By Suchit Reddi

Software Engineering

The term *Software Engineering* first appeared at the 1968 NATO Software Engineering Conference and was meant to provoke thought regarding what was then called the “software crisis”. It is defined as “An engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use”. It is a systematic approach to designing, developing, operating, and maintaining a software system.

A program is just a combination of source code and object code. Programs become software only if documentation and operating procedure manuals are prepared.

There are two types of Software:

- Generic products
  - Stand-alone systems are sold to any customer who wishes to buy them.
  - The software developer owns the specification of what the software should do, and the developer makes decisions on software change.
  - Examples are PC software such as graphics programs, CAD software, and software for specific markets.
- Customized products
  - Software commissioned by a particular customer to meet their own needs.
  - The customer owns the specification of what the software should do and makes software change decisions.

Computer Science is the study of theory and fundamentals. Software Engineering studies the practicalities of software design, development, and delivery.

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can’t, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

The software process consists of the following steps: Specification, Development (Design and programming), Validation (Checking), and Evolution (Modifications).

The software process model is the abstraction of the software development process. It consists of different views, such as the workflow view, where activities are the human actions that show the input, output, and dependencies. The dataflow view shows the transformation of information into output. The role/action view shows what work is done by people in different roles.

Software costs depend on various factors, like the process used and the type of software being developed. Roughly 60% are development costs, and 40% are testing costs. In custom software, evolution costs often exceed development costs.

CASE: Computer-Aided Software Engineering. Programs that support Requirement analysis, System modeling, Debugging, and Testing.

Attributes of good software are Performance (functional) and Quality (non-functional). Maintainability is the evolution of qualities such as testability. Dependability is reliability, safety, and security. Efficiency is the memory utilization, response, and processing times. Usability is how easy the software is to learn, how efficient it is to use, and how satisfied the users are.

Some development challenges include heterogeneity (compatibility with different computers), Delivery speeds, and trust.

The terms used to calculate software development **productivity** are LOC (Lines of Code) and PM (Person Months). For student programs, the productivity is 2.5-5 KLOC/PM. For SW organizations, it is 100-1000 LOC/PM. The significant difference occurs due to the difference between the type of work done by students and industries.

The *definition* of software (by IEEE) is given as a collection of programs, procedures, rules, and associated documentation and data. The differences between Student and Industrial Strength software are as follows:
➔ In student software: The developer is the user. Bugs are tolerable. Good UI isn't necessary. No documentation is required. It isn't used critically. Reliability and robustness aren't essential. No investment. No need for portability.
➔ In industrial strength software: Others are the users. Bugs aren't tolerated. UI is important. User, organization, and project require documentation. Supports important functions. Reliability and robustness are essential. Heavy investment. Portability is a crucial requirement.

Industrial strength software has better quality, costing 10 times more than student software. In **this course**, software means Industrial strength software. It is expensive. Its productivity is approximately 1000 LOC/PM, costing $3-$15 per LOC. A simple application for a business may have 20K-50K LOC. So, it costs $60K-$2.25 million for the software and $10K-$20K for the hardware.

Driving Factors:
Short delivery times are demanded. Cost and cycle time/schedule are fundamental driving forces. Both can be modeled by productivity, measured in terms of output per unit effort (e.g., LOC per PM). Higher productivity leads to lower costs and cycle time.

Quality is another driving factor that is more challenging to define. ISO standards have six attributes for describing it.
→ Functionality – The capability to provide functions that meet stated and implied needs when software is used.
→ Reliability – The capability to provide failure-free service.
→ Usability – The capability to be understood, learned and used.
→ Efficiency – The capacity to provide appropriate performance relative to the resources used.
→ Maintainability – The capability to be modified to make corrections, improvements, or adaptations.
→ Portability – The capability to adapt to different environments without applying actions/means other than those provided in the product.

