04-630
Data Structures and Algorithms for Engineers

Lecture 1: Introduction

George Okeyo: gokeyo@andrew.cmu.edu

Lecture 1

Introduction & The Software Development Life Cycle

- Motivation
- Goals of the course
- Syllabus & lecture schedule
- Course operation
- Preview of selected course material
- Software development tools for exercises and assignments
- Levels of abstraction in information processing systems
- The software development life cycle
- Software process models
- C/C++ revision dive

Logistics and Admin

- **Instructor**: Dr. George Okeyo, Assistant Teaching Professor, CMU Africa
- **Email**: gokeyo@andrew.cmu.edu
- **TAs**: Galane Basha Namomsa, Ghislain Bisamaza , Agasaro Ines Ituze, Elyse Manzi, Elvis Iraguha, Innocent Ingabire.
- **Class Time**: 1400-1550HRS, Monday, Wednesday
- **Location**: F205, F305
- **Course Credit**: 12 units
- **Office Hours:** Monday (17-18Hrs); Wednesday(16-17Hrs)

Who am I?

- George Okeyo joined Carnegie Mellon University Africa in **February 2021**. Prior to joining CMU Africa, he was a faculty member at the **De Montfort University, Leicester (UK)**. George remains an *honorary lecturer* at De Montfort University. He was previously a Lecturer at the Jomo Kenyatta University of Agriculture and Technology (JKUAT), Kenya.

- George completed his *Ph.D. in Computer Science from Ulster University(UK)* in 2013. He holds a *Master's degree in Information Systems* from *the University of Nairobi (Kenya)* and a *B.Sc. In Mathematics and Computer Science* from *JKUAT*.

- His ***research focus*** includes smart environments, mobile big data, semantic technologies, activity recognition, climate modelling, among others.

Assistant Teaching Professor
Department: CMU-Africa
Email: gokeyo@andrew.cmu.edu

4

Motivation

Software is everywhere, not only in IT sectors:

- Robotics & automation
- Automotive
- Aerospace
- Communications
- Medical
- Energy distribution and management
- Environmental control
- …

Motivation

Most software is in embedded systems

- Highly constrained in terms of
  - Memory
  - Processing power
  - Bandwidth

- Have exacting requirements for reliability, safety, availability

Motivation

Engineers who develop the software

- Do not always have a strong background in
    - Computer science
    - Computer engineering
    - Algorithms
    - Data Structures
- Have formal education in other engineering disciplines

Motivation

This is a problem …

- Suppose you've developed a software application
- And it works just fine in the current set of circumstances
- But can you be sure it will scale?
    - Larger data sets (input)
    - Larger user base
    - Tighter time and memory constraints
    - Migration to a distributed computing environment
- This is where a solid foundation in data structures & algorithms comes in

Motivation

Example 1

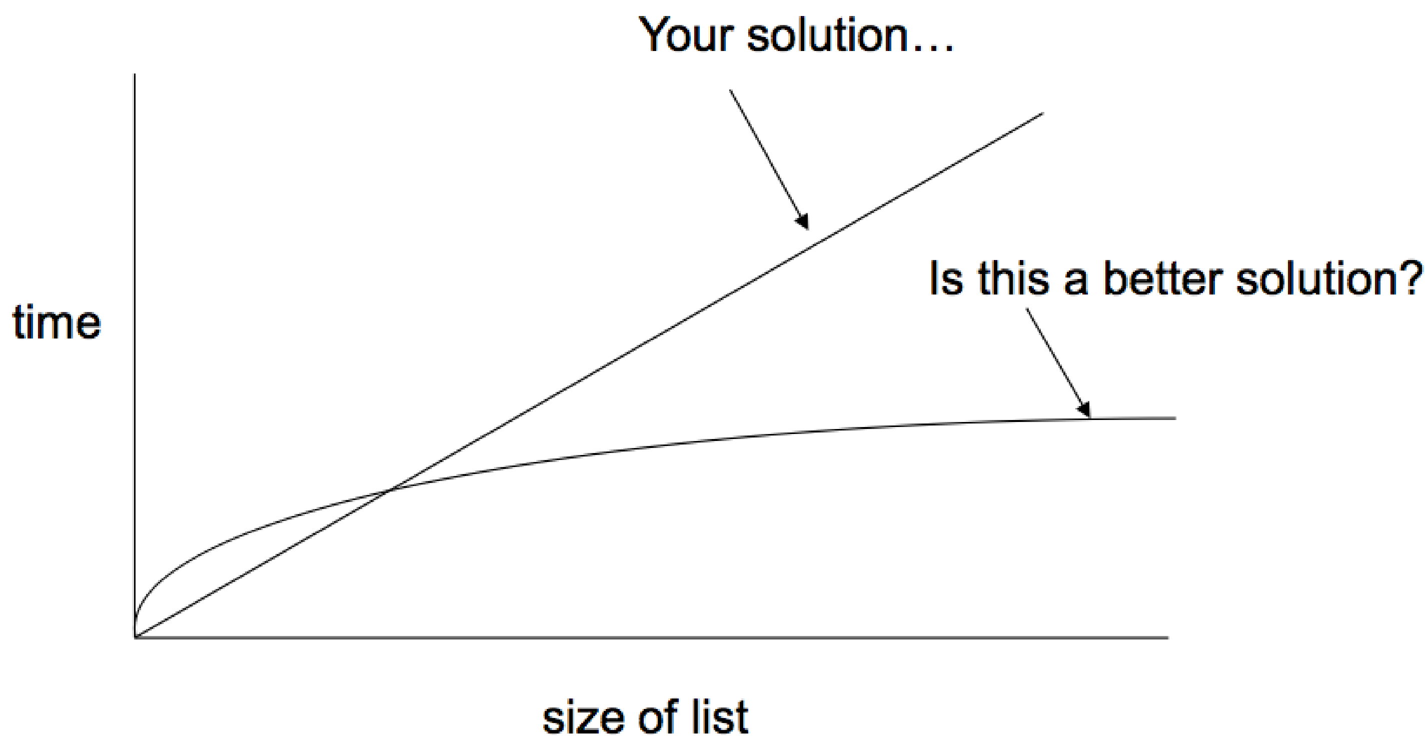
- **Problem**: Your program needs to find whether a list stored in memory contains a particular data element

- **Your solution**: Start from the beginning of the list and examine each element

- How good is this? What does it depend on?
- Can you do better?
- Under what circumstances could you improve this?
- Is the list the optimal data structure for this?

Motivation

Example 1



Motivation

Example 2

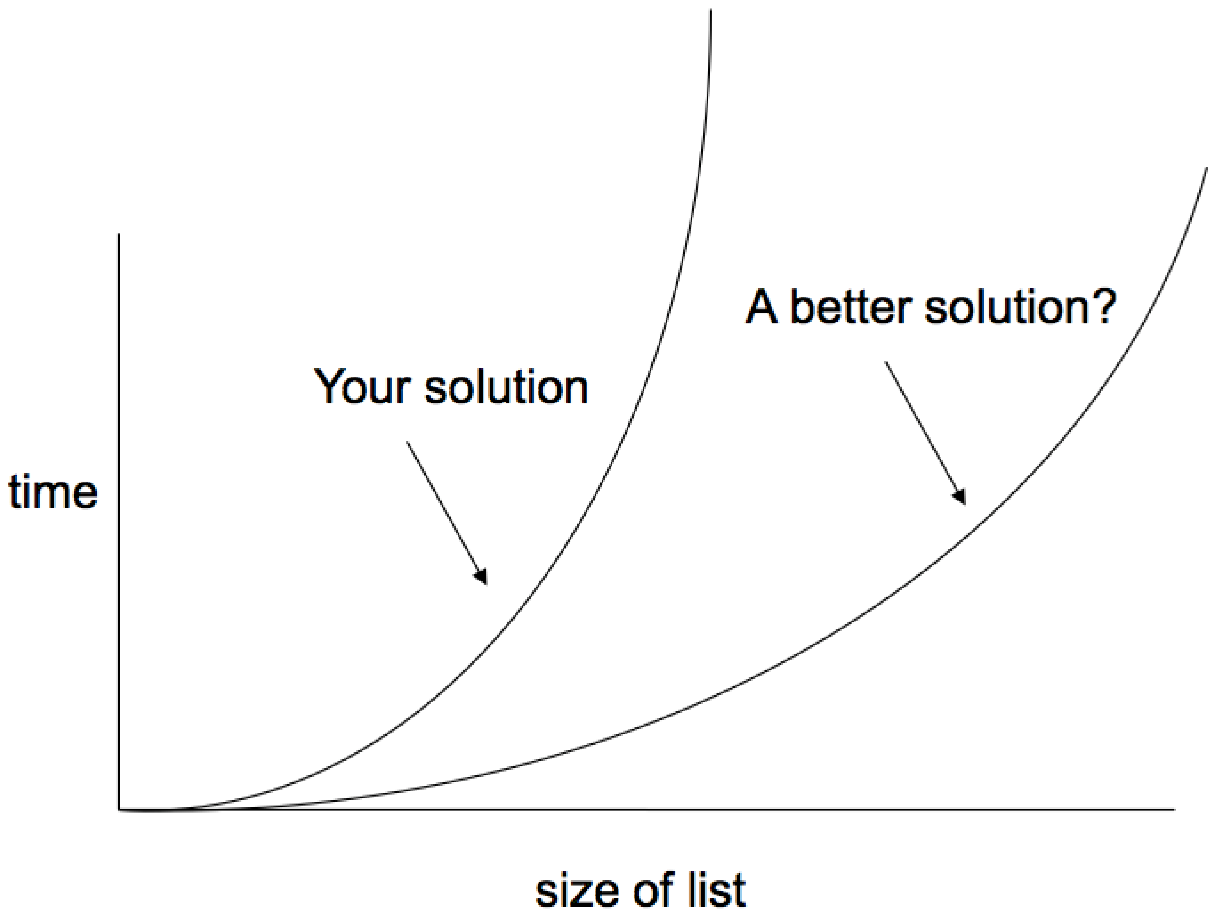
- **Problem:** You need to output a sorted list of elements stored in memory

- **Your solution:** Find and output the largest; find and output the next largest, ...

- How good is this?
- Can you do better?
- Under what circumstances?

Motivation

Example 2



Motivation

Example 3

- **Problem:** You are creating a car navigation assistant to devise a route that will allow the driver to visit a set of cities optimally, e.g., minimize fuel consumption, distance, or time

- This is the classic Travelling Salesman problem

- **Your solution:** List out all possible ways of visiting all cities. Select the one that minimizes the total distance traveled

- How good is this?
- Can you do better?

Motivation

Example 3

- **Your solution:**
  Assuming 1 microsecond to generate each path:

Cities	Computing time
2	Really fast
7	~1 Second
11	~1 Hour
12	~1 Day
14	~1 Year
17	~1 Century

- Can you do better? If so, what will it take?

Motivation

Example 4

- **Problem:** You have a system with a lot of legacy code in it, much of it is believed to be obsolete. You want to write a general program to find the code segments that are never actually executed in a system, so that you can then remove them

- Your solution: ?????

Motivation

So What?

- We have seen instances of four kinds of problem complexity that occur all the time in industry
  - Linear
  - Polynomial
  - Exponential
  - Undecidable

- Knowing which category your problem fits into is crucial
  - You can use special techniques to improve your solution

Motivation

So What?

