Chapter 2

Introduction to Computing and Software Engineering

2.1 Computer Science and IT as Disciplines

2.1.1 What is Computer Science?

- ❖ NOT about coding or hardware or software!
- ❖ **Computer Science is the study of computers (??)**

This leaves aside the theoretical work in CS, which does not make use of real computers, but of formal models of computers

A lot of work in CS is done with pen and paper!

Actually, the early work in CS took place before the development of the first computer

Computer Science is no more about computers than astronomy is about telescopes, biology is about microscopes, or chemistry is about test tubes. Science is not about tools. It is about how we use them, and what we find out we can do.

- ❖ **Computer Science is the study of how to write computer programs (programming) (??)**

  Programming is a big part of CS…but it is not the most important part.

- ❖ **Computer Science is the study of the uses and applications of computers and software (??)**

  Learning to use software packages is no more a part of CS than driver's education is part of automotive engineering.

  CS is responsible for building and designing software.

- ❖ **The study of algorithms**:

a. Their formal properties:
   - ✓ correctness, limits
   - ✓ efficiency/cost

b. Their hardware realizations:
   - ✓ computer design

c. Their linguistic realizations

        - ✓ programming languages

d. Their applications
    - ✓ network design, ocean modeling, bioinformatics, ...
    - ✓ a well-defined procedure that allows an agent to solve a problem.

- ❖ Computer Science is about PROBLEM SOLVING
- ❖ Computer Science is about DEVELOPING ALGORITHMS to solve complex problems
- ❖ Computers are merely tools for solving problems!
- ❖ Computer Science is a science concerned with information i.e. representation, storage, manipulation or processing and presentation of information. Like any other science, which uses some devices for the practical aspect, computer Science uses a special device called COMPUTER.
- ❖ is the systematic study of the feasibility, structure, expression, and mechanization of the methodical processes (or algorithms) that underlie the acquisition, representation, processing, storage, communication of, and access to information, whether such information is encoded in bits and bytes in a computer memory or transcribed in genes and protein structures in a human cell. The fundamental question underlying all of computing is: what computational processes can be efficiently automated and implemented?
- ❖ Well-developed, organized approaches to solving complex problems
- ❖ Test of a good algorithm:
    - o Does the algorithm solve the stated problem?
    - o Is the algorithm well-defined?
    - o Does the algorithm produce an output?
    - o Does the algorithm end in a reasonable length of time?

Computer science has different fields of specialization or sub-disciplines like other sciences. These are sub-disciplines of computer science:

Software Engineering: It is concerned about the development of better quality software by applying scientific & basic engineering principles. Software engineering is concerned with theories, methods and tools for professional software development.

Computer Engineering (Architecture): deals with studying, analyzing and designing of computer hardware (organization and interconnection of computer system components) and its working principle.
Automata theory: Is the study of machines or devices which accept a certain inputs such that the output or at least the probabilities of outputs are determined by the input.
Formal Language Theory: Embraces the study of programs of programming languages, which is important for the understanding, and construction of compilers.
Complexity theory: Concerned with the study and analysis of algorithms, which helps in measuring the efficiency of the algorithms.
Database Architecture: Involves the study and design of efficient methods for information storage, process & retrieval.
Artificial Intelligence: Is concerned with means by which computers may perform tasks that would be characterized as intelligent if performed by human beings.

2.1.2 What is IT? Components of IT, Functions and application of IT, Data and Information

A. **IT**

Any tool for manipulating data, information

- electronic: computer software and hardware - our focus

is the hardware and software that make information systems possible.

: A term used to refer to a wide variety of items and abilities used in the creation, storage, and dispersal of data and information. Its three main components are **computers, communications networks, and know-how**.

