

Contents

Preface

1 Foundations of Software Development

- 1.1 Taxonomy of software
- 1.2 Software Development Context: Internal vs. External Focus
 - 1.2.1 Building For a Business (Internal/Operational Software)
 - 1.2.2 Building As the Business (Core Product)
- 1.3 Overview of the Software Development Lifecycle (SDLC)
 - 1.3.1 Phases of the SDLC
 - 1.3.2 Common SDLC Models
 - 1.3.3 Choosing the Right Model
 - 1.3.4 SDLC Documentation
 - Software Requirements Specification (SRS)
 - Design Documents (HLD, LLD)
 - Test Plans and Test Cases
 - User Manuals
 - API Documentation
 - 1.3.5 Challenges in SDLC Implementation
- 1.4 Summary

2 Basics of UX/UE

- 2.1 Usability
- 2.2 Consistency
- 2.3 Responsiveness and Performance
- 2.4 Feedback and Affordances
- 2.5 Accessibility
- 2.6 Contextual Design
- 2.7 Simplicity (KISS: Keep It Simple, Stupid)
- 2.8 Error Handling and Prevention
- 2.9 User Control and Flexibility
- 2.10 Aesthetic and Minimalist Design

- 2.11 Task Efficiency and Workflow Optimization
- 2.12 Onboarding and Learnability
- 2.13 Mobile Responsiveness
- 2.14 Emotional Design and Engagement
- 2.15 Summary

3 Fundamentals of Software Design

- 3.1 Driver for Good Software Design
- 3.2 Core Software Design Principles
 - 3.2.1 Single Responsibility Principle (SRP)
 - 3.2.2 Open/Closed Principle (OCP)
 - 3.2.3 Liskov Substitution Principle (LSP)
 - 3.2.4 Interface Segregation Principle (ISP)
 - 3.2.5 Dependency Inversion Principle (DIP)
 - 3.2.6 You Aren't Gonna Need It (YAGNI)
 - 3.2.7 Encapsulation
 - 3.2.8 Don't Repeat Yourself (DRY)
- 3.3 Introduction to Design Patterns
 - 3.3.1 Singleton
 - 3.3.2 Factory Method
 - 3.3.3 Observer
 - 3.3.4 Strategy
 - 3.3.5 Command
 - 3.3.6 Adapter
 - 3.3.7 Visitor
 - 3.3.8 Proxy
- 3.4 Summary

4 Architectural Design

- 4.1 Overview of Major Architectural Styles
 - 4.1.1 Layered (N-Tier) Architecture
 - 4.1.2 Client-Server Architecture
 - 4.1.3 Microservices Architecture
 - 4.1.4 Event-Driven Architecture
 - 4.1.5 Service-Oriented Architecture (SOA)
 - 4.1.6 Monolithic Architecture
 - 4.1.7 Component-Based Architecture
 - 4.1.8 Peer-to-Peer (P2P) Architecture
 - 4.1.9 Repository Architecture (Blackboard Architecture)
 - 4.1.10 Pipeline (or Data Flow) Architecture

- 4.2 Selected Architectural Styles in Depth
 - 4.2.1 Multi-Tier (N-Tier) Architecture
 - Salient Characteristics of Multi-Tier Architecture
 - Quality Attributes Achieved by Multi-Tier Architecture:
 - Real-World Example Scenarios with Multi-Tier Architecture
 - 4.2.2 Microservices Architecture
 - Salient Characteristics of Microservices Architecture
 - Quality Attributes Achieved by Microservices Architecture
 - Real-World Example Scenarios with Microservices Architecture
 - 4.2.3 Serverless Architecture
 - Salient Characteristics of Serverless Architecture
 - Quality Attributes Achieved by Serverless Architecture
 - Real-World Example Scenarios of Serverless Architecture
 - 4.2.4 Event Driven Architecture
 - Salient Characteristics of Event-Driven Architecture
 - Quality Attributes Achieved by Event-Driven Architecture
 - Real-World Example Scenarios with Event-Driven Architecture
 - 4.2.5 Peer-to-Peer (P2P) Architecture
 - Salient Characteristics of Peer-to-Peer Architecture
 - Quality Attributes Achieved by Peer-to-Peer Architecture
 - Real-World Example Scenarios with Peer-to-Peer Architecture
- 4.3 Choosing the Right Architecture for Your Application
 - 4.3.1 Factors Influencing Architectural Decisions
 - 4.3.2 Trade-offs and Considerations
- 4.4 Design and Architecture Issues Cutting Across Application Types
 - 4.4.1 Data access abstraction
 - 4.4.2 Authentication and authorization
 - 4.4.3 Securing data in transit and at rest
 - 4.4.4 Input validation and sanitization
 - 4.4.5 Logging and auditing
 - 4.4.6 Instrumentation and metrics
 - 4.4.7 Concurrency and parallelism
 - 4.4.8 Error handling
- 4.5 API Design and Integration
 - 4.5.1 RESTful APIs (REpresentational State Transfer)
 - 4.5.2 SOAP APIs (Simple Object Access Protocol)

