SENG 11232 · Engineering Foundation

STUDENT PRE-LECTURE NOTES · LECTURE 02

Computational Thinking & Problem Decomposition

Module 1 — Thinking Like an Engineer

HOW TO USE Read these notes BEFORE the lecture. Highlight terms you don't recognise. Bring your questions to class. Use the blank spaces for your own notes.

Learning Objectives

By the end of this lecture, you should be able to:

- 1. Define computational thinking and explain its four pillars
- 2. Apply decomposition to break complex problems into manageable sub-problems
- 3. Use abstraction to model essential features of a problem
- 4. Construct and trace a simple algorithm

The Four Pillars

Decomposition

Breaking a complex problem into smaller, independent sub-problems that can each be solved separately.

- Top-down: start with the whole; break into major components; break further
- Bottom-up: identify atomic building blocks; compose into larger systems

KEY Good decomposition: low coupling (sub-problems mostly independent) + high cohesion (each part does one thing).

Abstraction

Hiding irrelevant details; exposing only what matters for the current level of analysis.

- Levels of abstraction: transistors → gates → machine code → assembly → high-level language → API → application

Pattern Recognition

Identifying similarities between a new problem and problems already solved. Enables solution reuse.

Algorithmic Thinking

Designing a finite, unambiguous, step-by-step solution that can be executed repeatedly.

Your notes:

Algorithms & Problem-Solving Framework

Properties of a Good Algorithm (Knuth)

- Finiteness: terminates in a finite number of steps
- Definiteness: each step is precisely defined — no ambiguity
- Input/Output: takes defined inputs, produces defined outputs
- Effectiveness: each step is executable by a person with pencil and paper

6-Step Problem-Solving Framework

- 1. UNDERSTAND: identify inputs, outputs, constraints, edge cases
- 2. EXPLORE: work through small examples by hand
- 3. PLAN: choose a strategy
- 4. IMPLEMENT: write clean code following the plan
- 5. TEST: verify all cases including edge cases
- 6. REVIEW: can it be simpler? More efficient?

NOTE	Never skip steps 1 and 2. Most bugs are caused by misunderstanding the problem, not coding errors.

Your notes:

____

____

____

Key Terms for This Lecture

Term	Your Definition (fill in during/after lecture)
Decomposition	
Abstraction	
Algorithm	
Pseudocode	
Pattern Recognition	
Big-O notation	
Precondition	
Postcondition	

Self-Check Questions

Answer these after the lecture to test your understanding:

- Q1. Define the four pillars of computational thinking with one example each.

Answer: ______________________________

- Q2. Write pseudocode for an algorithm that finds the minimum value in an unsorted list.

Answer: ______________________________

- Q3. What is the difference between a recursive and iterative algorithm? Give an example of each.

Answer: ______________________________

- Q4. Apply the 6-step framework to this problem: find all pairs of integers in a list that sum to a target value.

Answer: ______________________________

SENG 11232 Engineering Foundation · Lecture 02: Computational Thinking & Problem Decomposition