B. Components of IT

1. Computers

2. Communications networks

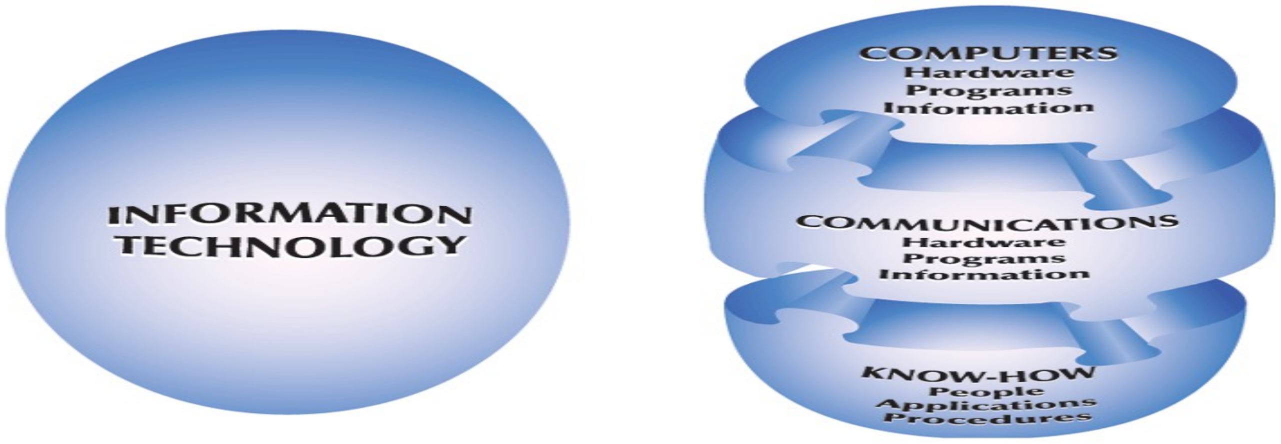
- A set of locations, or nodes, consisting of hardware, programs, and information linked together as a system that transmits and receives data and information.

3. **Know-how**

- The capability to do something well.

- Information technology know-how consists of:
  - Familiarity with the tools of IT; including the Internet
  - Possession of the skills needed to use these tools
  - An understanding of when to use IT to solve a problem or create an opportunity
- **System*:*** A set of components that interact to accomplish a purpose.
- **Information System:** A business information system designed to produce the information needed for successful management of a structured problem, process, department, or business.

Figure 1.4 The Forces of Information Technology



C. The Principles of Information Technology

1. **Helping People**

2. **Solving Problems**
   - ***Problem:*** A perceived difference between an existing condition and a desired condition.
   - ***Problem Solving:*** The process of recognizing a problem, identifying alternatives for solving it, and successfully implementing the chosen solution**.**

3. **Improving Our Lives**

D. Data & Information

Data: Raw facts, figures, and details.

The source of data can be primary or secondary based on its origin.

Information: An organized, meaningful, and useful interpretation of data.

Should be delivered to the right target at right time.

<u>Remark</u>: The one information/ primary for someone can be data/ secondary for others.

Knowledge: An awareness and understanding of a set of information and how that information can be put to the best use.

Data - Information – Knowledge – Wisdom

Fig 2-2: Data, Information, Knowledge and Occupations

- **Knowledge**
  - Created based on the information/data or own expertise
  - Answers How?
  - Is dynamic & context based
  - Helps in decision making
- **Wisdom**
  - Gives detailed understanding
  - Arriving at the judgment
  - Helps finalize future decisions/actions

2.2 Software Engineering as a Discipline

❖ Systematic approach for developing software!

❖ Methods and techniques to develop and maintain quality software to solve problems. !

❖ Study of the principles and methodologies for developing and maintaining software systems.

- **Issues addressed by Software Engineering**
  - ✓ How do we ensure the quality of the software that we produce?!
  - ✓ How do we meet growing demand and still maintain budget control?!
  - ✓ How do we avoid disastrous time delays?!
- **Size of programs continues to grow!**
  - ✓ **Trivial:** 1 month, 1 programmer, 500 LOC,!

    Intro programming assignments!
  - ✓ **Very small:** 4 months, 1 programmer, 2000 LOC!

    Course project!
  - ✓ **Small:** 2 years, 3 programmers, 50K LOC!

    Nuclear power plant, pace maker!
  - ✓ **Medium:** 3 years, 10s of programmers, 100K LOC!