- Competitive advantage is based on the characteristics of products sold or services provided
  - Functionality, timeliness, cost, availability, reliability, interoperability, flexibility, simplicity of use
- Innovation will be delivered through quality software
  - 90% of the innovation in a modern car is software-based
- Software determines the success of products and services

Goals of the Course

- Provide engineers (especially)who don't have a formal background in computer science with a solid foundation in the key principles of data structures and algorithms

- Leverage what software development experience they do have to make them more effective in developing efficient software-intensive systems

Goals of the Course

- Foster algorithmic thinking

- Appreciate the link between
  - Computational theory
  - Algorithms and Data Structures
  - Software implementation

- Impart professional practical skills in software development

- Develop the ability to recognize & analyze critical computational problems and assess different approaches to their solution

Goals of the Course

Key themes

- Principles and practice (analysis and synthesis)
- Practical hands-on learning (lots of examples and programming assignments)
- Detailed implementation, not just pseudo-code
- Broad coverage of the essential tools in algorithms and data structures

Syllabus & Lecture Schedule

Available on Canvas

Course Operation

Course Operation

- Lectures will be posted in advance[effective second week]:
  - read them before coming to class
  - read them again after class
- Readings: read them after class
- Recitations: work through solutions and prompt questions.

Course Operation

Assessment (I could vary the weights---you will be informed)

- 6-7 individual programming assignments (10% each; best six)

- Mid-semester examination (10%)

- Final examination (30%)

- Marking schemes will be distributed in due course
  - Functionality (based on testing using an unseen data set)
  - Documentation: internal and external
  - Tests and testing strategy

- Strict deadlines:
  Late submissions (even by 10 seconds) will not be accepted

  NO EXTENSIONS except on compassionate grounds

Course Operation

Assignments & Labs

- Labs provide guidance on how to approach assignments
- Guidance for assignments will only address the input, output, and overall structure
  - You have to decide what algorithm and data structure to use
- The amount of guidance for assignments will decrease with every assignment
- Do your own assignment and do not collaborate
  - TAs and I check for similarities between submissions before grading
  - I will apply the sanctions for cheating and plagiarism very strictly

Course Operation

- Do
  - Participate in class
  - Ask questions (you will be doing others a favour)
  - Discuss course material, readings, assignments with other students
  - Share thoughts but not written material (e.g., code, documentation)
  - Cite any work you use in assignments
  - Be a good teammate: do your fair share of the work equally & cooperate

- Don't
  - Cheat or plagiarize
    - Uncited use of any material from anywhere
    - Share / steal any material with/from former or current students

- Sanctions for cheating and plagiarism
  - Zero marks for first sharing infringement (both parties)
  - Fail the course (grade R) for second sharing infringement (both parties)
  - Fail the course (grade R) for first stealing infringement

Course Operation

Assignments & Labs

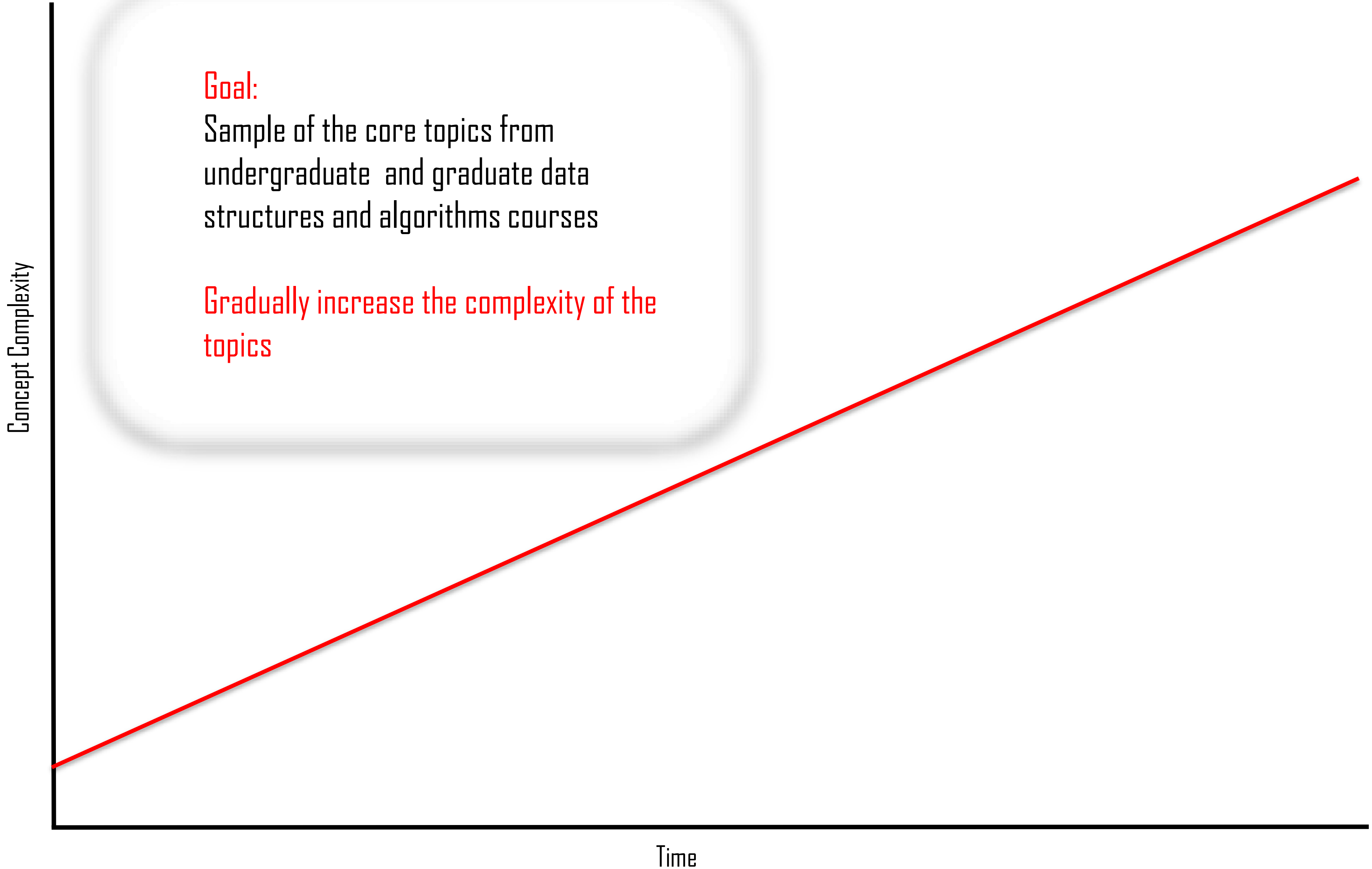
- Use of the Standard Template Library is not allowed
- We will cover the use of STL at the end of the course
  - At which point ...
  - We know how the STL data structures & algorithms are implemented
  - We know their strengths and weaknesses
  - We know when to use them
  - Learning STL at that point is just a matter of learning the API

Preview of Selected Course Material

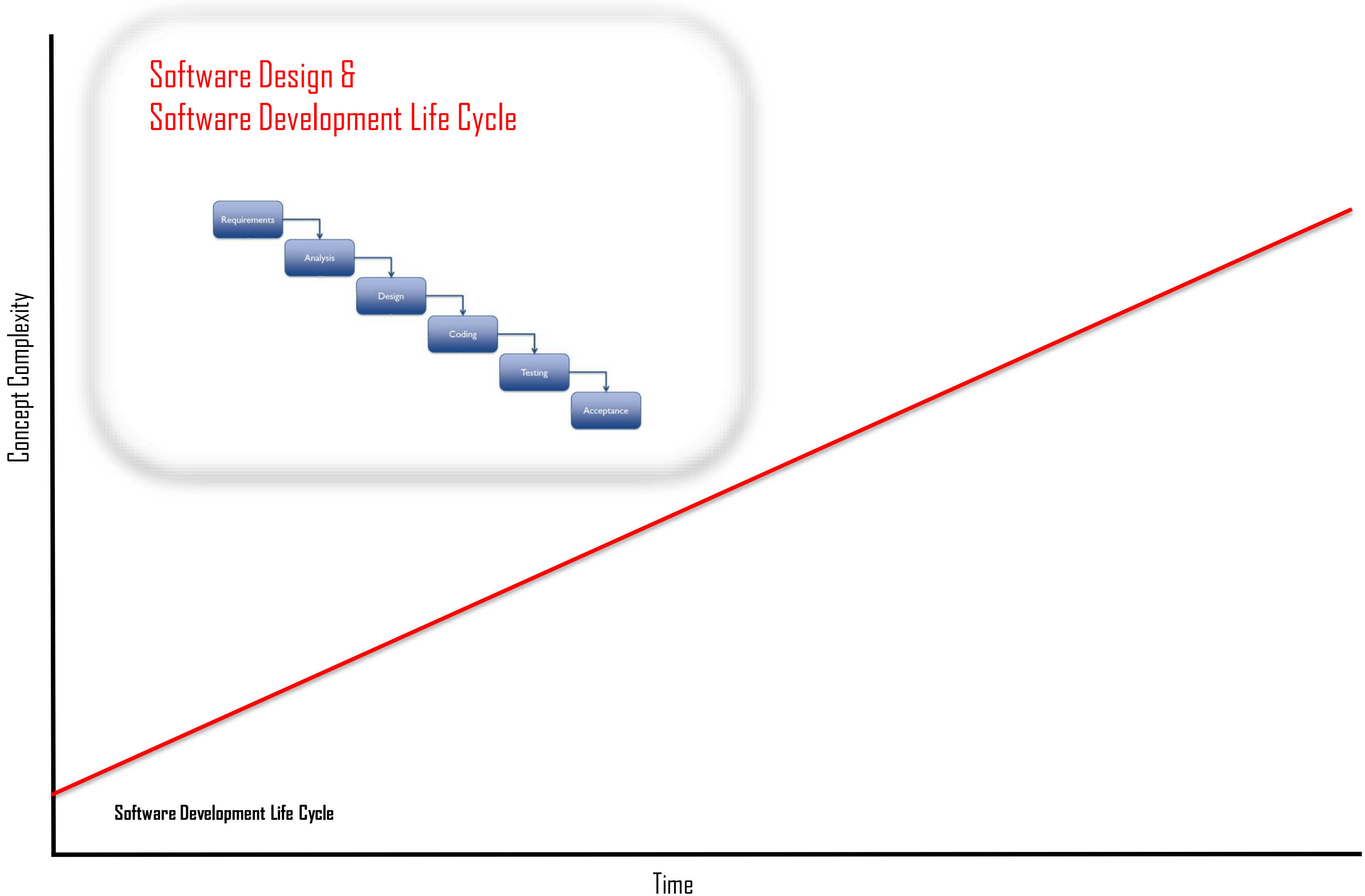
Data-Structures and Algorithms for Engineers

Goal:
Sample of the core topics from undergraduate and graduate data structures and algorithms courses

Gradually increase the complexity of the topics



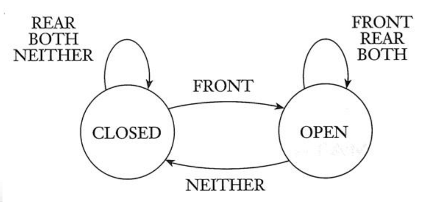
Data-Structures and Algorithms for Engineers



Data-Structures and Algorithms for Engineers

Formalisms for representing algorithms

I/O, Flow-charts, Pseudo-code, FSM, UML, ...



