Software Development
1. Introduction

Software Applications are increasingly used to tackle problems that concern everyday life:

- Automatic Bank tellers
- Airline reservation systems
- Air traffic control systems
- Financial investment systems

In the early generation of computers:

- Programs were fairly small (they were written to solve complex mathematical or algorithmic problems).
- A Program was written by a single programmer.
- The programmer was the user of the program (had the application domain expertise).
- Software development was considered an art.

In the current generation of computers:

- Programs are fairly large (they are increasingly required to satisfy complex user requirements).

  The UNIX Operating System comprises 3.7 million lines of code.

  The NASA Space shuttle software comprises 40 million lines of object code.

- Programs are being developed by teams that collaborate over periods spanning several years.

- Programmers are not the users of the system they develop, and they have no expert knowledge of the application system they develop.

Software development has become an Engineering discipline.
2. Software Engineering

Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software.
(IEEE standard glossary of Software Engineering Terminology)

- *Methods* are defined to guide engineers to perform their duties consistently.
- *Techniques* are used to solve problems.
- *Notations* are used to specify and communicate requirements and design decisions concisely and unambiguously.
- *Tools* are used to manage and automate development tasks.

The construction of software systems is quite different from the construction of physical products:

- *The cost of constructing software is incurred during development and not during production.*
- *Software is logical rather than physical in nature.*
- *Physical products wear out in time and therefore have to be maintained. Software does not wear out.*
- *The need to maintain software is caused by errors detected late or by changing requirements of the user.*
- *Physical systems are often continuous in nature: small changes in the specification lead to small changes in the product. Small changes in the specification of software may lead to changes in the software itself.*
- *In a similar way, small errors in software may have big effects.*

3. The Software Development Process

There are a distinguishable number of activities in the development of software called *phases.*
Analysis

- The analysis phase is concerned with *what* the system does, not with *how* it is done. It avoids implementation decisions.
- The *goal* of the analysis phase is to get a complete description of the problem to be solved and the requirements posed by and the environment in which the system is going to function.
- This phase produces the *Requirements Specification* document.
**Object Oriented Software Technology**

**Design**

System design is the high level strategy for solving the problem and building a solution.

- This phase includes decisions about the organization of the system into subsystems and the allocation of the subsystems to hardware and software components.
- The overall organization of the system is called the *System Architecture*.
- The result of the design phase is the technical or architectural specification.

**Implementation**

- The implementation phase expresses the design from the architecture document in a programming language.
- The code should be a regular translation of the design into the idioms of a particular programming language.
- The result of the implementation phase in an executable program.

**Testing**

Code is produced in pieces, and these pieces must be tested and assembled together before the system can be delivered to the customer.

Actually, each phase of the development process should be involved with testing. The earlier errors are detected, the cheaper their corrections.

At each phase boundary,

- check that the transition from phase I to phase I+1 is correct (*Verification*);
- check that we are still on the right track with regards to fulfilling the user requirements (*Validation*).

Measuring efficiency and then tuning and optimizing the software in the light of these measurements is also considered part of the test phase.
Maintenance

Maintenance is concerned with all the activities that are needed to keep the system operational:

- Corrective Maintenance: the repair of errors.
- Adaptive Maintenance: adapting the software to changes in the environment such as new hardware or a new release of an operating system or a database.
- Perfective maintenance: adapting the software to new or changing user requirements.

4. The Software Life Cycle

In order to control the progress of the software development process, we employ a phased development in which a number of clearly identifiable milestones are established between the start and finish of the project.

In general, milestones identified in a software development project corresponds to points in time at which certain documents become available,

e.g.

- After analysis there is a requirements specification,
- After design there is the architecture specification,
- After implementation there is a set of programs
- After testing there is a test report.

For any given project, we have to choose a life cycle model to manage it with. This involves defining the individual steps and phases, their possible interaction and deliverables.

Some popular life cycle models are
- Waterfall
- Prototyping
- Incremental
- Spiral
4.1 The Waterfall Life Cycle Model

- The most commonly used life cycle model today.
- Completion of one phase is identified by the attainment of a milestone, and has associated with it a set of deliverables.
- The deliverables form the contract for the next phase.
- This approach imposes a linear progression, producing an accumulated set of documents from successive phases with no opportunity to revise previous phases.