    Optimizing compiler!

2.2.1 Software

Software is defined as a collection of programs, procedures, rules, data and associated documentation. The s/w is developed keeping in mind certain h/w and operating system consideration commonly known as **platform** and engineering means systematic procedure to develop software. Some of the software characteristics are, it can be engineer or developed and second thing is software is complex in nature.

Software cannot be built fast enough to keep up with!

- ✓ H/W advances!
- ✓ Rising expectations!
- ✓ Feature explosion!: Increasing need for high reliability software

Important of software are due to much reason as it is used in:

i) Business decision making

eg.- accounting s/w, billing s/w

ii)For scientific research & engineering problem solving.

eg.- weather forecasting system, space research s/w

iii)It is embedded in multifunctional systems such as medical, telecom entertainment etc.
Eg-s/w for medical patient automation, s/w of GSM/CDMA service provides.

Software Quality

Several quality factors associated with software quality are as following:

Portability: A software product is said to be portable, if it can be easily made to work in different operating system environments, in different machines, with other software products, etc.

Usability: A software product has good usability, if different categories of users (i.e. both expert and novice users) can easily invoke the functions of the product.

Reusability: A software product has good reusability, if different modules of the product can easily be reused to develop new products.

Correctness: A software product is correct, if different requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable, if errors can be easily corrected, new functions can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Types of software:-

Computer s/w is can be divided into two types.

a) **System s/w**

. System s/w includes the operating system & all the utilities to enable the computer to run. Eg. Windows operating system

b) **Application s/w**

Application s/w consists of programs to perform user oriented tasks. eg -word processor, database management. Application s/w sits about the system s/w because it needs help of the system s/w to run.

Program Vs Software Product

Programs: Set of instructions related to each other.

Products: Collection of program designed for specific task.

Programs are defined by individuals for their personal use.

A sum product is usually developed by a group of engineers working as a team.

Program	**Product**
Usually small size …………………………..	Usually large size.
Single user……………………………………	Large no of users.
Single developer ……………………………	Team of developer.

Lack proper documentation	Good documentation support.
Adhoc development	Systematic development.
Lack of UI	Good UI
Have limited functionality	Exhibit more functionality

Types of software products

Generic products: This type of software product are developed by an organization and sold on open market to any customer eg. System software, application software
Customized (or bespoke) products: This type of software products are developed by a software contractor and especially for a customer.
Embedded Product: Combination of both hardware and software

2.2.2 Why is software engineering needed?

The economies of **ALL** developed nations are dependent on software. More and more systems are software controlled

Software engineering is concerned with theories, methods and tools for professional software development. Software engineering expenditure represents a significant fraction of budget in all developed countries. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

All software engineers use tools, and they have done so since the days of the first assemblers. Some people use stand-alone tools, while others use integrated collections of tools, called environments. Over time, the number and variety of tools has grown tremendously. They range from traditional tools like editors, compilers and debuggers, to tools that aid in requirements gathering, design, building GUIs, generating queries, defining messages, architecting systems and connecting components, testing, version control and configuration management, administering databases, reengineering, reverse engineering, analysis, program visualization, and metrics gathering, to full-scale, process centered and software engineering environments that cover the entire lifecycle, or at least significant portions of it. Indeed, modern software engineering cannot be accomplished without reasonable tool support.

The role of computers, their power, and their variety, are increasing at a dramatic pace. Competition is keen throughout the computer industry, and time to market often determines success. There is, therefore, mounting pressure to produce software quickly and at reasonable cost. This usually involves some mix of writing new software and finding, adapting and integrating existing software. Tool and environment support can have a dramatic effect on how quickly this can be done, on how much it will cost, and on the quality of the result. They often determine whether it can be done at all, within realistic economic and other constraints, such as safety and reliability. Software engineering tools and environments are therefore becoming increasingly important enablers, as the demands for software, and its complexity, grow beyond anything that was imagined at the inception of this field just a few decades ago.

- ✓ To predict time, effort, and cost!
- ✓ To improve software quality!
- ✓ To improve maintainability!
- ✓ To meet increasing demands!
- ✓ To lower software costs!
- ✓ To successfully build large, complex software systems!
- ✓ To facilitate group effort in developing software!

