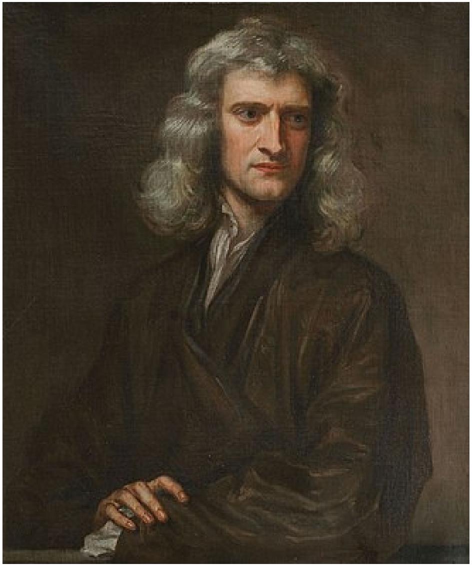
Software Engineering
Shoulders of Giants

Omkarendra Tiwari

Department of IT
IIIT Allahabad

February 2, 2023



If I have seen further it is by standing on the shoulders of Giants.

Outline

Part I: Giants

Programming

The art of programming is the art of organizing complexity.

Software Engineering

Apollo 11

Moon landing almost did not happen.

Test Driven Development

Listening, Testing, Coding, Designing.

Thats all there is to software.

I am not a great programmer. I'm just a good programmer with great habits.

Clean code

Would you rather Test-first, or debug-Later.

Good Practices in software development

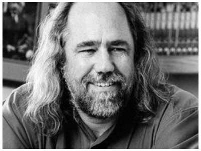
Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Version Control System

. . . there's almost always a range of responses, and

"**it depends**" is almost always the right answer in any big question.

A picture worth thousand words: Giants behind UML



Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change. —Grady Booch

The name of a class should reflect its intrinsic nature and not a role that it plays in an association. For example, Owner would be a poor name for a class in a car manufacturer's database. What if a list of drivers is added later? What about persons who lease cars? The proper class is Person (or possibly Customer), which assumes various different roles, such as owner, driver, and lessee.—James Rumbaugh

When a user uses the system, she or he will perform a behaviorally related sequence of transactions in a dialogue with the system. We call such a special sequence a use case.—Ivar Jacobson

Part II: Design and Architecture

Journey Ahead

Analysis

Design

Development

Maintenance and Evolution

Analysis: An Overview

Representing elicited requirements

- Text or diagrams are used
- Easier to communicate
- Quick to review for correctness, completeness and consistency

Analysis approaches

- Structured Analysis
  - Data and process transforming data are considered as two entities
- Object-oriented Analysis
  - Classes (objects) and their association/collaboration is modeled

Object-oriented Analysis

Diagrams

- Activity diagrams
- Use-case diagrams
- Class diagrams
- CRC model

Software Elements

- Statement
- Block
- Method
- Class
- Package
- Component
- Library/executables

Design

Plan, organize or structure

- Every software has a design
- Each design offers certain qualities (desirable or not)
- Design decisions are made throughout the SDLC
- A software is acceptable only if it satisfies user requirements

Requirements and Design

- User Requirements Objectives/tasks that the software should achieve
- Functional Requirements Functionalities/services a software system offers
- Non-functional Requirements How well the software accomplishes the task
- Requirements Elicitate it. Validate it. Change still arrives.
- Design Some decisions are taken early, some are delayed. Trade-offs are involved. You can not have all good qualities.

Requirements Dictate Design

Bicycle: Handlebar, suspension, tires

- Thick tire; Straight Handles; Front suspension for shock absorption;[a]
- Curve and drop handlebars; Light and aerodynamic alloy frames; [b]
- Straight handlebars and fat tire; [c]

[a] Mountain Bikes
[b] Road Bikes
[c] Snow/Sand Bikes

Object-oriented Thinking

Class

- Encapsulation of Data and Operations
- Abstraction of a real world entity or a phenomenon
- Represents a unit/module/component/subsystem/system/service

Class Elements

- Field State of the object
- Public Methods Behavior of the object

Design Decisions at Class Level[1]

Best Practices

- Keep the fields as Private
- Provide a constructor
- Initialize the fields via constructor
- Avoid getter/setter methods
- Five to seven public methods, not more
- Define *interface* and then *implement* in a class
- Check for anti-patterns and refactor the code

[1] Find more in 'Further Readings' slide

Object Orientation

- Abstraction
- Encapsulation
- Composition
- Aggregation

- Class
- Instance
- Polymorphism
- Inheritance

Object-oriented Design Principle

SOLID

- Single Responsibility Principle
- Open-close Principle
- Liskov's Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Architectures and Requirements

Performance

Keep large components; avoid distributed over networks.

Security

Keep Layers. Add authentication mechanism.

Availability

Keep redundant components that can be replaced quickly.

Maintainability

Keep components smaller; separate data component with consumer components.

Further Readings

Highly Recommended

- https://www.yegor256.com/2014/11/20/seven-virtues-of-good-object.html