- Customer must have a clear idea of the problem.
- Handles requirement changes badly.
- Design decisions are difficult to reverse.
- Long delay before a program is delivered.
4.2 The Prototyping Life Cycle Model

In situations where it is difficult to get and maintain a sufficiently accurate perception of the requirements of the user, develop a prototype.

A prototype is a working model of (possibly part of) a system, that emphasizes certain aspects of it.

To produce the prototype cheaply:
- Use a very high level language, in which an executable version can be produced quickly.
- Implement the minimum functionality.

This relatively inefficient version can be used to test the usability of the proposed system.
4.3 The Incremental Life Cycle

- The functionality of the system is produced and delivered to the customer in small increments.
- Starting from the existing situation we proceed towards the desired situation in a number of steps (employing the waterfall phased approach).

Developing the software in this way avoids the “Big Bang” effect, i.e. For a long time nothing happens, and then suddenly there a completely new situation.

Incremental development can also be used to fight the “Overfunctionality” syndrome. Since the users finds it difficult to formulate their real needs, they tend to demand more than they need or even want.

With the incremental approach, attention is first focused on the essential features. Additional functionality is added only if and when it is needed.

4.4 The Spiral Life Cycle

- During the development of a software system, a number of problems have to be solved.
In solving a problem, the most difficult parts are tackled first, or the parts that have the highest risks (risk with respect to a successful completion of the project).

Following this line of thought Bohem (IEEE computer, May 1988) suggests a spiral model of the software development process, in which each convolution of the spiral gives rise to the following activities:

- Identifying the subproblem that has the highest risk associated with it.
- Finding a solution to that problem.

If obtaining the proper set of user requirements is seen as the area with the highest risk, follow the spiral a few times around to solve this problem (use prototyping).

If starting from a precise requirements specification, follow the spiral once, using the traditional process model with its phases and corresponding milestones as intermediate steps.

Incrementally developing software boils down to tracking the spiral a number of times, once for each increment.
During maintenance, the Errors reported and changing requirements are triggers to track the spiral.

There are no fixed phases in this model. The phases shown in the figure above are merely examples. Management must decide how to structure the project into phases.

**Risk Management**

The spiral model is distinguished from other software cycle models in its explicit consideration of risk.

Risk is simply something that can go wrong.

* e.g.

Adopting a new operating system, that is strong on features, but is still in Beta.
Using an operating system that is low on features.

Risks are a consequence of inadequate information. They are resolved by initiating some actions that discover information that reduces uncertainty.

Using the spiral life cycle:

a) Elaborate objectives such as performance, functionality, etc.
b) Enumerate alternative ways of achieving these objectives and the constraints imposed on each of these alternatives.
c) Assess each alternative against each objective. This will result in the identification of sources of project risks.
d) Evaluate these risks by analysis, prototyping, simulation, etc.

5. System Complexity

The problems we try to solve in software often involve elements of complexity, in which we find a myriad of competing and contradictory requirements.

e.g. The requirements for an airline reservation system

The requirements for a command and control system

The system complexity is compounded by the users inability to give a precise expression to their needs in a form that a developer can understand, and the developers lack of expertise in the domain of the user.

It is essential to develop an early understanding of the problems and the specific requirements. This should be achieved by modeling the requirements with the user.

5.1 Modeling

The common way of expressing requirements today is with large volumes of text.

Such documents are difficult to comprehend, are open to varying interpretation, and too often contain elements that are design rather than essential requirements.

A model is an abstraction of something for the purpose of understanding it before building it.

A model
- omits nonessentials
- is easier to manipulate than the original.

Models are used to:
- Convert the textual requirement into a robust and stable set of specification models.
- Communicate with the customer and establish a common ground understanding.
- Reduce the complexity of the system by separating out a small number of important things to deal with at the time.

5.2 Abstraction

Abstraction is a fundamental human capability that permits us to deal with complexity.
Object Oriented Software Technology

The human mind can cope with only a limited amount of information at one time (about seven, plus or minus two, pieces of information at once).

We deal with complexity of an object by abstracting from it. We ignore its nonessentials, so reducing it to an idealized object.

e.g.