It is clear that software is important:

Critical applications: Software has found its way into many applications where failure has serious consequences. For example: car braking and steering, process control, safety systems in hazardous processes, civilian avionics, communication networks and telephony, . . .

Competitiveness: Software is seen as the key to competitiveness in many areas. In retail, finance and entertainment, e-commerce is seen as a critical development; in many other areas of economic activity good software is seen as a key element in the competitiveness of firms.

Economically: The estimated value of systems containing embedded software will exceed 1012 dollars in the next few years. This is only one market for software, there are many others.

This does not mean Software Engineering is important at the moment. Critics point to well-publicized failures to supply well-engineered software systems by suppliers who do attempt to use best practice.

These well-publicized failures mask many projects that are successful and are delivered on time and on price. We can argue that the aims of Software Engineering are important and in some contexts those aims can be realised.

Software Engineering

Application of engineering for development of software is known as software engineering. It is the systematic, innovative technique and cost effective approach to develop software. And person involved in developing software product is called software engineer. S/w engineer is a licensed professional engineer who is skilled in engineering discipline.

Qualities / Skills possessed by a good software engineer:

1. **General Skill** (Analytical skill, Problem solving skill, Group work skill)
2. **Programming Skill** (Programming language , Data structure , Algorithm , Tools(Compiler, Debugger))
3. **Communication skill** (Verbal , Written, Presentation)
4. **Design Skill** (s/w engineer must be familiar with several application domain)

IEEE definition of Software engineering: A systematic, disciplined and quantifiable approach to the development, operation, maintenance and refinement of software.

Factor in emergence of software engineering:

1. People who are developing software were consistently wrong in their estimation of time, effort and cost.
2. Reliability and maintainability had difficulty of achieved
3. Fixing bugs in delivered software was difficult
4. Delivered software frequently didn't work
5. Increased cost of software maintenance
6. Increased demand of software
7. Increased demand for large and more complex software system
8. Increasing size of software

2.2.3 SOFTWARE ENGINEERING PRINCIPLES

Software engineering is a layered technology. The bedrock that supports software engineering is a quality focus. The foundation for software engineering is the *process* layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework for a set of *key process areas* that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work product (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

Software engineering *methods* provide the technical how-to's for building software that encompasses requirements analysis, design, program construction, testing, and support.

Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Software engineering *tools* provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called **computer-aided software engineering (CASE),** is established. CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

2.2.4 SOFTWARE CHARACTERISICS

Characteristics of s/w can be easily distinguished as of from the h/w. For hardware products, it can be observed that failure rate is initially high but decreases as the faulty components are identified and removed. The system then enters its useful life. After some time (called product life time) the components wear out, and the failure rate increases. This gives the plot of hardware reliability over time their characteristic is like "bath tub" shape.

On the other hand, for software the failure rate is at its highest during integration and test. As the system is tested, more and more errors are identified and removed resulting in reduced failure rate. This error removal continues at a slower pace during the useful life of the product. As the software becomes obsolete no error corrections occurs and the failure rate remains unchanged.
Therefore after analyzing the facts we can write the key characteristics as follows:-
a) Most s/w s are custom built rather than assembled from existing components.
b) s/w is developed or engineered not manufactured c) s/w is flexible d) s/w does not wear out.

2.2.5 CAUSES AND SOLUTION FOR S/W CRISIS

Software engineering appears to be among the few options available to tackle the present software crisis.
Let us explain the present software crisis in simple words, by considering the following.
The expenses that organizations all around the world are incurring on software purchases compared to those on hardware purchases have been showing a worrying trend over the years
Organizations are spending larger and larger portions of their budget on software not only are the software products turning out to be more expensive than hardware, but also presented lots of other problems to the customers such as: software products are difficult to alter, debug, and enhance; use resources non optimally; often fail to meet the user requirements; are far from being reliable; frequently crash; and are often delivered late.
Due to ineffective development of the product characterized by inefficient resource usage and time and cost over-runs. Other factors are **larger problem sizes, lack of adequate training in software engineering, increasing skill shortage, and low productivity improvements.**

S/W crisis from programmer point of view:

i) Problem of compatibility.
ii) Problem of Portability.
iii) Proclaiming documentation.
iv) Problem of pirated s/w.
iv) Problem in co-ordination of work of different people.
v) Problem of proper maintenance.

