- Codebook
  - 1 Traits of Experts
    - 1.1 Writing Code Vs. Writing Industrial Code
    - 1.2 Are Able to Understand How Either Developer Thought
    - 1.3 Better Design, Modularization
    - 1.4 Experts Are Faster, Due to Experience
      - 1.4.1 Know Common Patterns, Techniques, Structures
      - 1.4.2 Know How to Approach the Code
      - 1.4.3 Know Which Questions to Ask
      - 1.4.4 Navigate the Problem Faster, Without Help
    - 1.5 Experience Helps With Program Comprehension
    - 1.6 Experts Might Stick to Old Tools
    - 1.7 Experts Might Be Misled by Strong Assumption
    - 1.8 Might Have to Fallback to Same Techniques As Beginners
    - 1.9 Having an Understanding of an Unexcepting Structure
    - 1.10 Knowing the Right Tools
  - 2 What Should/Be Taught (What University Can Do)
    - 2.1 Basics Need to Be Taught First
    - 2.2 Software Maintenance
      - 2.2.1 Tools for Software Maintenance
      - 2.2.2 Skills for Software Maintenance, Practical Experience
      - 2.2.3 Maintaining Software, Not Just Writing From Scratch
    - 2.3 Problem-solving, Analytical Thinking
    - 2.4 Practical Projects, Cases
    - 2.6.1 Industry Placement
    - 2.5 Integrated, Larger Projects
    - 2.6 Working in a Team
    - 2.7 Workflow Management, Issue Tracking
      - 2.7.1 Commit Messages
      - 2.7.2 Traceability, E.G., Requirements Implementation
    - 2.8 Containerization, Docker
    - 2.9 Software Architecture, Design Principles
      - 2.9.1 More Focus on Higher-level (In Contrast to Coding)
      - 2.9.2 Design For Scalability, For Real Deployments
    - 2.10 Importance of Documentation, Other-than-code Artefacts
    - 2.11 Choice/Number of Programming Languages
    - 2.12 Experience Cannot Be Taught at University
    - 2.13 Not Taking Things for Granted
    - 2.14 Visualization
  - 3 Support by Company (What Company Can Do)
    - 3.1 Mentoring
    - 3.2 Training and Onboarding
  - 4 Code Cognition Models
    - 4.1 Control Flow
    - 4.2 Data Flow
  - 5 Strategies
    - 5.1 Understand Heritage
    - 5.2 Top-down
    - 5.3 Bottom-up
    - 5.4 Understand Overall Structure
      - 5.4.1 Software Architecture
      - 5.4.2 Blueprint, Understanding the Big Picture
      - 5.4.3 Understand Overall Architecture, Design Principles
      - 5.4.4 Understand Overall Organization, to Enable Navigation
    - 5.5 Understand Dependencies, Interaction Among Components
    - 5.6 Understand Context
      - 5.6.1 Understand Starting Point, Significance of Maintenance Task
      - 5.6.2 Understanding a Higher-level Goal
      - 5.6.3 Business Context, Business Domain
      - 5.6.4 Understand Requirements
    - 5.7 Understanding Why Something Works in a Certain Way
    - 5.8 Construct a Mental Model
    - 5.9 Reverse Engineer Docs, Specs, Architecture, Models
    - 5.10 Have Good Knowledge of Tools
    - 5.11 Have Good Knowledge of Technologies ("The Stack", Platform)
    - 5.12 Understand Starting Point
      - 5.12.1 Identifying Some Good to Start from (in Source Code)
      - 5.12.2 Understanding Interaction (in Code, Or Elsewhere)
    - 5.13 Comprehension Strategies
      - 5.13.1 Identify Starting Point for Comprehension, Filter
      - 5.13.2 Zoom in/Avoid Comprehension and Minimize Effort
      - 5.13.3 Debug Program to Elicit Runtime Information
      - 5.13.4 Follow Dependencies/Follow Work Pattern
      - 5.13.5 Employ Structural Comprehension Strategy
      - 5.13.6 Call Graph, Follow Method Calls
      - 5.13.7 Test Expected Behavior
        - 5.13.7.1 Check Effects of UI Interaction on Back-end
        - 5.13.7.2 Small Change First, Before Big Change
        - 5.13.7.3 Interact With UI to Test Expected Behaviour
      - 5.13.8 Take Notes to Reflect Mental Model and Record Knowledge
    - 5.14 Graphical Overviews, Flowcharts
    - 5.15 Documentation As a Last Resort
    - 5.16 Switching Between Pieces of Information Once Clear
    - 5.17 Goal of Modifying the Code According to Task at Hand
  - 6 Information Sources
    - 6.1 Similar Code, in Repositories, Looking For Common Patterns
    - 6.1.1 Projects in Software Repositories (Github)
    - 6.2 Issue Tracker
      - 6.2.1 History of the Project in the Issue Tracker