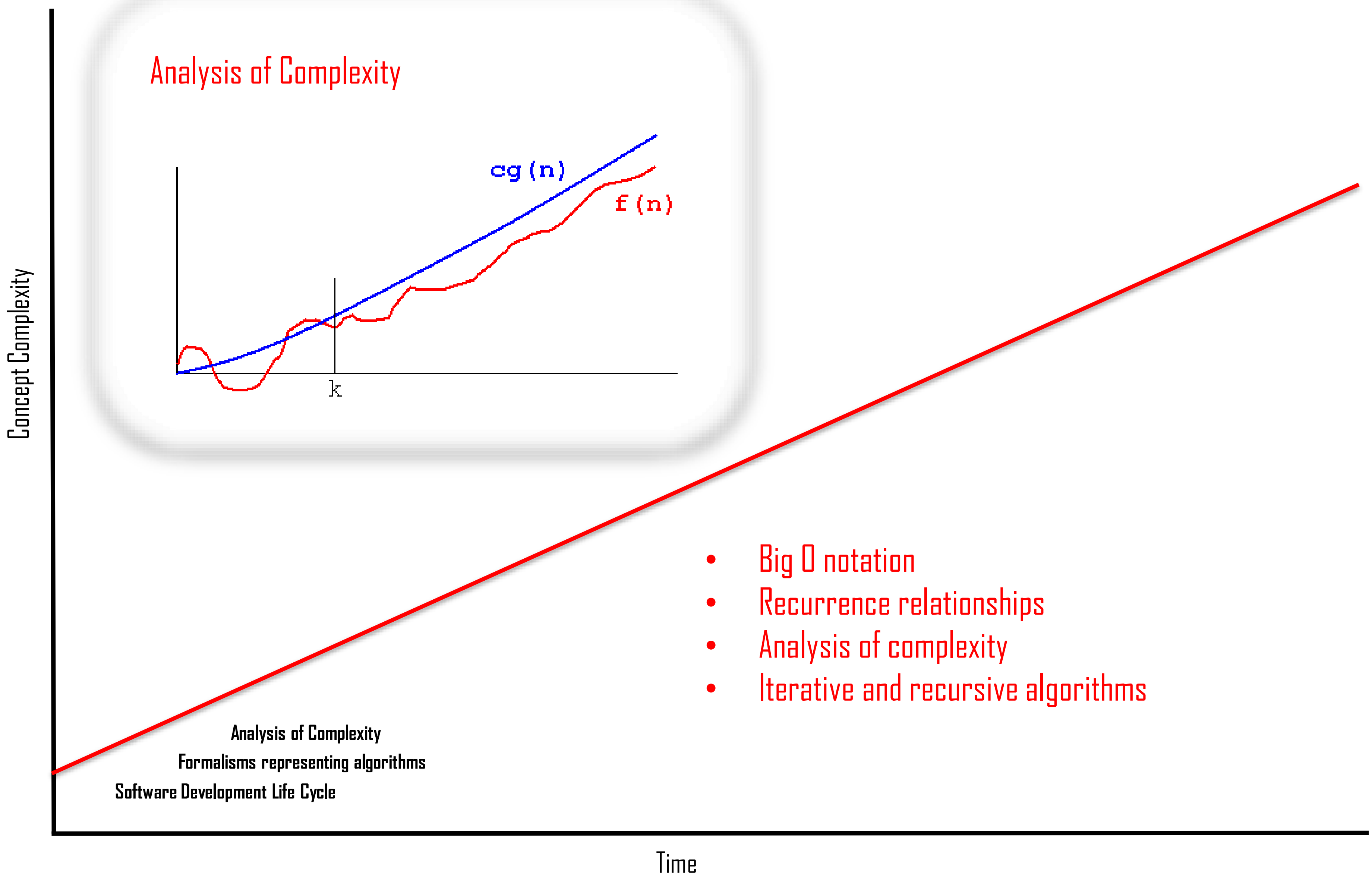
Concept Complexity

Formalisms representing algorithms

Software Development Life Cycle

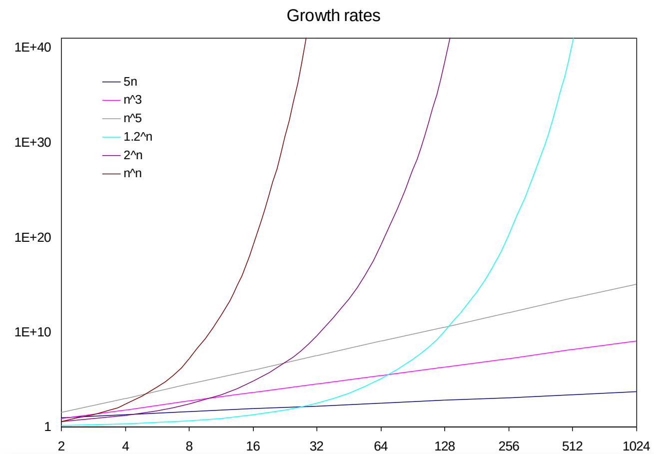
Time

Data-Structures and Algorithms for Engineers



Data-Structures and Algorithms for Engineers

Analysis of Complexity



- Tractable, intractable complexity
- Determinism and non-determinism
- P, NP, and NP-Complete classes of algorithm

Analysis of Complexity

Formalisms representing algorithms

Software Development Life Cycle

Concept Complexity

Time

Data-Structures and Algorithms for Engineers

Analysis of Complexity

	function/ n	10	20	50	100	300
Polynomial	n^2	1/10,000 second	1/2,500 second	1/400 second	1/100 second	9/100 second
	n^5	1/10 second	3.2 seconds	5.2 minutes	2.8 hours	28.1 days
Exponential	2^n	1/1000 second	1 second	35.7 years	400 trillion centuries	a 75 digit-number of centuries
	n^n	2.8 hours	3.3 trillion years	a 70 digit-number of centuries	a 185 digit-number of centuries	a 728 digit-number of centuries

- Tractable, intractable complexity
- Determinism and non-determinism
- P, NP, and NP-Complete classes of algorithm

Concept Complexity

Analysis of Complexity

Formalisms representing algorithms

Software Development Life Cycle

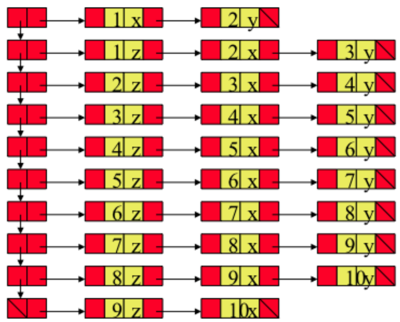
Time

Data-Structures and Algorithms for Engineers

Analysis of Complexity

x	y	0	0	0	0	0	0	0	0
z	x	y	0	0	0	0	0	0	0
0	z	x	y	0	0	0	0	0	0
0	0	z	x	y	0	0	0	0	0
0	0	0	z	x	y	0	0	0	0
0	0	0	0	z	x	y	0	0	0
0	0	0	0	0	z	x	y	0	0
0	0	0	0	0	0	z	x	y	0
0	0	0	0	0	0	0	z	x	y
0	0	0	0	0	0	0	0	z	x

n x n matrix:
$O(n^2)$ space complexity



2x(2 + 4 + 4) + (n-2)x(2 + 4 + 4 + 4)
= 20 + 14n - 28 = 14n - 8:
O(n) space complexity

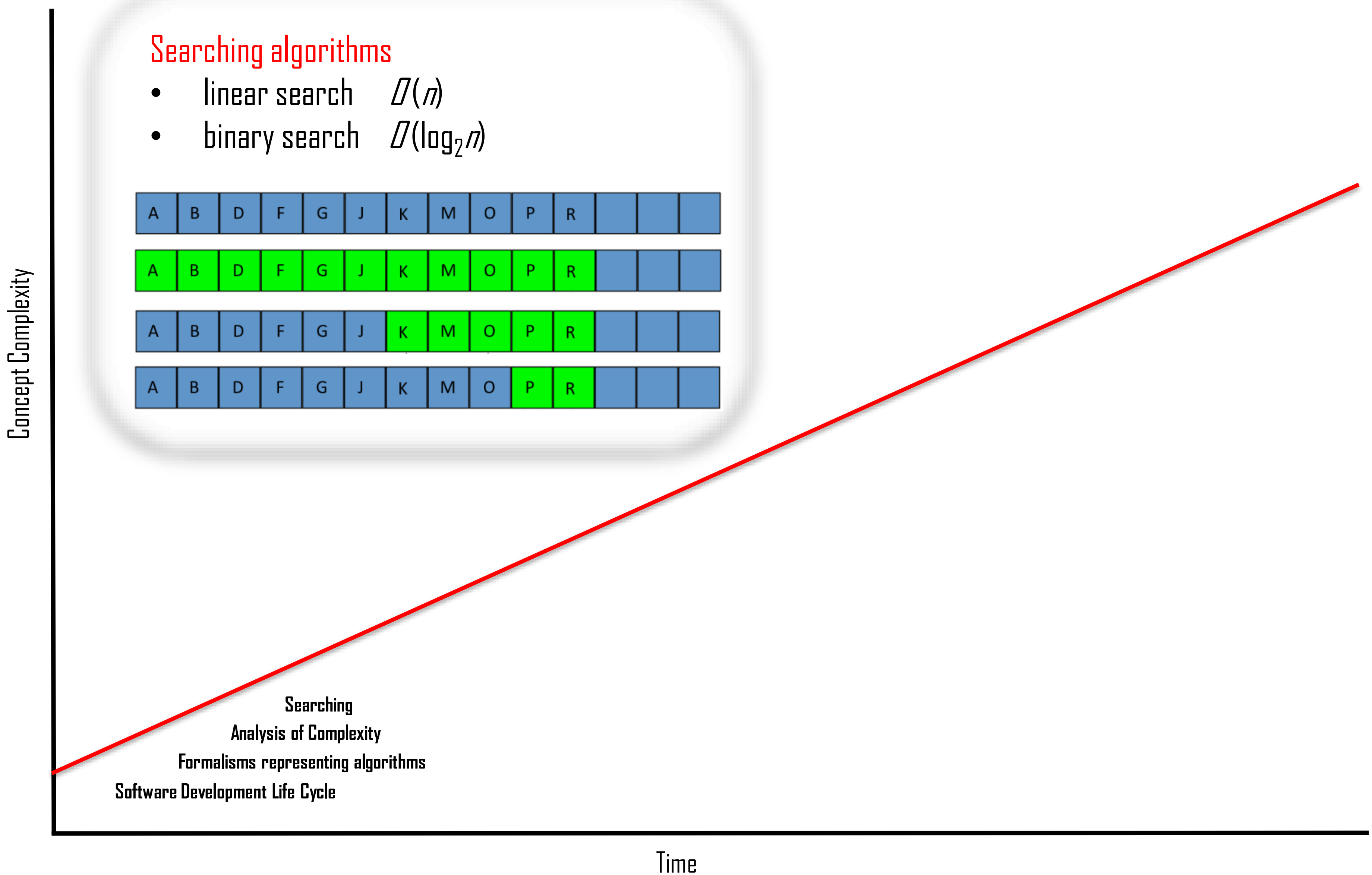
Concept Complexity

Time vs. space complexity

Analysis of Complexity
Formalisms representing algorithms
Software Development Life Cycle