- 4.5.3 GraphQL APIs
 - 4.5.4 WebSocket APIs
 - 4.5.5 RPC APIs (Remote Procedure Call)
 - 4.5.6 Library or SDK APIs
 - 4.5.7 File-Based APIs
 - 4.5.8 Custom Protocols
 - 4.5.9 Serverless APIs
- 4.6 How to choose an API style?
- 4.7 Summary
- 5 Architecting software in the era of AI/ML**
 - 5.1 Key Opportunities in AI/ML Software Architecture
 - 5.2 Important Issues and Threats in AI/ML Software Architecture
 - 5.2.1 Ethical and Bias Issues
 - 5.2.2 Model Drift and Data Quality
 - 5.2.3 Data Privacy and Security
 - 5.2.4 Model Interpretability and Explainability
 - 5.2.5 Infrastructure and Scalability
 - 5.3 How is architecting software different given AI/ML in the mix today?
 - 5.3.1 Data as the Primary Asset
 - 5.3.2 Model Lifecycle Management
 - 5.3.3 Real-Time Processing and Decision-Making
 - 5.3.4 Decoupling AI/ML Components
 - 5.3.5 Infrastructure for AI/ML
 - 5.3.6 Explainability and Transparency
 - 5.3.7 Ethical AI and Bias Management
 - 5.3.8 Security and Data Privacy
 - 5.3.9 Governance and Compliance
 - 5.4 Summary
- 6 LLM-powered and Agentic Applications**
 - 6.1 LLM-powered applications
 - 6.2 Should you build an Agentic application or plain LLM-powered one?
 - 6.3 Architecture of an agentic application
 - 6.3.1 Core components and interactions
 - User Interface (UI) Layer
 - Orchestration Layer
 - Data Layer
 - LLM Core Layer
 - Safety and Monitoring Layer

- 6.4 Architectural Foundations of LLM-Powered Software
 - 6.4.1 The Agentic Architecture Pillars
 - 6.4.2 Architectural Patterns: Making Trade-offs Explicit
 - Pattern 1: Agent as a First-Class Service
 - Pattern 2: Orchestrator–Worker with Explicit Fallbacks
 - Pattern 3: Context as a First-Class Data Type
 - Pattern 4: Deterministic Boundaries for Non-Deterministic Systems
 - Pattern 5: Observability and Replayability
 - Pattern 6: Schema-Governed Contracts
- 6.5 Summary
- 7 Database Design and Management**
 - 7.1 Core Concepts
 - 7.1.1 Data Modeling: Sketching Before Building
 - 7.1.2 Normalization: Reducing Redundancy and Improving Integrity
 - 7.1.3 Relationships: How Tables Connect
 - 7.2 Real-World Example: A Simple Online Bookstore
 - 7.3 Adding Behavior: Triggers and Stored Procedures/Functions
 - 7.3.1 Triggers: Automated Actions
 - 7.3.2 Stored Procedures and Functions: Reusable Database Code
 - 7.4 Beyond Relational: A Glimpse into NoSQL
 - 7.5 Summary
- 8 Software Testing and Quality Assurance**
 - 8.1 Difference Between Testing and Quality Assurance
 - 8.2 Goals and Objectives of Software Testing
 - 8.3 Software Development Life Cycle (SDLC) and Testing
 - 8.3.1 Shift-Left Testing
 - 8.4 Types and Levels of Testing
 - 8.4.1 Unit Testing
 - 8.4.2 Integration Testing
 - 8.4.3 System Testing
 - 8.4.4 Specialized Testing Types
 - 8.5 Testing Techniques and Methodologies
 - 8.5.1 Black Box Testing
 - 8.5.2 White Box Testing
 - 8.5.3 Gray Box Testing
 - 8.5.4 Static vs. Dynamic Testing