    - 6.3 Github, Commit Messages
    - 6.4 Stack Overflow, Forums, Blogs
    - 6.5 Software Architecture
      - 6.5.1 Draw the Architecture
    - 6.6 Established Procedure/s in the Company
    - 6.7 Existing Repository in Version Control Software
    - 6.8 Tacit Knowledge, Based on Experience
    - 6.9 People
      - 6.9.1 Get Into the Mind of the Original Developer
      - 6.9.2 Talk to Developers
        - 6.9.2.1 Knowledge in People's Heads
        - 6.9.2.2 Experts, People With the Right Knowledge
        - 6.9.2.3 Developers That Have Worked on the Code Previously
        - 6.9.2.4 Communication Preferred Over Documentation
      - 6.9.3 Talk to Users
    - 6.10 Test Cases
    - 6.11 Exploration of the Code (Ad Onward)
      - 6.11.1 Source Code More Trusted Than Documentation
      - 6.11.2 Access to the Code, Look at the Code
      - 6.11.3 Self-learning
      - 6.11.4 Experimenting With Code
    - 6.12 Documentation
      - 6.12.1 Search in Documentation
    - 6.13 Inline Comments
    - 6.14 Information About Task at Hand, E.G. Bug Report
    - 6.15 Ide Tools, Search
      - 6.15.1 More Classic Ide Than Visual Tools
      - 6.15.2 Grep, Search Across All Files
    - 6.16 Dedicated Comprehension Tools Are Rarely Used
    - 6.17 Standards Facilitate Comprehension
    - 6.18 Courses, Learning Management System
    - 6.19 Google or Other Search Engines
    - 6.20 (Structure of the) Programming Language
    - 6.21 Tutorials, Blogs
  - 7 Technical Strategies
    - 7.1 Web Applications
    - 7.2 Architecture Implementation Consistency
    - 7.3 Modularization, Microservices
    - 7.4 Partition the Problem For Teamwork
    - 7.5 Documentation
      - 7.5.1 Documentation Transd to Source Code
    - 7.6 Pipelines, Connecting Tools
    - 7.7 Automation
    - 7.8 Deployment
      - 7.8.1 Continuous Integration, Continuous Delivery
      - 7.8.2 Frequent Releases, Fast Releases
    - 7.9 Testing
      - 7.9.1 Manual Testing
      - 7.9.2 Understanding Rationale Behind Tests
      - 7.9.3 Talk Early Testing
      - 7.9.4 Regression Tests
  - 8 Technical Skills
    - 8.1 Maintain Documentation
    - 8.2 Mastering Complex Frameworks
    - 8.3 Software Version Control, Repositories, Git, Github
    - 8.4 Model-based Approaches, Uml
    - 8.5 Code Review, Linting, Coding Standards
    - 8.6 Programming Languages
    - 8.7 Memory Management
    - 8.8 Troubleshooting, Making Things Work
    - 8.9 Structure/Design Your Code First, Then Code
    - 8.10 Security
    - 8.11 Error Handling
  - 9 Kinds of Approaches
    - 9.1 Reverse Engineering
    - 9.2 Reengineering
    - 9.3 Restructuring
    - 9.4 Refactoring
  - 10 Dimensions of Change
    - 10.1 Time of Change
    - 10.2 Change Type
    - 10.3 Change History
    - 10.4 Degree of Automation
    - 10.5 Activeness
    - 10.6 Change Frequency
    - 10.7 Anticipation
    - 10.8 Artifact
    - 10.9 Granularity, Scope
    - 10.10 Impact
    - 10.11 Change Propagation
    - 10.12 Availability
    - 10.13 Openness
    - 10.14 Safety and Security
    - 10.15 Degree of Formality
  - 11 Techniques to Support Program Comprehension
    - 11.1 Visualization
    - 11.2 Slicing
    - 11.3 Filtering
    - 11.4 Metric, Control Source Code Analysis
    - 11.5 Data Analysis
    - 11.6 Feature Location
    - 11.7 Feature Extraction
    - 11.8 Compression, Summarization, Clustering
    - 11.9 Recursion
    - 11.10 FCA
    - 11.11 Querying
    - 11.12 Order Analysis
    - 11.13 Multiple Traces
  - 12 Maintenance Tasks
    - 12.1 Adaptive Maintenance
    - 12.2 Perfective Maintenance
    - 12.3 Corrective Maintenance
    - 12.4 Reuse
    - 12.5 Code Leverage
  - 13 Maintenance Activities
    - 13.1 Understand System
    - 13.2 Refine Requirements
    - 13.3 Develop Preliminary Design
    - 13.4 Design/Add Code
    - 13.5 Debug
    - 13.6 Regression Tests
    - 13.7 Generate and Test Hypotheses
    - 13.8 Repair Code
    - 13.9 Reuse
      - 13.9.1 Integrate Components
      - 13.9.2 Locate Components
    - 13.10 Leverage Code
      - 13.10.1 Integrate Identified Components
      - 13.10.2 Obtain and Modify Components
      - 13.10.3 Reconfigure Existing to Use Components
      - 13.10.4 Find Solution Based on Components