Time

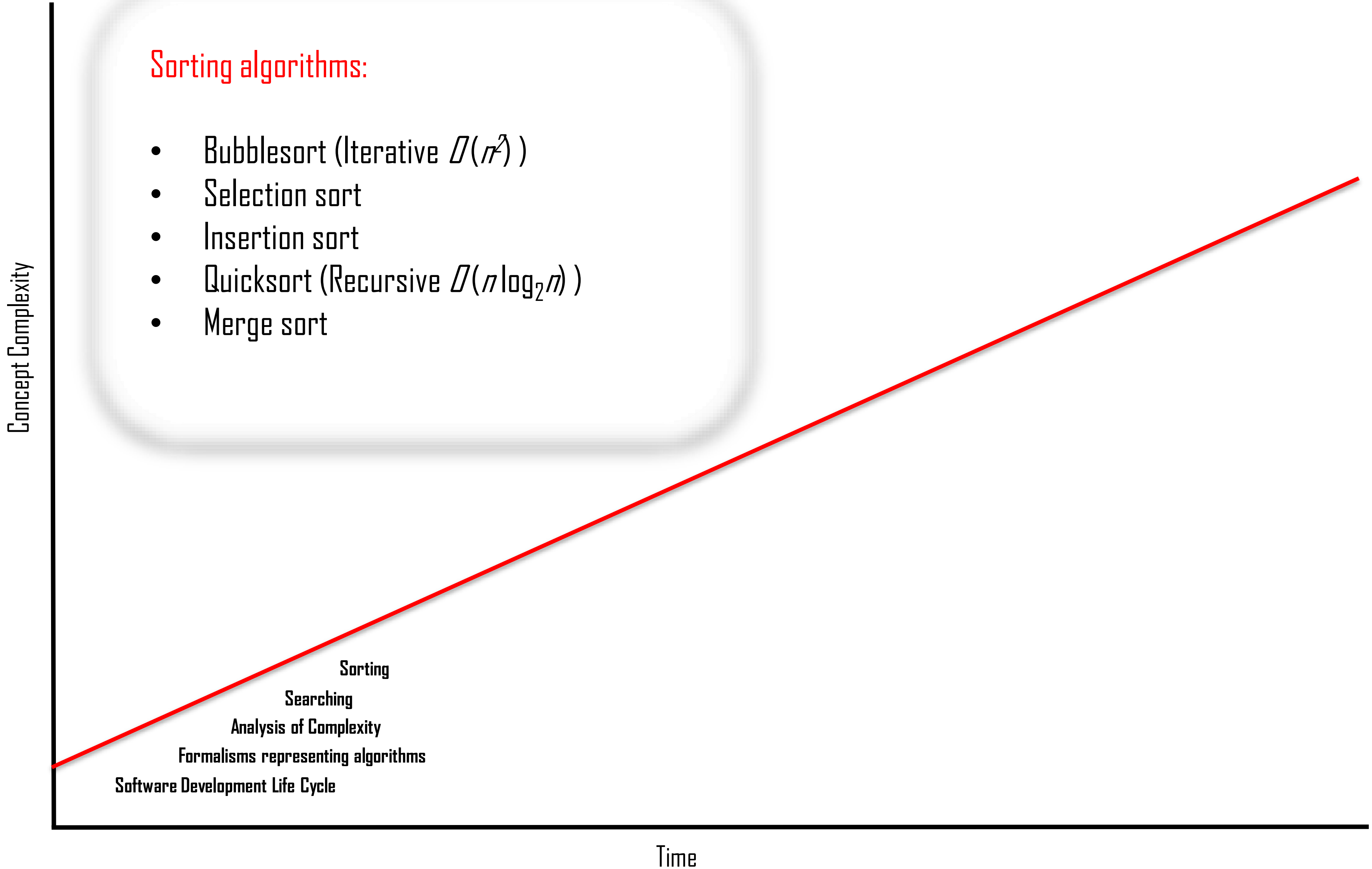
Data-Structures and Algorithms for Engineers



Data-Structures and Algorithms for Engineers

Sorting algorithms:

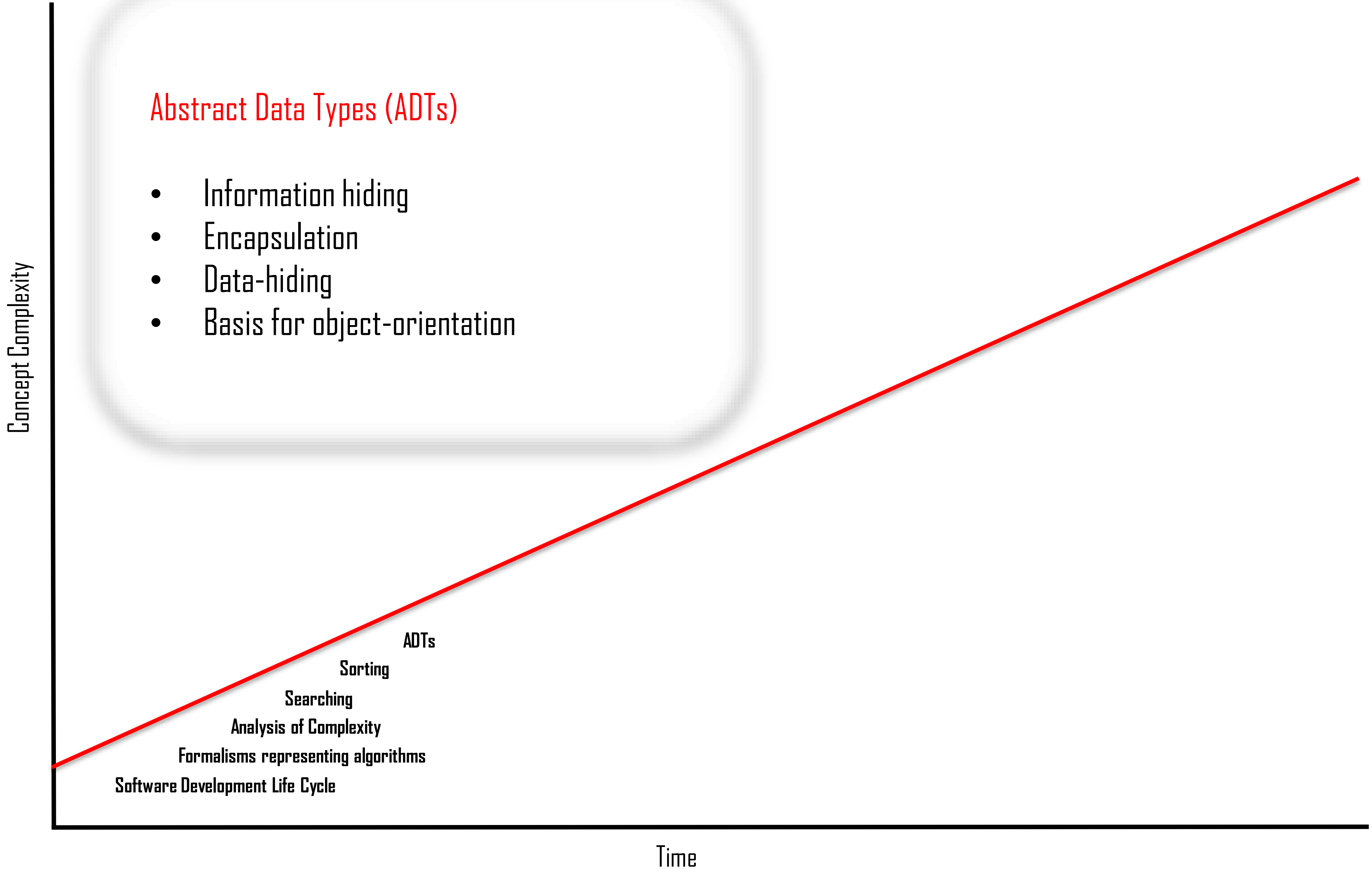
- Bubblesort (Iterative $O(n^2)$)
- Selection sort
- Insertion sort
- Quicksort (Recursive $O(n \log_2 n)$)
- Merge sort



Data-Structures and Algorithms for Engineers

Abstract Data Types (ADTs)

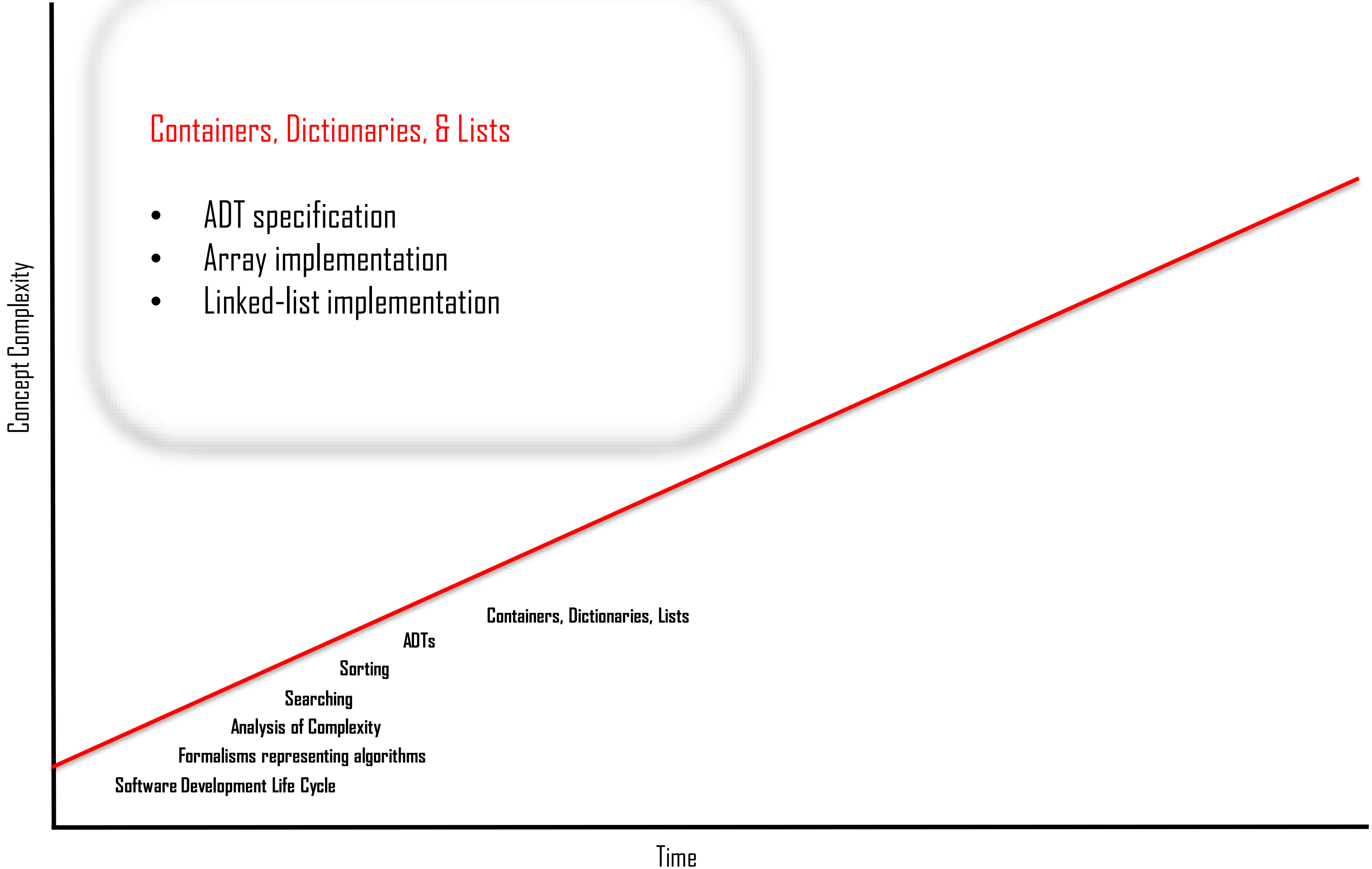
- Information hiding
- Encapsulation
- Data-hiding
- Basis for object-orientation