S/W crisis from user point of view:

i) s/w cost is very high.
ii) Price of h/w grows down.
iii) Lack of development specification.
iv) Problem of different s/w versions.
iv) Problem of bugs or errors.

S/W Application: S/W applications can be grouped into 8 different areas as given below.

System s/w, Real-time , Embeded , Business , PC , AI , Web-based, Engineering & scientific s/w

S/W engineering processes:-

Process: The process is a series of states that involves activities, constraints, resources that produce an intended output of some kind. Or it is a state that takes some input and produced output

Software process: A s/w process is a related set of activities & sub processes that are involved in developing & involving a s/w system. There are four fundamental process activities carried out by s/w engineering while executing the s/w process are:

i) **S/W Specification**: The functionality of s/w & constraints on its operation must be defined.

ii) S/W Development: The s/w that mixes the specification must be produced.

iii) S/W Validation: The s/w must be validated to ensure that it performs desired customer activities.

iv) S/W Evolution: The s/w must evolve to meet changing customer needs with time.

The s/w industry considers the entire s/w development task as a process according to Booch & Rumbaugh. According to them a process defines who is doing what, when & how to reach a certain goal.

Generic quality attributes in a software process:

1. Understandability
2. Visibility
3. Reliability
4. Robustness
5. Adaptability
6. Rapidity
7. Maintainability
8. Supportability

Software Project:

1. A project is a temporary endeavor undertaken to create a unique product.
2. Temporary means every project has a definite beginning and a definite end.
3. Unique means the product must be different in some ways from all similar products.

❖ **Why is software development difficult?**

- **Personnel characteristics!**
  - Ability!, Prior experience, Communication skills, Team cooperation, Training!
- **Facilities and resources!**
  - Identification!, Acquisition!
- **Management issues!**
  - Realistic goals!, Cost estimation!, Scheduling!, Resource allocation!
  - Quality Assurance!, Version Control!, Contracts,…

2.2.6 Frequently Asked Questions about Software Engineering

1. **What is software?**

Computer programs and associated documentation such as requirements, design models and user manuals. Software products may be developed for a particular customer or may be developed for a general market. Software products may be:

1. **Generic**: Developed to be sold to a range of different customers e.g. PC software such as Excel.
2. **Custom (Bespoke):** Developed for a single customer according to their specification.

New software can be created by developing new programs, configuring generic software systems or reusing existing software.

2. **What is the difference between software engineering and computer science?**

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software. Computer science theories are currently insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).

3. **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 process concerned with developing the software infrastructure, control, applications and databases in the system. **System engineers** are involved in system specification, architectural design, integration and deployment**.**

4. **What is a software process?**

A set of activities whose goal is the development or evolution of software. Generic activities in all software processes are:

Specification: What the system should do and its development constraints
Development: Production of the software system
Validation: Checking that the software is what the customer wants
Evolution: Changing the software in response to changing demands.

5. **What is a software process model?**

A simplified representation of a software process, presented from a specific perspective.Examples of process perspectives are: **Workflow Perspective**: Sequence of activities, **Data-Flow Perspective**: Information flow. **Role/Action Perspective**: Who does what?

Generic process models

- ✓ Waterfall
- ✓ Iterative development
- ✓ Component-based software engineering
- ✓ Spiral

6. **What are the costs of software engineering?**

Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs. Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability. Distribution of costs depends on the development model that is used. Figure 2-3 shows the Cost Distribution of models.

