EECS 22L: Software Engineering Project in C Language
Lecture 2

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 2: Overview

- Software Development Process
  - Software architecture design

- Project 1 Introduction
  - The game of chess

- Application Specification
  - Customer requests, goals, requirements
  - Discussion of features, options, considerations

- Technical Advise
  - Suggestions for data structure organization
  - Essential objects and operations
  - Algorithm and control flow
  - Components and task partitioning

Software Development Process

- EECS 22L Software Development Process
  1. Application specification
     - User's perspective (aka. client, customer, consumer)
     - Documentation
  2. Software architecture design and specification
     - Developer's perspective (aka. producer)
     - Software layers and modules
     - Documentation
  3. Implementation, testing, and debugging
     - Unit testing
     - System testing
  4. Software release
     - Binary program and documentation
     - Source code and documentation

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   3

Software Development Process

2. Software Architecture Design and Specification
   - Goal: Specify the developer's perspective!
     - What data structures are used? What algorithms?
     - What modules is the program composed of? Dependencies?
     - How do the modules interact? What functions and parameters?
   - Deliverable: Software Architecture Document
     - ➢ *Detailed description of the software components and structures!*
     - Data structures and algorithms
       - How is data organized?
       - How is data processed?
     - Software layers and modules
       - Software architecture with layers of modules and libraries
       - Application Programming Interface (API) of modules (header files!)
     - Implementation plan
       - Project timeline
       - Tasks and team member responsibilities

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   4

Software Architecture Document

- Contents of a Software Architecture Document (1/2)
    - Title page
        - Software title, version
        - Author/producer, affiliation
    - Front matter
        - Table of contents
        - Glossary
    - Software Architecture Overview
        - Introduction, goals, features
        - Major software components (e.g. module hierarchy), diagrams
        - Major interfaces (e.g. application procedural interfaces), diagrams
    - Installation
        - System requirements, compatibility
        - Setup and configuration
        - Building, compilation, and installation

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   5

Software Architecture Document

- Contents of a Software Architecture Document (2/2)
    - Various views on the software architecture
        - Use-case view, logical view, process view, deployment view (typically described in Unified Modeling Language, UML)
    - Documentation of packages, modules, interfaces
        - Detailed description of data structures
        - Detailed description of functions and parameters
        - Detailed description of data input and output (incl. format)
    - Development plan and timeline
        - Partitioning of tasks
        - Timeline of development, testing, releases
    - Back matter
        - Copyright, contact information
        - Legal, license, disclaimer of warranty
        - Index, References, Appendix

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   6

Software Architecture Document

- Example: Diagram of Software Layers and Modules
  - Stack of major components in the HW/SW architecture
    - Application modules
    - OS and third-party libraries
    - Operating system (OS) infrastructure
    - Hardware platform

	Chess (main module)	
	Strategy (AI)	User Interface (GUI)
Software	Rules (chess)	X11 Graphics Library (libSDL)
	Math (libm)	Std. C (libc)	
	Linux OS (RHEL-6-x86_64)	
Hardware	PC Hardware (x86_64 server)	

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   7

Software Architecture Document

- Example: Documentation of Chess Strategy Module
  - Module dependencies
    - Provides: Evaluation of potential moves
    - Requires: **libChessRules.a, libc.a**
  - Exported functions

    ```
    t_Move *SelectBestMove(
               t_MoveList *LegalMoves,
               t_Board    *Board,
               t_Player   Color)
    ```

    - Arguments:
      - **LegalMoves** list of potential moves (which must be legal)
      - **Board**   current board position
      - **Color**   player to make the next move
    - Result:
      - **BestMove**   pointer to the "smartest" move in the **LegalMoves** list
    - Notes:
      - Returns **NULL** if list of **LegalMoves** is empty
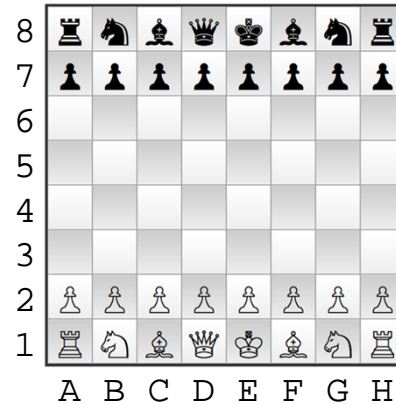
Chess (main)
Strategy (AI)
Rules (chess)

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   8

Project 1 Introduction

- The Game of Chess
  - Board
    - Initial positions
  - Pieces
    - Queen
    - Rook
    - Bishop
    - Knight
    - King
    - Pawn
  - Moves
    - Capture
    - Check
    - Checkmate

Application Specification

- Chess Game: Program Specification (1/3)
  - Basic functions: (Customer requests, requirements)
    1. Official rules of chess
    2. Interactive user interface (player sees board, makes moves)
    3. Interactive player (human user) vs. automatic player (computer)
    4. User chooses the side to play, white or black
    5. Human readable log file
    6. Computer moves in reasonable time (less than 1 minute)

  - ➢ Tournament support!