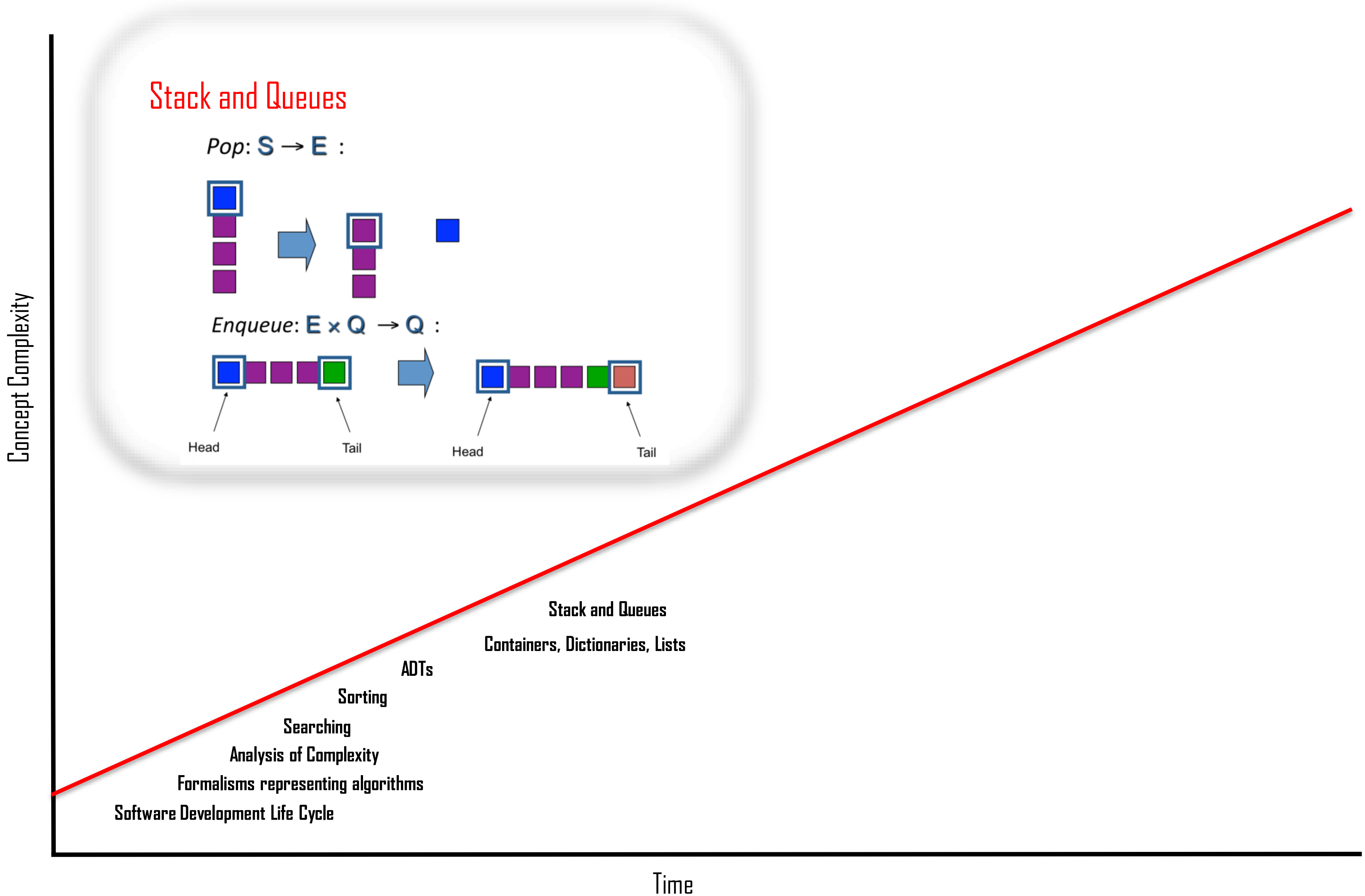
Data-Structures and Algorithms for Engineers

Containers, Dictionaries, & Lists

- ADT specification
- Array implementation
- Linked-list implementation



Data-Structures and Algorithms for Engineers



Data-Structures and Algorithms for Engineers

Trees

- Binary trees
- Binary search trees
- Tree traversal
- Applications of trees
  (e.g. Huffman coding)
- Height-balanced trees
  (e.g. AVL Trees, Red-Black Trees

Concept Complexity

Trees

Stack and Queues

Containers, Dictionaries, Lists

ADTs

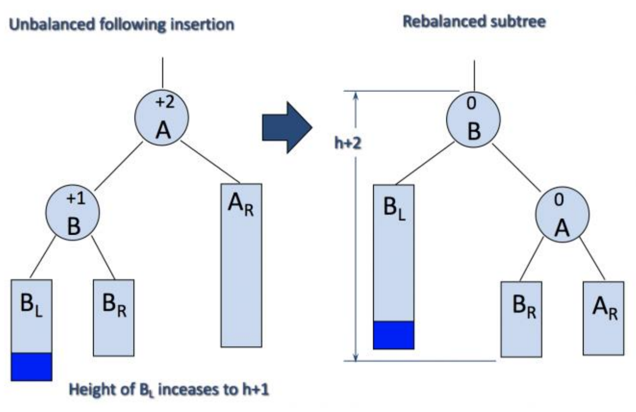
Sorting

Searching

Analysis of Complexity

Formalisms representing algorithms

Software Development Life Cycle



Time

Data-Structures and Algorithms for Engineers

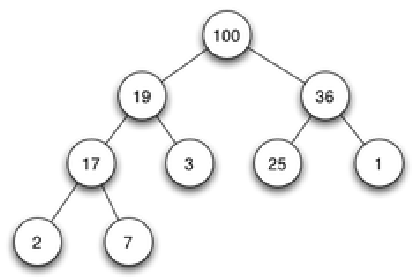
Heaps

Priority queues

Binary heaps, min/max-heaps

Heap operations

Heap sort



Concept Complexity

Heaps

Trees

Stack and Queues

Containers, Dictionaries, Lists

ADTs

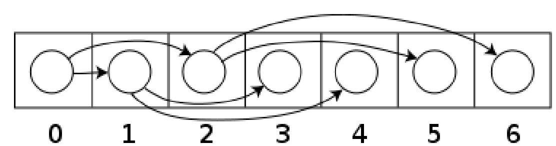
Sorting

Searching

Analysis of Complexity

Formalisms representing algorithms

Software Development Life Cycle

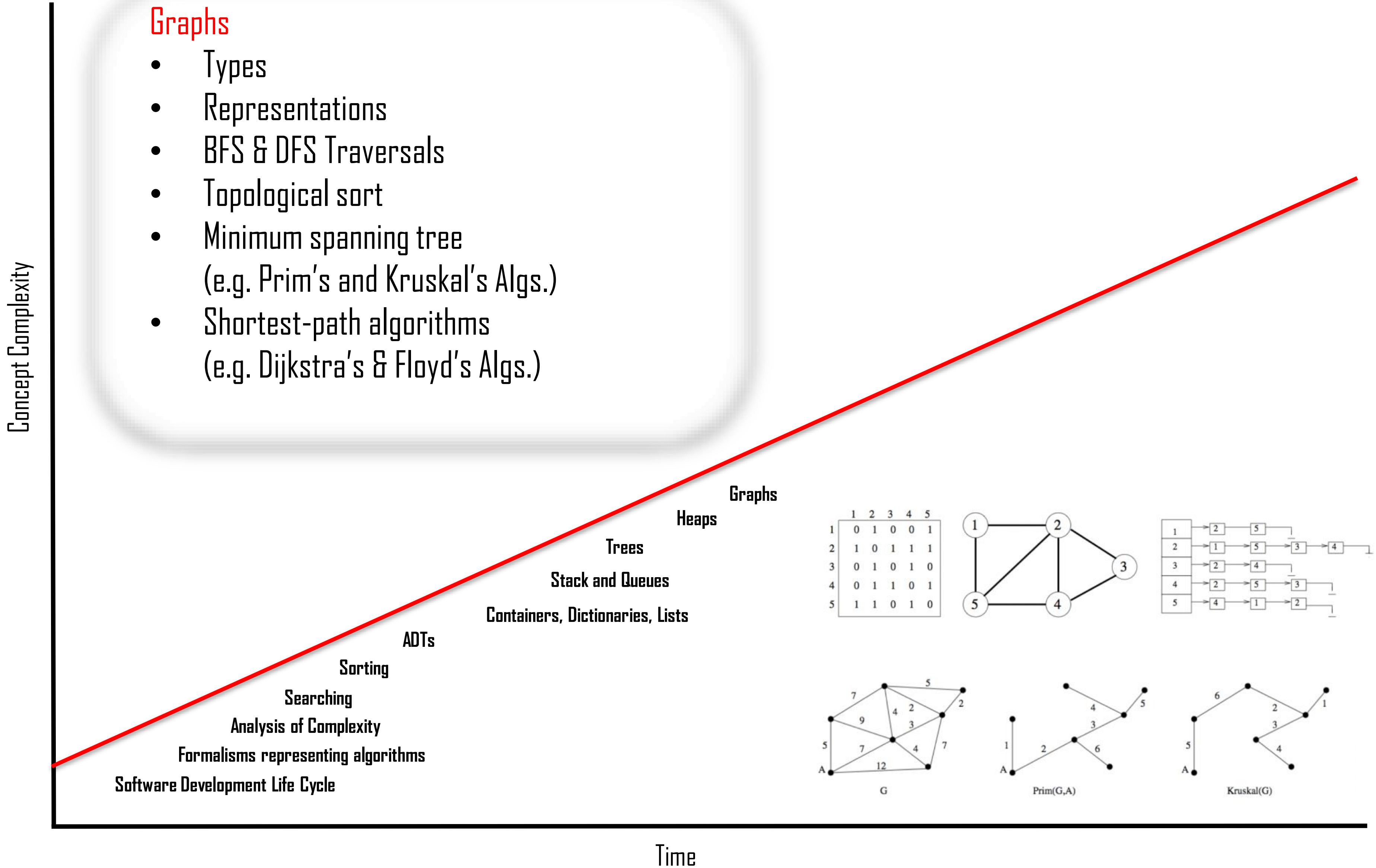


Time

Data-Structures and Algorithms for Engineers

Graphs

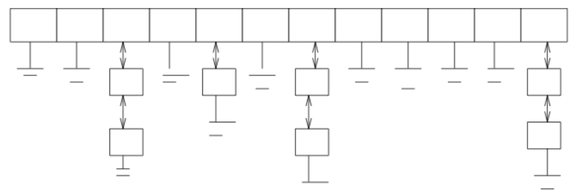
- Types
- Representations
- BFS & DFS Traversals
- Topological sort
- Minimum spanning tree
  (e.g. Prim's and Kruskal's Algs.)
- Shortest-path algorithms
  (e.g. Dijkstra's & Floyd's Algs.)