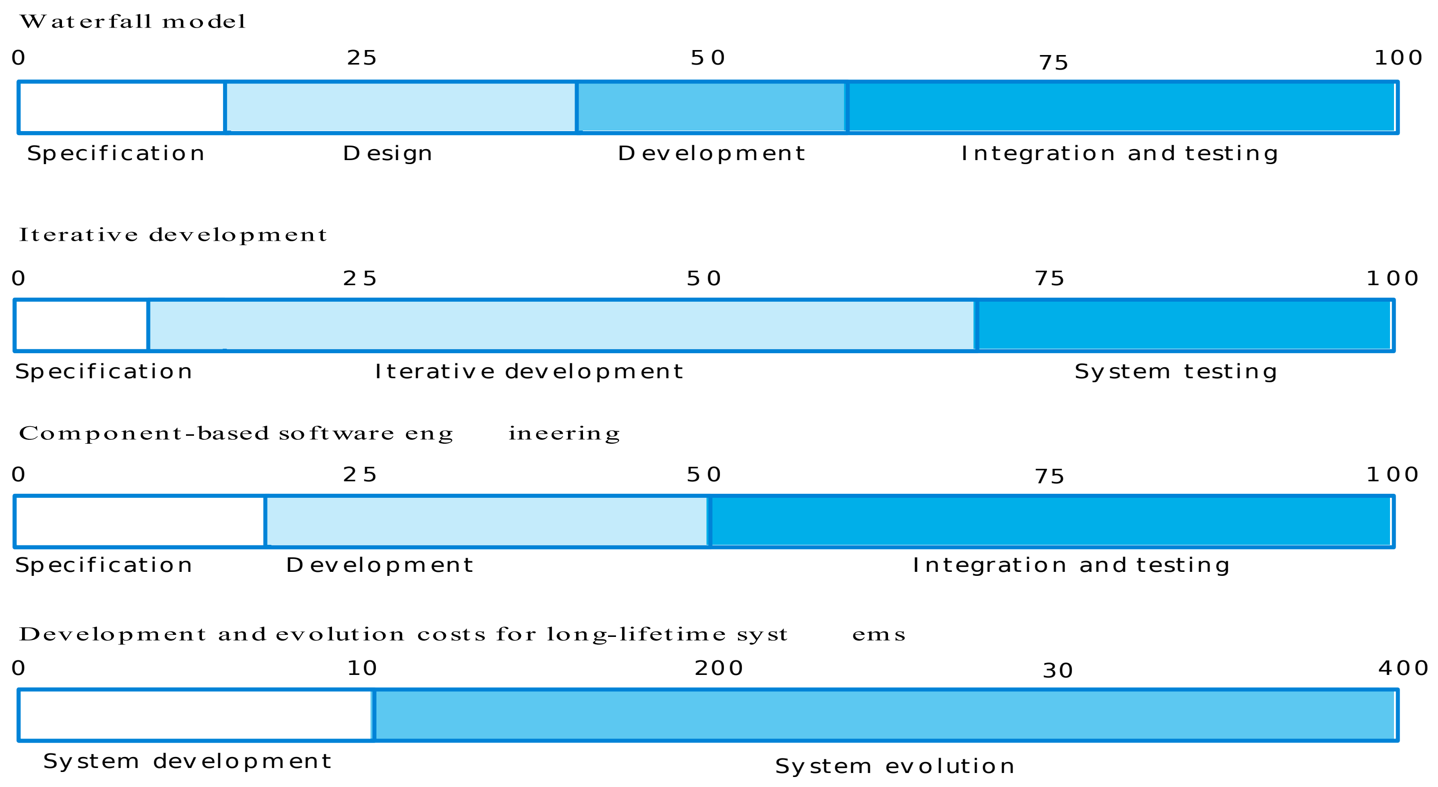


Figure 2-3: Cost Distribution

7. **What are software engineering Methods?**

Structured approaches to software development which include system models, notations, rules, design advice and process guidance.

Model Descriptions: Descriptions of graphical models which should be produced
Rules: Constraints applied to system models
Recommendations: Advice on good design practice
Process guidance: What activities to follow.

8. **What is CASE (Computer-Aided Software Engineering)**

Software systems that are intended to provide automated support for software process activities. CASE systems are often used for method support. Support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run. There are 2 categories.

