduction to Systems Engineering
- Stanford Online: Distributed Systems
- Carnegie Mellon Software Engineering Institute resources

Appendix C

Glossary of Key Terms

ADR Architecture Decision Record—document capturing important architectural decisions

CAP Consistency, Availability, Partition tolerance theorem

CI/CD Continuous Integration/Continuous Delivery

DDIA Designing Data-Intensive Applications (Kleppmann)

EOS Entrepreneurial Operating System (Wickman)

OO Object-Oriented (design/programming)

SE Systems Engineering

SRP Single Responsibility Principle

TDD Test-Driven Development

End of Curriculum

Version 1.0 — Comprehensive Systems Engineering and Software Architecture Program