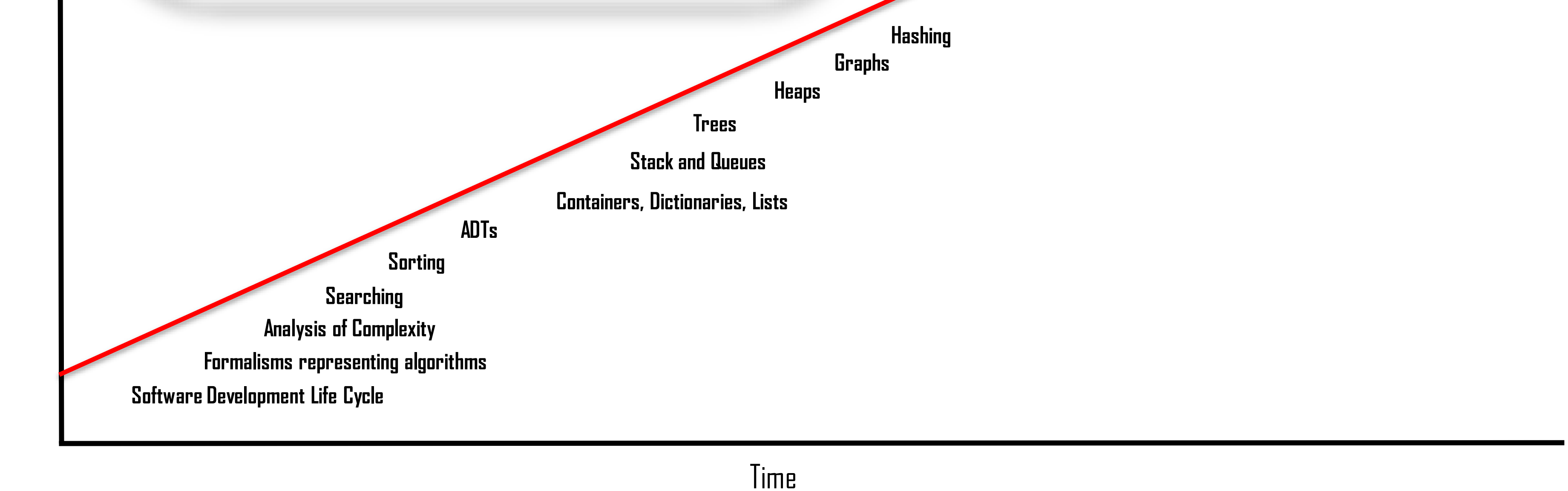


Data-Structures and Algorithms for Engineers

Hashing

- Hash functions
- Collisions
- Chaining & Probe policies





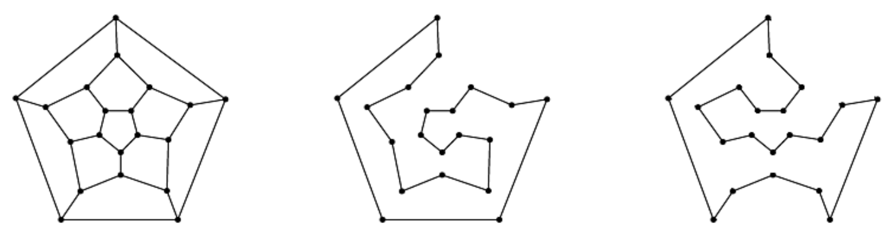
Data-Structures and Algorithms for Engineers

Algorithmic Strategies

- Brute-force
- Divide-and-conquer
- Greedy algorithms
- Combinatorial Search
- Backtracking
- Branch-and-bound

Concept Complexity

Algorithmic Strategies
Hashing
Graphs
Heaps
Trees
Stack and Queues
Containers, Dictionaries, Lists
ADTs
Sorting
Searching
Analysis of Complexity
Formalisms representing algorithms
Software Development Life Cycle

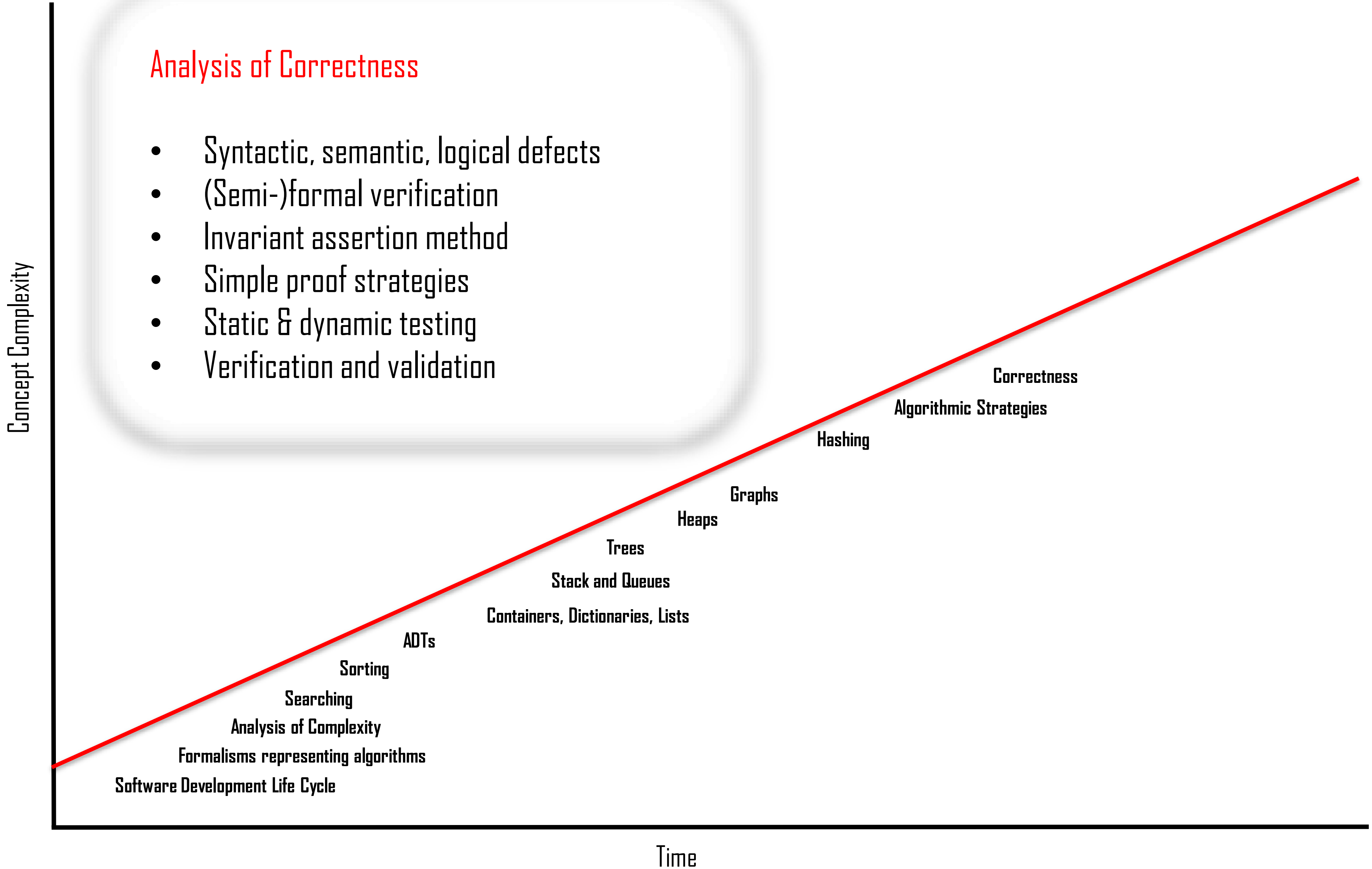


Time

Data-Structures and Algorithms for Engineers

Analysis of Correctness

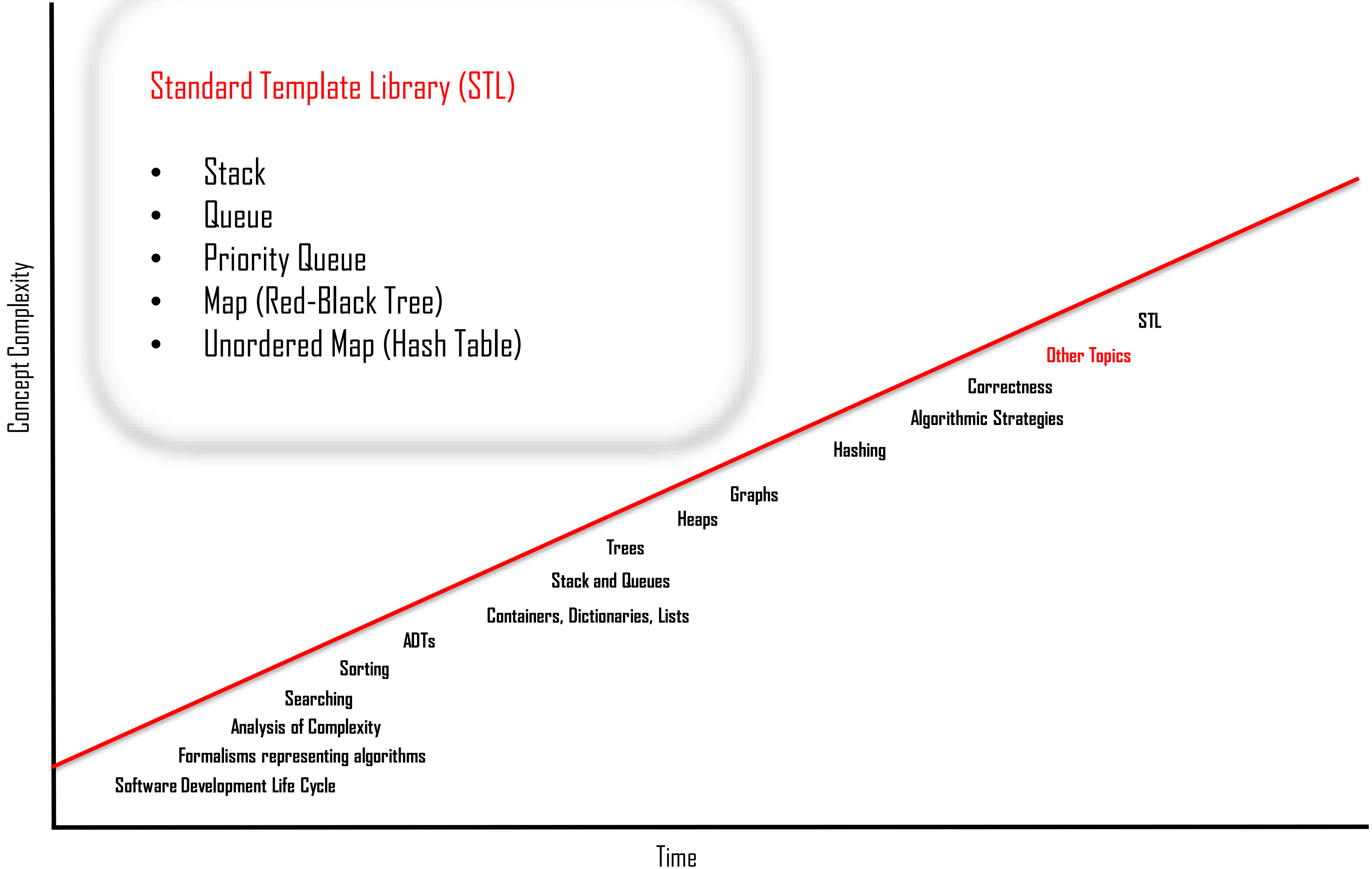
- Syntactic, semantic, logical defects
- (Semi-)formal verification
- Invariant assertion method
- Simple proof strategies
- Static & dynamic testing
- Verification and validation