Upper-CASE: Tools to support the early process activities of requirements and design such as Use Case,…

Lower-CASE: Tools to support later activities such as programming, debugging and testing like Java, C++,.

9. **What are the attributes of good software?**

The software should deliver the required:

A) functionality and performance to the user
B) Maintainability: Software must evolve to meet changing needs.
C) Dependability: Software must be trustworthy.
D) Efficiency: Software should not make wasteful use of system resources.
E) Acceptability: Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.

10. **What are the key challenges facing software engineering?**

Heterogeneity, delivery and trust are the major challenges.

Heterogeneity: Developing techniques for building software that can cope with heterogeneous platforms and execution environments

Delivery: Developing techniques that lead to faster delivery of software

Trust: Developing techniques that demonstrate that software can be trusted by its users.

2.2.7 Professional and Ethical Responsibility

Software engineering involves wider responsibilities than simply the application of technical skills. Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals. Ethical behavior is more than simply upholding the law.

Issues of Professional Responsibility

Confidentiality: Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

Competence: Engineers should not misrepresent their level of competence. They should not knowingly accept work which is out with their competence.

Intellectual property rights: Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.

Computer misuse: Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses). In general, you should respect the Code of Ethics (read it).

2.2.8 Software Engineering Statistics

- ✓ The U.S. spends $2.3 trillion on projects every year, an amount equal to one-quarter of the nation's gross domestic product.
- ✓ The world as a whole spends nearly $10 trillion of its$40.7 trillion gross product on projects of all kinds.
- ✓ More than sixteen million people regard project management as their profession; on average, a project manager earns more than $82,000 per year.
- ✓ More than half a million new information technology (IT) application development projects were initiated during 2001, up from 300,000 in 2000.
- ✓ Famous business authors and consultants are stressing the importance of project management. As Tom Peters writes in his book, Reinventing Work: the Project 50, "To win today, you must master the art of the project!"

2.2.9 **Success and Failure Factors**

Why Software is delivered Late?

- ✓ An unrealistic deadline
- ✓ Changing but unpredicted customer requirements
- ✓ Underestimation of efforts needed
- ✓ Risks not considered at the project start
- ✓ Unforeseen technical difficulties
- ✓ Unforeseen human difficulties
- ✓ Miscommunication among project staff
- ✓ Failure to recognize that project is falling behind schedule

Why IT Projects Fail?

"The project team, the suppliers, the customers and others stakeholders can all provide a source of failure, but the most reasons for project failure are rooted in the project management process itself and the aligning of IT with organizational cultures" (Tilmann and Weinberger, 2004).

Based on a research carried out by the Coverdale Organization (Cushing, 2002), respondents identified:

- ✓ Estimation mistakes,
- ✓ Unclear project goals and objectives, and
- ✓ Project objectives changing during the project as key factors in project failures.

The following list the primary causes for the failure of complex IT projects:

- ✓ Poor planning
- ✓ Unclear goals and objectives
- ✓ Objectives changing during the project
- ✓ Unrealistic time or resource estimates
- ✓ Lack of executive support and user involvement
- ✓ Failure to communicate and act as a team
- ✓ Inappropriate skills

In specific terms - Reasons why Projects Fail:

Don't manage risks: "Management must actively attack a project risks, otherwise they will actively attack you" (Gilb 1988)

- ✓ Building the wrong thing
- ✓ Don't involve users at all stages

Technology Fails: Don't use system architecture to reduce impact of technology failure

IT Cortex

The statistics presented here all converge to establish that:

- ✓ IT project is more likely to be unsuccessful than successful
- ✓ About 1 out of 5 IT projects is likely to bring full satisfaction
- ✓ The larger the project the more likely the failure

This raises, of course, a series of questions:

- ✓ Would an organization be better off without undertaking IT projects?
- ✓ Does the attention shown by top management for strategic projects reflect the actual stakes?
- ✓ What will increase the chances of success?
- ✓ What edge does my project have with respect to those on the casualty list?

1.7 Software Crisis

Crisis: is a turning point in the course of anything.