- A map is a model of its territory.
- In order to be useful it must be simpler than the territory it models. If it included every detail, it would have to be the same size as its territory, and would defeat its own purpose.
- A map aids us because it abstracts out those features of the territory that we wish to model.
- A road map models how best to drive from one location to another.

Abstraction
- does not increase the number of things we can comprehend at one time, but
- does imbue them with greater meaning.

e.g. Consider the mental feat involved in memorizing numbers. Perhaps seven digits are all you can memorize. But if you group them and call them a telephone number, you have created a higher, abstract level at which all seven digits are a single entity.

Using this mechanism, you can now memorize perhaps seven telephone numbers, thus increasing your ability to deal with complexity by nearly an order of magnitude.

6. Software Development Methodology

A method is a disciplined process for generating a set of models that describe various aspects of a software system under development, using some well defined notations.

Methods
- instill a discipline into the development of complex software systems;
- define the models that serve as common vehicles for communication among the members of a development team;
- define the milestones needed to measure hence manage progress and risk.

A method(ology) consists of:
- A set of modeling concepts to express the semantics/knowledge of the model.
- A notation for representing the modeling concepts, perhaps in a number of ways.
A good notation enables people to see and understand the important aspects of a model easily.

**Decomposition**

Most methodologies work by decomposition, i.e. “Divide and conquer”.

When designing a complex software system, it is essential to decompose it into smaller parts, where each of which can in turn be refined independently (recall a human can handle few parts rather than all the parts at once).

There are two main types of decomposition: algorithmic and object-oriented.

1. **Algorithmic Decomposition** :

   Structured analysis and design is based on algorithmic decomposition.

   The system is decomposed into parts called *processes*.

   e.g.

   - *Get Order*
   - *Produce Annual budget*
   - *Update Account*

   - Each process constitute a task or operation to be performed by the system.
   - A transaction is performed by invoking one or more processes that are interlinked by data flows.
   - Processes can be decomposed further to add more details.

   Ultimately, processes are mapped into functions (as in the C language) and the data flows are mapped into function calls with parameters.

   The disadvantages of this approach include:

   1. Process decomposition is arbitrary.
   2. Scaling up is difficult
   3. Requirements update has a domino effect on the model.
   4. The boundary between analysis and design is not well defined
   5. Weak binding of data and processes.

   **Example**:
Develop a program that can increment, decrement and reset a numerical value. The numerical value can be displayed in decimal, octal or hexadecimal notation, i.e. base 10, base 8 or base 16.

2. Object-Oriented Decomposition

- Object Oriented Analysis and Design is based on object oriented decomposition.
- The system is decomposed according to the key abstractions, called objects, in the problem domain.
- Transactions are performed by a set of autonomous objects that collaborate, by sending each other messages, to complete the required work.

In this approach each object embodies its own unique behavior, and each one models some object in the real world.
Object Oriented Software Technology
Object Oriented Software Technology

Tutorial

1. Explain why the waterfall model of the software life cycle is not an accurate reflection of software development activities.

2. Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software life cycle model that might be used as a basis for managing the development of the following systems:
   a. A system to control anti-lock braking system
   b. A virtual reality system to support software maintenance
   c. A university accounting system that is intended to replace an existing system.
   d. An interactive system that allows railway passengers to find train times from terminals installed in stations.

1. A university intends to procure an integrated student management system holding all details of registered students including personal information, courses taken and examination marks achieved. The alternative approaches to be adopted are either:
   a. Buy a database management system and develop an in-house system based on this database.
   b. Buy a system from another university and modify it to local requirements.
   c. Join a consortium of other universities, establish a common set of requirements and contract a software house to develop a single system for all of the universities in the consortium.

Identify two possible risks in each of these strategies and suggest techniques for risk resolution which would help in deciding which approach to adopt.

1. Using algorithmic decomposition to develop a model for a calculator. The calculator has 10 decimal numbers, 0 – 9, and can perform the following operations: +, x, /, -, %, C(clear), and CE(clear entry). Also, the calculator provides a memory for holding values. The memory is operated by two buttons, M+ to add a value to memory, M- to subtract a value from memory, M= to display the value held in memory, and Mc to clear the memory.

2. Solve the problem specified in exercise 4 using object-oriented decomposition.