- 8.6 Test Planning and Documentation
 - 8.6.1 Test Plans
 - 8.6.2 Test Cases and Test Scenarios
 - 8.7 Test Scripts
 - 8.7.1 Traceability Matrix
 - 8.8 Test Automation
 - 8.8.1 Automation Tools and Frameworks
 - Bug Tracking Tools
 - Performance Testing Tools
 - Code Analysis Tools
 - Mobile Testing Tools
 - 8.8.2 Continuous Testing in CI/CD Pipelines
 - 8.9 Security Testing
 - 8.9.1 Core Security Concepts
 - 8.9.2 Security Testing Techniques
 - Tools and Resources
 - 8.10 Performance Testing
 - 8.10.1 Simulating Expected User Load
 - 8.10.2 Key Metrics to Monitor During Load Testing
 - 8.10.3 Stress Testing
 - 8.10.4 Soak Testing
 - 8.11 Compliance and Regulatory Testing
 - 8.11.1 Industry-Specific Regulations: Finance, Healthcare, etc.
 - 8.11.2 Data Protection Compliance: GDPR, HIPAA Requirements
 - 8.11.3 Accessibility Testing: Compliance with WCAG Standards
 - 8.12 Future Trends in Testing
 - 8.12.1 AI in Testing: Intelligent Automation and Prediction
 - 8.12.2 Shift-Right Testing: Extending the Testing Lifecycle
 - 8.12.3 Testing for Emerging Technologies
 - 8.13 Summary
- 9 Deployment and Maintenance**
- 9.1 Automating the Release Pipeline – CI/CD
 - Core Concepts: What Is CI and CD?
 - 9.1.1 Building a CI/CD Pipeline: A Step-by-Step Guide
 - Version Control Integration
 - Automated Builds
 - Automated Testing
 - Artifact Repository
 - Deployment Automation

- 9.1.2 Tools and Technologies: Choosing the Right CI/CD Platform
 - 9.1.3 Best Practices for Effective CI/CD
 - 9.1.4 Example Scenario: CI/CD Pipeline for a Web Application
- 9.2 Infrastructure as Code (IaC)
 - 9.2.1 IaC Core Concept
 - 9.2.2 The Benefits of Adopting IaC
 - 9.2.3 Declarative vs. Imperative Approaches
 - 9.2.4 Popular IaC Tools
 - 9.2.5 Key Practices for Effective IaC
- 9.3 Containerization – Packaging and Orchestrating Applications
 - 9.3.1 Understanding Docker
 - 9.3.2 Introduction to Kubernetes
- 9.4 Observability – Understanding Application Behavior in Production
 - 9.4.1 The Three Pillars of Observability
 - 9.4.2 Key Metrics to Monitor
 - 9.4.3 Log Aggregation and Analysis
 - 9.4.4 Distributed Tracing
 - 9.4.5 Alerting and Notification
 - 9.4.6 Example Scenario: A Monitoring using Prometheus and Grafana
- 9.5 Version Control (Git) – Mastering Collaboration and Code Management
 - 9.5.1 Core Git Concepts
 - 9.5.2 Branching Strategies
 - 9.5.3 Advanced Git Techniques
 - 9.5.4 Example Scenario: Collaborating on a Project using Git and GitHub
- 9.6 Deployment Strategies – Choosing a Product Release Approach
 - 9.6.1 Big Bang Deployment
 - 9.6.2 Rolling Deployment
 - 9.6.3 Blue-Green Deployment
 - 9.6.4 Canary Deployment
 - 9.6.5 Feature Flags
 - 9.6.6 Choosing the Right Strategy
- 9.7 Configuration Management – Managing Application Settings Across Environments
 - 9.7.1 Environment Variables
 - 9.7.2 Configuration Files
 - 9.7.3 Secrets Management

- 9.7.4 Best Practices
- 9.8 Automating Database Changes: Database Migrations
 - 9.8.1 The Need for Database Migrations
 - 9.8.2 Migration Tools
 - 9.8.3 Writing Migration Scripts
- 9.9 Summary
- 10 Bringing it All Together**
 - 10.1 High-Level Description Of The Problem
 - 10.2 Development of Web Applications
 - 10.2.1 What Type of Web Application to Build
 - 10.2.2 Choice of Frameworks and Libraries
 - 10.2.3 Monolithic vs. Microservices Architecture
 - 10.2.4 Database Design and Structure
 - 10.2.5 State Management: Stateful vs. Stateless Architecture
 - 10.2.6 Front-End and Back-End Communication
 - 10.2.7 Security Considerations
 - 10.2.8 Session Management
 - 10.2.9 Caching Strategies
 - 10.2.10 Deployment Strategy
 - 10.3 Architecture Of The Energy Billing Solution
 - Rationale for the Chosen Option
 - Theory of Operation – Key Flows
 - API design (representative endpoints)
 - UI Summary (Roles and Main Screens)
 - Security and Privacy
 - Scaling and Operations
 - Observability and Monitoring
 - Testing Strategy
 - 10.4 Rationale for Design Decisions