The software crisis:

- ✓ It becomes clear in the mid-60s that:
- ✓ Software systems were developed far too slowly.

- ✓ Many software projects failed
- ✓ Most projects were delivered late and cost far more than was budgeted.
- ✓ Delivered systems were often of very low quality

❖ The term "software crisis" was coined by some attendees at the first NATO Software Engineering Conference in 1968 at Garmisch, Germany.

• The aim of this conference was already to tame the “software crisis”. Since this era, software development seems to have been continuously in crisis and we still debate today if we are
software engineers or software craftsmen. Many reports exist on software projects have difficulties to respect initial budget and schedule, which is confirmed by real life experience. Multiple approaches have been devised and promoted to change this situation.

• Do you see a debate about the fact that tunnel or road building is not engineering? Have you read about the” building crisis”? I don’t.

• Construction engineering has been presented in the past as a model for software engineering. If we could define precisely the requirements like the construction industry blueprints, we would be able to achieve the same reliable results.

• This was the foundation of the methodologies crafted in the 20th century (SSADM, Information Engineering, Merise, RUP, etc.) that rely on a complete initial analysis and architecture of software projects.

• It is true that you don’t see a lot of abandoned half-built bridges or roads as you can witness canceled software projects. However, you can see examples where the final price is really higher than the initial estimation. The main tunnel through the Swiss Alps still under construction is expected to cost the double of its initial budget…. and there are 8 years left to make this worse. We can then wonder if construction engineering should really be an example for software development.

• The “software crisis” of the 1960s and 1970s was so called because of a string of high profile software project failures: over budget, overdue, etc.

• The crisis arose in part because the greater power available in computers meant that larger software projects were tackled with techniques developed on much smaller projects.

• Techniques were needed for software project management.
Good project management cannot guarantee success, but poor management on significant projects always leads to failure.

❖ Software projects have several properties that make them very different to other kinds of engineering project.

(1) The product is intangible.
It's hard to claim a bridge is 90% complete if there is not 90% of the bridge there.
It is easy to claim that a software project is 90% complete, even if there are no visible outcomes.
(2) We don't have much experience.
Software engineering is a new discipline, and so we simply don't have much understanding of how to engineer large scale software projects.
(3) Large software projects are often "bespoke" (made to order).
Most large software systems are one-off, with experience gained in one project being of little help in another.
(4) The technology changes very quickly: Most large software projects employ new technology
1.8 Software Project Failure Costs Billions
Better Estimation & Planning Can Help: There are so many studies attempting to quantify the cost of software failures. They don't agree on percentages but they generally agree that the number is at least 50 to 80 billion dollar range annually.
Standish Chaos Reports: Standish is probably the most referenced. They define success as projects on budget, of cost, and with expected functionality. There are several updates to the Standish "Chaos" reports.
The 2004 report shows:
- ✓ Successful Projects: 29%
- ✓ Canceled projects cost $55 Billion Annually?
- ✓ Challenged Projects: 53%
- ✓ Failed Projects: 18%

Standish Findings By Year Update for 2009
Succeeded: 32%, Failed: 24%, Challenged: 44%
Most projects cost more than they return, Mercer Consulting:
When the true costs are added up, as many as 80% of technology projects actually cost more than they return. It is not done intentionally but the costs are always underestimated and the benefits are always overestimated.
Dosani, 2001 Oxford University Regarding IT Project Success (Saur & Cuthbertson, 2003)
Successful: 16%, Challenged: 74%, Abandoned: 10%
British Computer Society: Successful: 16%, Failure Costs Tens of Billions of British Pounds in the European Union National Institute of Standards and Technology

(NIST), Software defects cost nearly $60 Billion Annually, 80% of development costs involve identifying and correcting defects

Tata Consultancy 2007

62% of organizations experienced IT projects that failed to meet their schedules
49% suffered from budget overruns, 47% had higher-than-expected maintenance costs, 41% failed to deliver the expected business value and ROI, 33% file to perform against expectations

Note: For all the above problems: late, failure, cost and quality issues strictly following software engineering principles has ultimate solutions to develop large software.