1

Introduction to Software Engineering

The nature and complexity of software have changed significantly in the last 30 years. In the 1970s, applications ran on a single processor, produced alphanumeric output, and received their input from a linear source. Today's applications are for more complex; typically have graphical user interface and client– server architecture. They frequently run on two or more processors, under different operating system, and on geographically distributed mechanics.

Rarely in history has a field of endeavor evolved as rapidly as software development. The struggle to stay, abreast of new technology deal with accumulated development backlogs, they can, just to stay in place. The initial concept of one "guru",indispensable to a project and hostage to its continued maintenance has changed. The software Engineering (SEI) and group of "gurus" advise us to improve our development process. Improvement means "ready to change". Not every member of an organization feels the need to change. It is too easy to dismiss process improvement efforts as just the Latest management fad. Therein we the seeds of conflict, as some member of a team embrace new ways of working, while other, matter "overing dead body". Therefore, there is an urgent need to adopt software engineering concepts, strategies, practices to devoid conflict, add to improve the software development process order to deliver good quality maintainable software in time and within budget.

1.1. INTRODUCTION TO SOFTWARE

Software is described by its capabilities. The capabilities relate to the functions it executes, the features it provides and the facilities it offers. Software written for sales order processing would have different functions to process different types of sales orders from different market segments. The features, for example, would be to handle multi-currency computing, updating of product, sales and tax status in MIS reports and books of accounts. The facilities could be printing of sales orders, e-mail to customers, reports and advice to the stores department to dispatch the goods. The facilities and features could be optional and based on customer choice.

The software is developed keeping in mind certain hardware and operating system considerations, known as platform. Hence, software is described alongwith its capabilities and the platform specifications that are required to run it.

1.1.1. Definition of Software

Software is a set of instructions to acquire inputs and to manipulate them to produce the desired output in terms of functions and performance as determined by the user of the software. It also includes a set of documents, such as the software manual, meant for users to understand the software system. Today's software comprises the Source Code, Executables, Design Documents, Operations and System Manuals and Installation and Implementation Manuals.

Software is:

(*i*) Instructions (computer programs) that when executed provide desired function and performance.

(*ii*) Data structures that enable the programs to adequately manipulate information.

(*iii*) Documents that describe the operation and use of the programs.

OR

The term software refers to the set of computer programs, procedures, and associated documents (flowcharts, manuals, etc.) that describe the programs and how they are to be used.

To be precise, software means a collection of programs whose objective is to enhance the capabilities of the hardware.

OR

Definition of Software given by IEEE

Software is the collection of computer programs, procedure rules and associated documentation and data.

1.1.2. Importance of Software

Computer software has become a driving force.

- It is engine that drives business decision-making.
- It serves as the basis for modern scientific investigation and engineering problem solving.
- It is embedded in all kinds of systems like transportation, medical, telecommunications, military, industrial processes, entertainment, office products etc.

It is important as it affects nearly every aspect of our lives and has become pervasive in our commerce, our culture and our every day activities software impact on our society and culture is significant. As software importance grows, the software community continually attempts to develop technologies that will make it easier, faster and less expensive to build high-quality computer programs.

1.2. TYPES OF SOFTWARE

Computer software is often divided into two categories:

1. **System software.** This software includes operating system and all utilities that enable the computer to function.

2. **Application software.** These consist of programs that do real work for users. For example, word processors, spread sheets, and database management systems fall under the category of applications software.

System software are low level programs that interact with the computer at a very basic level there include operating system, computers utilities for managing resources. In contrast, applications software includes database programs, word processors, and spread sheets.

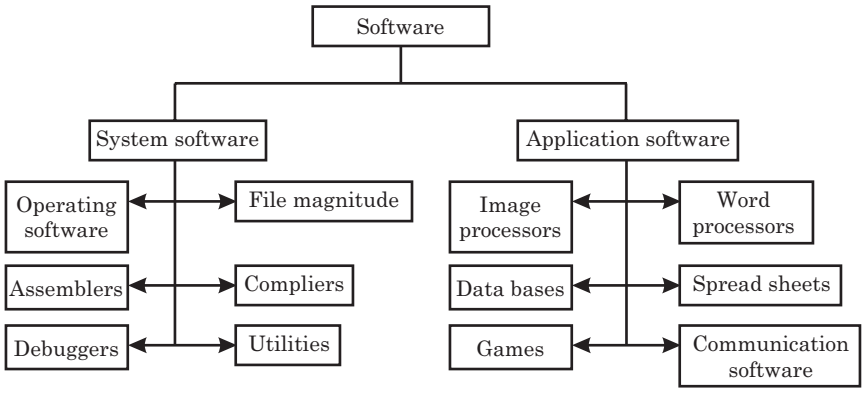

Fig. 1.1 Types of Software

Application software sits a top of system software because it needs help of system software to run. Fig. 1.1 gives an overview of the software classification and its types.