Data-Structures and Algorithms for Engineers

Standard Template Library (STL)

- Stack
- Queue
- Priority Queue
- Map (Red-Black Tree)
- Unordered Map (Hash Table)



Software Development Tools for Exercises and Assignments

Software Development Tools for Exercises and Assignments

- Installation of software development environment
  - Set up Development environment.
  - Environment will let you analyze memory and CPU usage for your solutions.

Software Development Tools for Exercises and Assignments

- Preferred practice for software that supports encapsulation and data hiding (e.g. ADT & OO classes)

- 3 files: Interface, Implementation, and Application Files
  - Interface
    - between implementation and application
    - Header File that declares the class type
    - Functions, classes, are declared, not defined (except inline functions)
  - Implementation
    - `#includes` the interface file
    - contains the function definitions
  - Application
    - `#includes` the interface file
    - contains other (application) functions, including the `main` function

Software Development Tools for Exercises and Assignments

When writing an application, we are ADT/class users

- Should not know about the implementation of the ADT/class
- Thus, the interface must furnish all the necessary information to use the ADT/class
    - It also needs to be very well documented (internally)
- Also, the implementation should be quite general (cf. reusability)

Levels of Abstraction
in Information Processing Systems

Algorithms + Data Structures = Programs

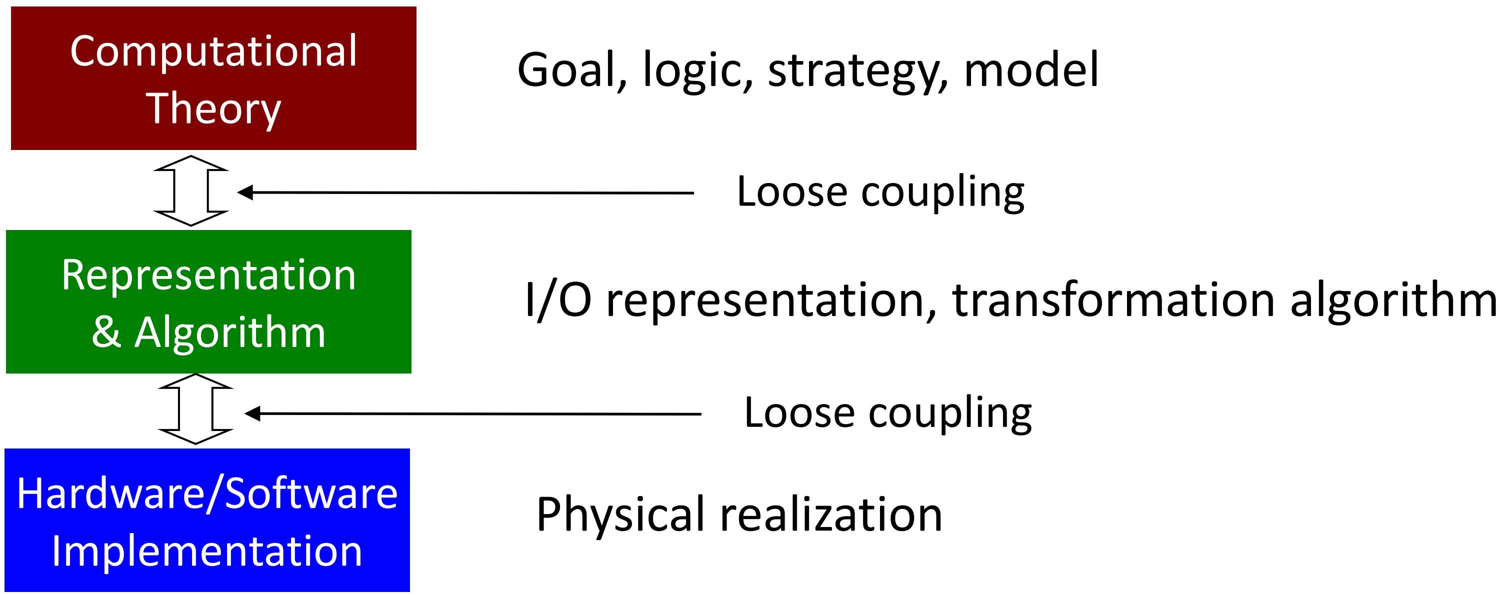
Niklaus Wirth, 1976

Inventor of Pascal and Modula programming languages
Winner of Turing Award 1984

1969

Information Processing:
Representation & Transformation

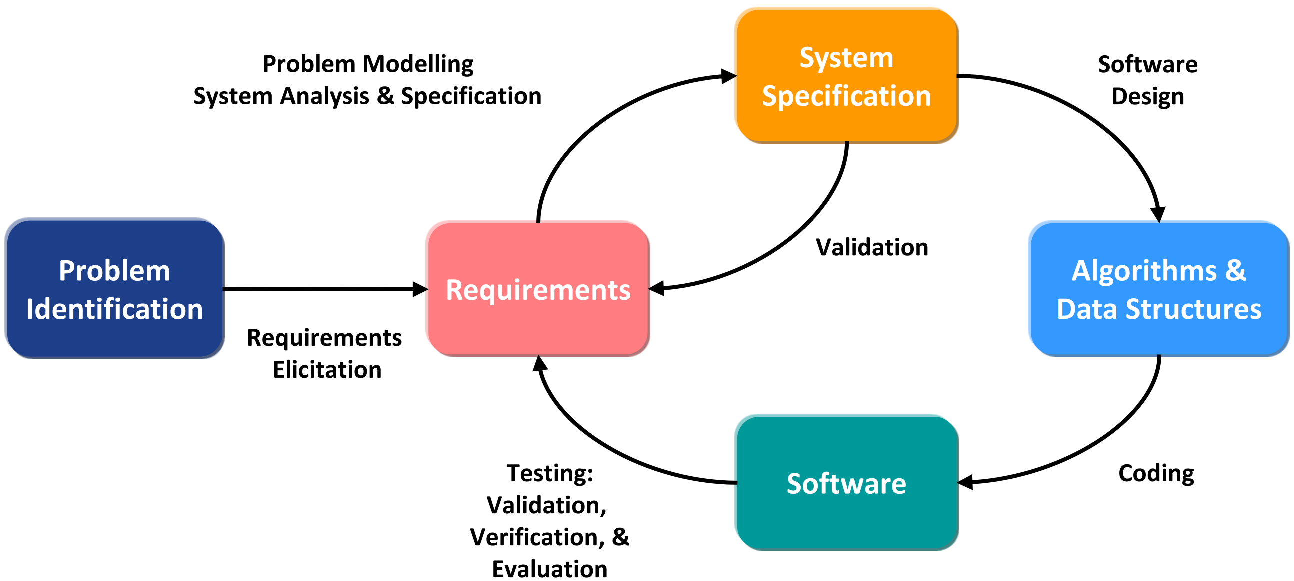
Marr’s Hierarchy of Abstraction / Levels of Understanding Framework



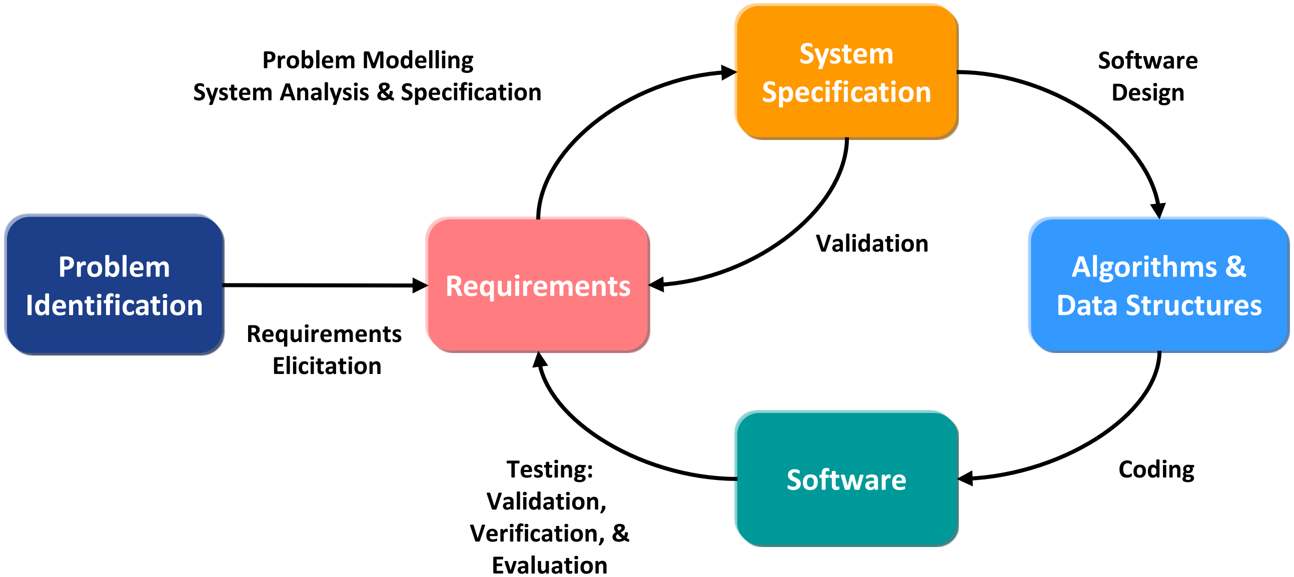
D. Marr and T. Poggio. "From understanding computation to understanding neural circuitry", in E. Poppel, R. Held, and J. E. Dowling, editors, Neuronal Mechanisms in Visual Perception, volume 15 of Neurosciences Research Program Bulletin, pages 470–488. 1977.
D. Marr. Vision. Freeman, San Francisco, 1982.
T. Poggio. The levels of understanding framework, revised. Perception, 41:1017–1023, 2012.

