Chapter 1
Introduction to Software Engineering

Why Engineer Software?
- Air traffic control case study
  - $2.3 Billion spent without any usable deliverable
- Enormous social cost of NOT engineering software
- Too many large-scale software development failures for too many decades
- Software development initiatives fail for many different reasons, most of which are addressed by good software engineering practices

Sources of Complexity in Software Development
- Technical staff lacks expertise in the complex application domain
- Difficulty in interpersonal communications
- Ambiguity of natural language, used to express user needs
- Difficulty of mentally grasping and coordinating the details of a large-scale development project

How Software Development Projects Fail
- No functioning software results
- The resulting software does not adequately address the need of the users
- Software contains incorrect computations
- The software is too difficult to use correctly
- The system response time is too slow to be used without frustration

Why Inter-personal Communication is Difficult
- Technical staff and end users have different background knowledge
- Technical staff often do not possess the vocabulary used in the application domain
- End users frequently do not understand how to effectively express their needs in terms that are clear to systems developers
- Ambiguity of natural language

Maintaining Software Systems
- Maintenance is an inevitable aspect of software development
- It allows software to evolve with an organization
- Too often too many resources are directed to maintenance and away from new development
- Software requires excessive maintenance efforts if it is poorly structured
- It is better to invest in well-structured software up front then to invest excessively in maintenance later
Elements of a Software Development Paradigm

- **Conceptualization**
  - What elements are instrumental in how the developers think about the software system?

- **Representation**
  - Must convey what the software project is all about
  - Most effectively done with a software modeling notation

- **Implementation**
  - Addresses how the source code is structured

A Brief History of Software Engineering Techniques

- **Structured Programming**
  - No gotos

- **Functional Decomposition**
  - Top-down organization of subprograms

- **Structured Analysis**
  - Recognition that analyzing the problem statement has critical influence on the success of the overall project
  - Formal modeling of subprogram interaction with dataflow diagrams

More History of SE Techniques

- **Data-centered analysis**
  - Uses techniques developed in structured analysis
  - Data modeling occurs using entity relationship diagrams before functional modeling

- **Object-oriented analysis**
  - No longer segregates the modeling of functions and data
  - Objects aggregate data with functions that operate on the data

Sample Dataflow Diagram

Sample Entity Relationship Diagram

Principles of the Object-Oriented Paradigm

- **Abstraction**
- **Modularity**
- **Modeling**
- **Inheritance**
  - Encapsulation (covered in chapter 2)
- **Polymorphism** (covered in Chapter 2)
Abstraction

- Expressing the software requirements in abstract terms, suppressing details so that a comprehensive portrayal can be made.
- The abstract representation must then be connected to more detailed expressions, so that no information is lost in modeling the system.
- This allows us to reason about the breadth of functionality of the developing system.

Modularity

- Breaking a complex software system into smaller modules allows reducing its overall complexity.
- Modules should aggregate some coherent aspect of the system.
- Modules should present a simple interface.
- Classes, functions, and abstract data types are examples of modules.

Abstract Data Type Example

### Interface

- POP
- PUSH(item)
- ISEMPTY()

### Stack

```
[12, 24, 98, 33, 13]
```

Implementation Option: Array
Implementation Option: Linked List

Modeling

- Modeling is the process of using a graphic notation to express properties of the software system.
- Models express how the system is built out of modules and how these modules interact.
- One objective of modeling is to express characteristics of the software system in an unambiguous manner.

Inheritance

- A property distinguishing classes from abstract data types.
- Allows characteristics from one class to be passed on to its subclasses.
- Subclasses may override inherited characteristics.

Cost of not Engineering Software

- Need to restructure code.
- High maintenance costs.
- Software is unacceptable to end users.
- Unreliable code.
- Need to rewrite code.
- Difficulty integrating system modules.
- Difficult project management.
- Budget and time overruns.
Why Software Engineering is not Universal

- Understanding software development as programming only without recognition of importance of analysis and design
- Short-sighted technical management
- Pressure of deadlines
- Poor project estimation, thus unreasonable deadlines

Working in Teams

- Be conscientious about due dates
- If you are less motivated by the class project, negotiate an equitable grade and take a lesser role
- Meet regularly with your team
- Always create an agenda for every team meeting
- Rotate responsibility for chairing team meetings