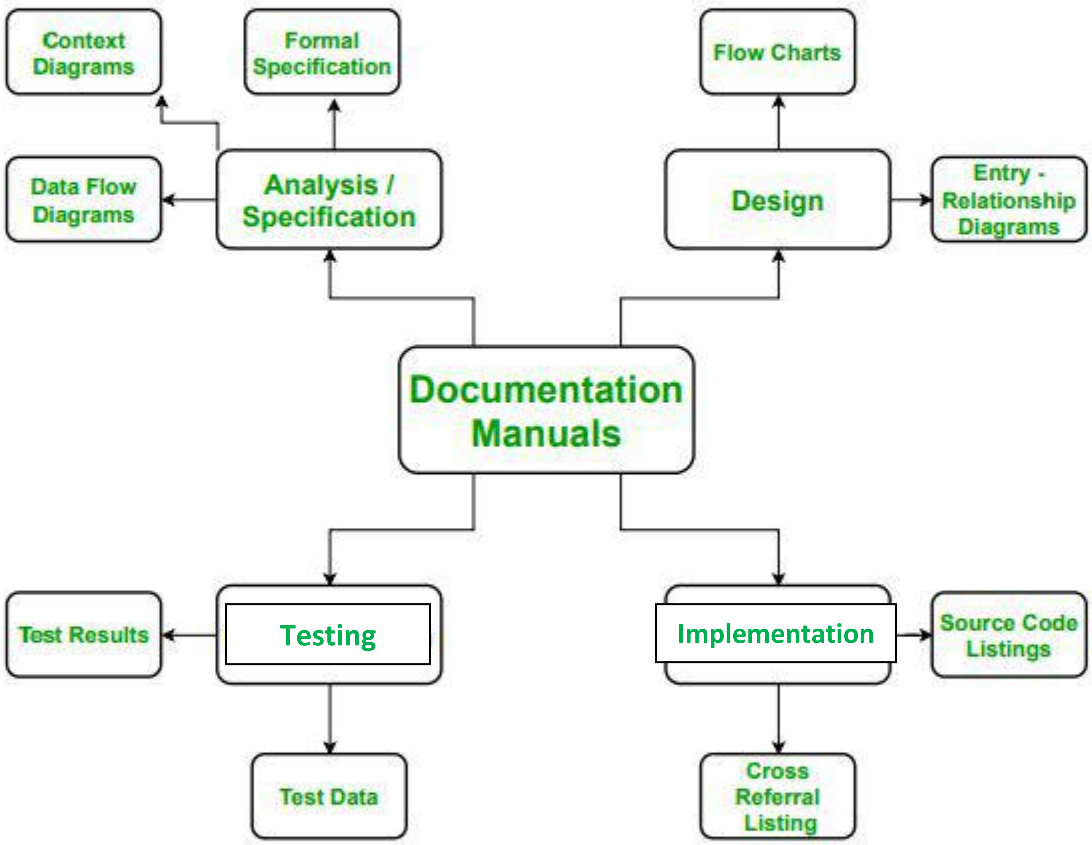
Multiple dimensions mean it is challenging to reduce Quality to a single number. For some projects, reliability is more important than usability, while some are the opposite.

Reliability is generally considered the main Q criterion. It is the probability of failure; it is hard to measure and approximated by no. of defects in software. Quality = No. of defects delivered / Size, where defects delivered is the no. of defects found in operation. In current practices, it is less than 1 def/KLOC.

Once the software is delivered, it enters the **maintenance** phase, where residual errors are fixed – corrective maintenance, upgrades are done – adaptive maintenance. Maintenance can take up more effort than development over the life of the software (it can even be a 20:80 ratio). Hence, maintainability is another critical quality attribute.

The software should support change, as requirements constantly change, and 40% of development may go into implementing changes.

Scalability is another crucial element in software development. Methods for solving small problems do not scale up for large problems. Two dimensions of a project are engineering and project management. For small scale, both can be done informally. For large scale, both must be formalized. The software should be able to handle large problems.



List of Operating Procedure Manuals:
Operating procedures have two parts, user and operational manuals. User manuals have a system overview, Beginner's guide tutorial, and Reference guide. Operational manuals have Installation and System administrator guides.

Some terminologies:

1. Deliverables and Milestones – Deliverables are source code, user manuals, operating procedures, etc. Milestones ascertain the project's status, such as the finalization of specifications and completion of design documentation.
2. Product and Process – The product is delivered to the customer. The process is how software is produced.
3. Productivity and Effort
4. Module and Software Components – An independent deliverable piece of functionality providing access to its services through interfaces.
5. Generic and Customized Software Products – Generic products are developed for everyone. Customized products are designed for particular customers.

Software is developed through stages: Requirement analysis and specification, Design, Coding, and Testing.
Error prevention is preferred over error correction in present times. Modern times require detecting errors to be as close to their point of introduction as possible.

Exploratory style is where a single programmer develops software, and Modern software development processes are done by a team. Now, errors are detected in each phase of development as opposed to only during testing. Coding is only considered a small part of program development, unlike when coding was synonymous with program development. A lot of effort and attention is being paid to requirement specifications. There is a different design phase in modern software development. It has been made systematic, and periodic reviews are carried out during all stages. More care is poured into making consistent and good-quality documentation, which makes management more effortless. Projects are being appropriately planned, with an estimation of requirements, scheduling tasks, and monitoring mechanisms. They also use CASE (Computer-Aided Software Engineering) tools.