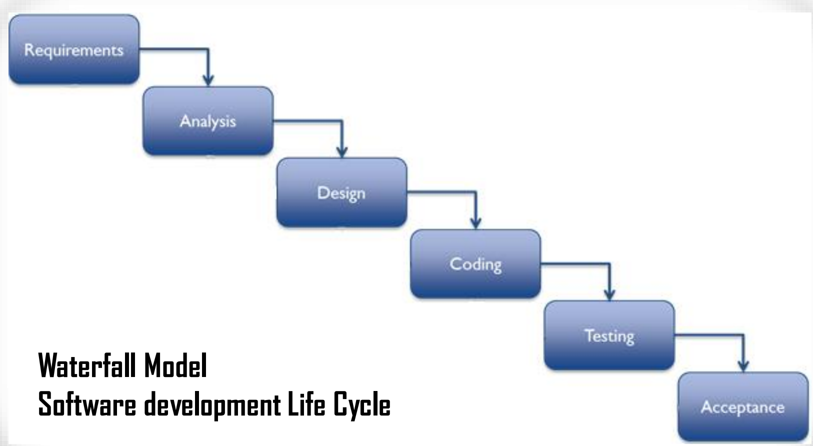
The Software Development Life Cycle

The Software Development Life Cycle



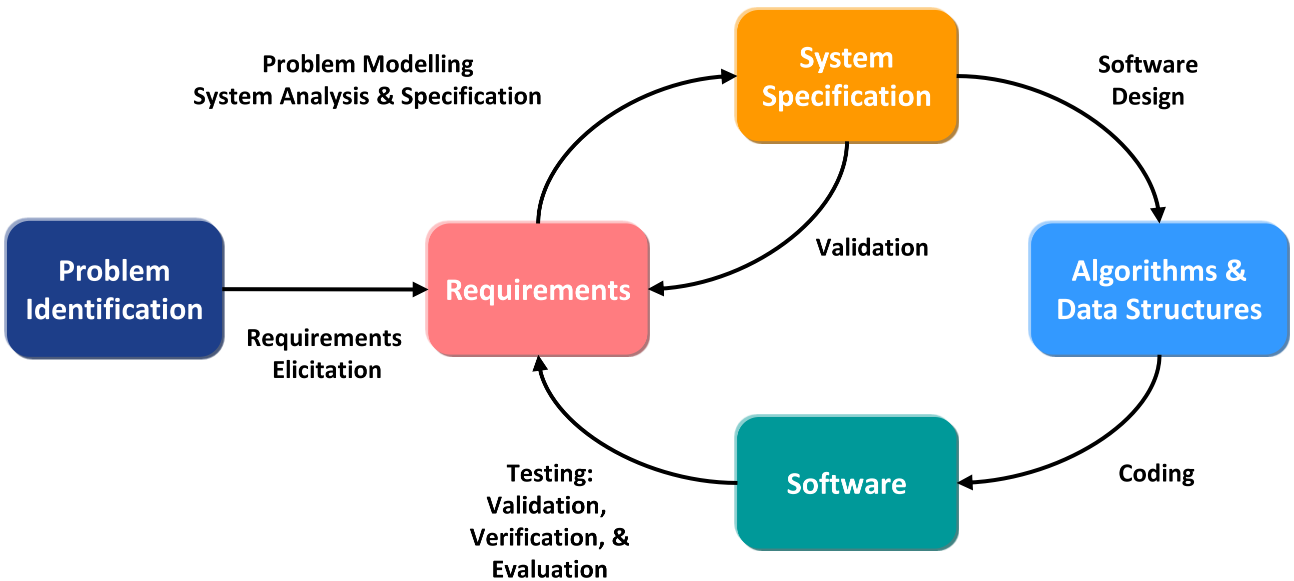
The Software Development Life Cycle





Waterfall Model
Software development Life Cycle

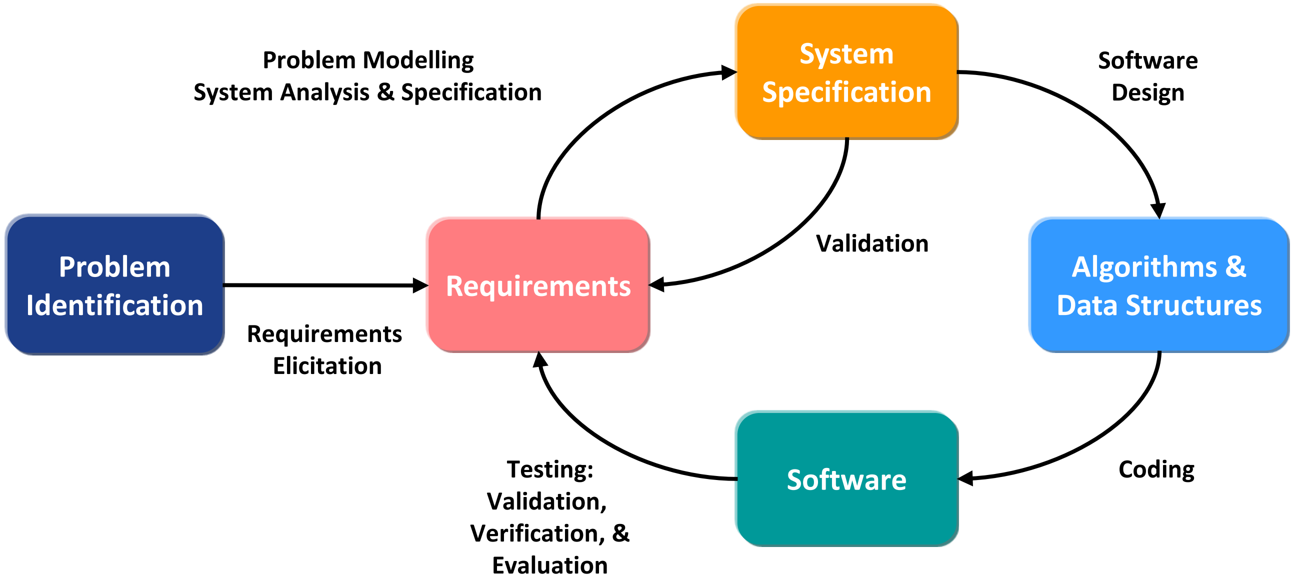
The Software Development Life Cycle



Life Cycle Models (Software Process Models):

- Waterfall (& variants, e.g. V)
- Evolutionary
- Re-use
- Hybrid
- Spiral
- ...

The Software Development Life Cycle



Software Development Methodologies:

Top-down
Structured
- Yourdon Structured Analysis (YSA)
- Jackson Structured Analysis (JSA)
- Structured Analysis and Design Technique (SADT)

Object-oriented analysis, design, programming
Component-based software engineering (CBSE)

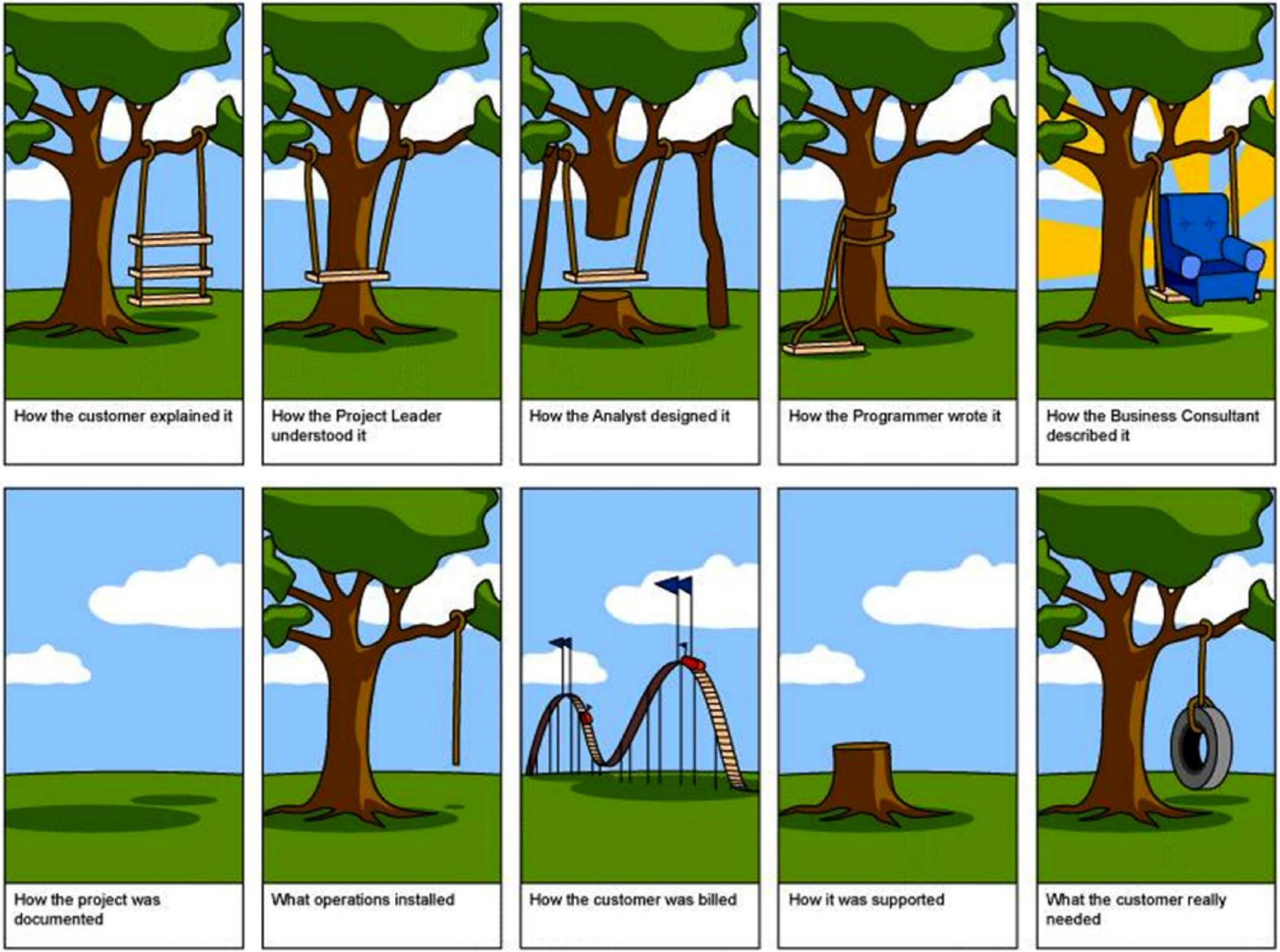
Software Development Life Cycle

1. Problem identification
2. Requirements elicitation
3. Problem modeling
4. System analysis & specification
5. System design
6. Module implementation and system integration
7. System test and evaluation
8. Documentation

Computational Theory

Representation & Algorithm

Hardware/Software Implementation


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Software Process Models

- The Waterfall model
  - Separate and distinct phases of specification and development
- Evolutionary development
  - Specification and development are interleaved
- Formal transformation
  - A mathematical system model is formally transformed to an implementation
- Reuse-based development
  - The system is assembled from existing components

Revision

- Review the material in Lecture 0 to bring your self up to speed regarding C/C++ programming.
- You can practice coding the revision material in any desired environment.

Acknowledgement

- Adopted and Adapted from Material by:
- David Vernon: vernon@cmu.edu ; www.vernon.eu