Application Specification

- Chess Game: Program Specification (2/3)
  - Advanced options: (Customer goals, extra features)
    1. Human vs. human, computer vs. computer
    2. Withdraw previous moves (undo)
    3. Different computer levels: beginner, intermediate, expert
    4. Hints on possible moves
    5. Graphical user interface (GUI)
    6. Chess clocks (timers)
    7. Interactive board setup
    8. Support for official algebraic notation
    9. ...
  - Considerations
    - ➢ Illegal move ends the game. The player loses.
    - ➢ Implement legal moves correctly!
    - ➢ Good strategy is more important than fancy features.

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   11

Application Specification

- Chess Game: Program Specification (3/3)
  - Challenge bonus: (Customer's idea, special bonus)
    - ➢ Solve the weekly LA Times Chess Puzzle

Jan. 4, 2015

Position No. 4345: White mates in two.

Position No. 4344: 1.Qg7! Hint: White mates next move with: Rf3, Qa7, Qd4, Qg1, Qxg3, Qxg1, or Qxa7.

  - Considerations
    - ➢ Advanced options 7 and 8 needed
    - ➢ Multi-move search strategy needed…

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   12

Technical Advise

- Data Structure Organization
    - ➢ *The importance of a well-defined data structure cannot be overestimated!*
        - Use object-oriented approach (even in plain ANSI-C)
        - Decompose into modules (manage complexity)
        - Use proper terms (ensure good code readability)
        - Consider efficiency
            - Execution speed
            - Memory size
    - – Good questions to ask yourself
        - What are the objects at hand? (Classes)
        - What operations are needed for the objects? (Methods)
        - What is the overall algorithm?
        - What components does the system consist of?

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   13

Technical Advise

- Suggestions for Data Structure Organization (1/4)
    - ➢ **Essential Objects** for the Chess Application
        - a player (who is either black or white)
        - a piece type (either king, queen, bishop, knight, rook, or pawn)
        - a piece (a combination of player/color and piece type)
        - a board (8x8 matrix of squares, with or without a piece on them)
        - a position of a piece (known to the user as "e2", for example)
        - a move (a combination of a start and end position, e.g. "e2 e4")
        - a log (a list of moves)

    - ➢ How can these basic objects be represented best?
        - What ANSI C primitives can be used?
        - What data structures need to be built?

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   14

Technical Advise

- Suggestions for Data Structure Organization (2/4)
  - ➤ **Essential Operations** for the Chess Application
    - on a board, lookup a piece at a given position
    - on a board, put a given piece onto a given position
    - on a board, move a piece from a position to another
    - for a piece on the board, compute all reachable positions
    - for a piece on the board, compute all legal moves
    - on a board, check whether or not a player's king is in check
    - on a board, check whether or not a player's king is in checkmate
    - for a player and a given board, compute all legal moves
    - from a list of moves, select the best one
    - …
  - ➤ How can these functions be represented best?
    - What function signatures are needed?

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   15

Technical Advise

- Suggestions for Data Structure Organization (3/4)
  - ➤ **Essential Algorithm** for the Chess Application
    - Overall control flow (main loop)
      - – setup
      - – display the board
      - – repeat
        - » white player makes a move
        - » display the board
        - » if black is in checkmate, white wins!
        - » black player makes a move
        - » display the board
        - » if white is in checkmate, black wins!
  - ➤ How can the computer make a smart move?
    - First, calculate all legally possible moves
    - Then, pick the best one!

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   16

Technical Advise

- Suggestions for Data Structure Organization (4/4)
    - ➢ **Essential Components** for the Chess Application
        - main program
        - user interface (textual and/or graphics)
        - chess objects (data structures for pieces, boards, moves)
        - chess rules (possible moves, legal moves)
        - lists (or trees) of moves
        - strategy (artificial intelligence, AI) module
        - log file module
        - feature modules
        - documentation and testing
    - ➢ How can these tasks / modules be partitioned best?
        - What dependencies exist? What can be done in parallel?
        - What is best done by everyone?

EECS22L: Software Engineering Project in C, Lecture 2   (c) 2015 R. Doemer   17