Software Life Cycle Model

It is a descriptive and diagrammatic model of the software life cycle that identifies all the activities undertaken during product development. It divides the life cycle into phases and establishes an order among different activities. It helps identify inconsistencies, redundancies, and omissions in the development process. It helps in developing a common understanding of activities among software developers.

Why is Life Cycle Model important?

In an exploratory model where a single programmer develops the software, he has the freedom to decide his steps like Code → Test → Design, Code → Design → Test → Change code, or Specify → Code → Design → Test.

When a team develops software, there must be an understanding among the members as to when to do what. Otherwise, the project would be a failure. In a life cycle model, entry and exit criteria are set for every phase. Only when they are satisfied is a phase considered to be complete.

The ultimate objective of software engineering is to produce good quality, maintainable software within a reasonable time frame and at an affordable cost. The software life cycle starts from the concept of exploration and ends at the retirement of the software. The cycle includes a requirement phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and finally, retirement phase.

The software life cycle model is an abstraction of a software life cycle, similar to classes and objects. A software life cycle model is often called a software development life cycle (SDLC).

Software Development Life Cycles

I. Build and Fix Model

It is a simple two-phase model and not well-defined. The first phase is to write code, and the next is to fix it. Here, fixing may be error correction or the addition of additional functionality. This model is used when a product is constructed without specifications. So, the developer builds a product that is reworked as many times as necessary to satisfy the client. This may work well on small programming exercises of 100 or 200 lines long. It is unsatisfactory for software of any reasonable cost, as code soon becomes unfixable.

II. The Waterfall Model

It is the most familiar model having five phases. These phases always occur in order and do not overlap; each phase should be completed before the next one begins.

Requirement Analysis & Specification	→	Design	→	Implementation and Unit Testing	→	Integration and System Testing	→	Maintenance

a) Requirement Analysis and Specification Phase:
   Understanding the customer's requirements and documenting them properly. This is done together with the customer. The output of this phase is a document known as Software Requirement Specification (SRS), which is written in natural language describing what the system will do without explaining how it will be done.
b) Design Phase:

Transforming the requirements specification to be implementable in some programming language. Software architecture is defined, and high-level design work is performed. This work is documented as Software Design Description (SDD).

c) Implementation and Unit Testing Phase:
Design is implemented. The code is examined and modified during testing. Small modules are tested in isolation.

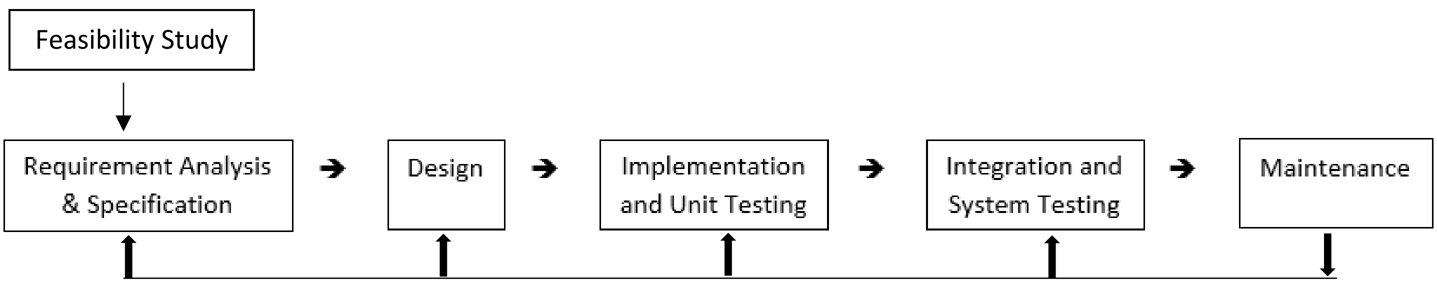
d) Integration and System Testing Phase:
Testing how the integration of all modules is working together. A critical phase, when done correctly, will result in high-quality software, satisfied users, low maintenance costs, and more accurate and reliable results. It consumes one-third to half of the cost of a typical development project.

e) Operation and Maintenance Phase:
Error correction, capability enhancement, removal of obsolete capabilities, and optimization are done after the software is delivered to the customer's site, installed, and operational. This phase preserves the value of software over time.

Problems of the Waterfall Model:
Defining all the requirements at the beginning of a project is complex, and it is not suitable for accommodating any change. It takes time to develop a working version of the system. It doesn't scale up well to large projects, and real projects are rarely sequential.

III. Iterative Waterfall Model

The main difference between the standard and iterative waterfall models is the feedback paths from each phase to its preceding phases. This will make the model more usable in practical software development projects.



a) Feasibility Study:

b)