Now we will have an overview of these different classes of software one by one.

1. **Operating Systems.** It is basically the system which provides interface between the user and the hardware. It manages hardware like, memory, CPU, input output devices files etc. for the user. Most commonly used operation systems include Microsoft Windows, DOS, XENIX, Mac OS, OS/2, Unix MVS, etc.

2. **Utilities.** Utilities are programs that perform the very specification tasks related to managing system resources. Operating system in a number of utilities for managing disk printers, other devices.

3. **Compilers and Interpreters.** Compiler is a program that translates source code into object code. The compiler derives its name from the way it works, looking at the entire piece of source code and collecting and reorganizing the instructions. Thus, a compiler differs from an interpreter, which analyzes and executes each line of source code in succession, without looking at the entire program. The advantage of interpreters is that they can execute a program immediately. Compiler require some time before an executable program emerges. However, programs produced by Compiler run much faster than the same programs executed by an interpreter.

4. **Word Processors.** A word processor is a program that enables you to perform word processing functions. Word processors use a computer to create, edit, and print documents. Of all computer applications, word processors are the most common.

To perform word processing, you need a computer, the word processing software (word processor), and a printer. A word processor enables you to create a document, store it electronically on a disk, display it on a screen, modify it by entering commands and characters from the keyboard, and print it on a printer.

The great advantage of word processing over using a typewriter is that you can make changes without retyping the entire document. If you make a typing mistake, you simply back up the cursor and correct your mistake. If you want to delete a paragraph, you simply remove it, without leaving a trace. It is equally easy to insert a word, sentence, or paragraph in the middle of a document. Word processors also make it easy to move sections of text from one place to another within a document, or between documents. When you have made all the changes you want, you can send the file to a printer to get a hardcopy. Some of the commonly used word processors are Microsoft Word, WordStar, WordPerfect, AmiPro, etc.

5. **Spread Sheets.** A spreadsheet is a table of values arranged in rows and columns. Each value can have a predefined relationship to the other values. If you change one value, therefore, you may need to change other values as well.

Spreadsheet applications (often referred to simply as spreadsheets) are computer programs that let you create and manipulate spreadsheets electronically. In a spreadsheet application, each value sits in a cell. You can define what type of data is in each cell and how different cells depend on one another. The relationships between cells are called formulas, and the names of the cells are called labels.

Once you have defined the cells and the formulas for linking them together, you can enter your data. You can then modify selected values to see how all the other values change accordingly. This enables you to study various what-if scenarios.

There are a number of spreadsheet applications in the market, Lotus 1-2-3 and Excel being among the most famous. These applications support graphic features that enable you to produce charts and graphs from the data.

Some spreadsheets are multidimensional, meaning that you can link one spreadsheet to another. A three-dimensional spreadsheet, for example, is like a stack of spreadsheets all connected by formulae. A change made in one spreadsheet automatically affects other spreadsheets.

6. **Presentation Graphics.** Presentation Graphics enable users to create highly stylized images for slide shows and reports. The software includes functions for creating various types of charts and graphs and for inserting text in a variety of fonts. Most systems enable you to import data from a spreadsheet application to create the charts and graphs. Presentation graphics is often called business graphics. Some of the popular presentation graphics software is Microsoft Power Point, Lotus Freelance Graphics, Harvard Presentation Graphics, etc.

7. **Database Management Systems (DBMS).** A DBMS is a collection of programs that enable you to store, modify, and extract information from a database. There are many different types of DBMS, ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are some examples of database applications-computerized library systems, automated teller machines, flight and railway reservation systems, computerized inventory systems, ete.

From a technical standpoint, DBMS can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Requests for information from a database are made in the form of a query, which is a stylized question. Different DBMS support different query languages, although there is a semi-standardized query language called SQL (structured query language). Sophisticated languages for managing database systems are called fourth-generation languages, or 4GLs for short.

The information from a database can be presented in a variety of formats. Most DBMS include a report writer program that enables you to output data in the form of a report. Many DBMS also include a graphics component that enables you to output information in the form of graphs and charts. Some examples of database management system are IDMS, IMS, DB2, Oracle, Sybase, informix, Ingress, MS-SQL Server, MS-Access, ete.

8. **Image Processors.** Image processors or graphics programs enable you to create, edit, manipulate, add special effects, view, and print and save images.

(*i*) **Paint Programs.** A paint program is a graphics program that enables you to draw pictures on the display screen, which is represented as bit maps (bit-mapped graphics). Most paint programs provide the tools in the form of icons. By selecting an icon, you can perform functions associated with the tool. In addition to these tools, paint programs also provide easy ways to draw common shapes such as straight lines, rectangles, circles, and ovals.

Sophisticated paint applications are often called image-editing programs. These applications support many of the features of draw programs, such as the ability to work with objects. Each object, however, is represented as a bit map rather than as a vector image.

(*ii*) **Draw Programs.** A draw program is another graphics program that enables you to draw pictures, then store the images in files, merge them into documents, and print them. Unlike paint programs, which represent images as bit maps, draw programs use vector graphics, which makes it easy to scale images to different sizes. In addition, graphics produced with a draw program have no inherent resolution. Rather, they can be represented at any resolution, which makes them ideal for high-resolution output.

(*iii*) **Image Editors.** Image Editor is a graphics program that provides a variety of special features for altering bit-mapped images. The difference between image editors and paint programs is not always clear-cut, but in general image editors are specialized for modifying bit-mapped images, such as scanned photographs, whereas paint programs are specialized for creating images. In addition to offering a host of filters and image transformation algorithms, image editors also enable you to create and superimpose lavers.

1.3. CLASSES OF SOFTWARE

Software is classified into the following two classes:

1. **Generic software.** Generic software is designed for a broad customer market whose requirements are very common, fairly stable and well understood by the software engineer. These products are sold in the open market, and there could be several competitive products in the market. The database products, browsers, ERP/CRM and CAD /CAM packages, OS and system software are examples of generic software.

2. **Customized software.** Customized products are those that are developed for a customer where domain, environment and requirements being unique to that customer cannot be satisfied by generic products.

Legacy systems, software written for specific business processes that are typical of the specific industry are used when a customized software product is needed. Process control systems, traffic management systems. hospital management systems and manufacturing process control systems require customized software.

The developer manages a generic product and the customer manages a customized product. In other words, requirements and specifications in a generic product are controlled by the developer, whereas in the case of customized product, these are controlled by the customer and influenced by the practices of that industry.

1.4. INTRODUCTION TO SOFTWARE ENGINEERING

Software has become critical to advancement in almost all areas of human endeavour. The art of programming only is no longer sufficient to construct large programs. There are serious problems in the cost, timeliness, maintenance and quality of many software products.

Software engineering has the objective of sowing these problems by producing good quality, maintainable software, on time within budget. Producing good quality, maintainable software, on time within budget. To achieve this objective, we have to focus in a disciplined manner on both the quality of the product and on the procures used to develop the product.

Few important definitions given by several authors and institutions are as under:

IEEE Comprehensive Definition. *Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e., the application of engineering to software.*

According to Barry Boehm. *Software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation.*

According to Fairley. *Software Engineering is a methodological and managerial discipline concerning tile systematic production and maintenance of software products that are developed and maintained with in anticipated and controlled time and cost limits.*

According to Fritz Bauer. *Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*

According to Somerville. *Software Engineering is concerned with the theories, methods and tools that are needed to develop the software products in a cost effective way.*

According to Dermis. *Software Engineering is the application of principles, skills and art to the design and construction of programs and systems of programs.*

According to Morven Gentleman. *Software Engineering is the use of methodologies, tools and techniques to resolve the practical problems that arise in the construction, deployment, support and evolution of software.*

According to Stephen Schach. *Software engineering is a discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements.*

According to Pomberger and Blaschck. *Software Engineering is the practical application of scientific knowledge for the economical production and use of high-quality software.*

According to Rafael J. Barros. *Software Engineering is the application of methods and scientific knowledge to create practical cost-effective solutions for the design, construction, operation and maintenance of software and associated products, in the seroice of mankind.*

Other Definitions. *Software Engineering deals with cost effective solutions to practical problems by applying scientific knowledge in building software artifacts in the service of mankind.*

OR

Software Engineering is the application of methods and scientific knowledge to create practical cost-effective solutions for the design, construction, operation and maintenance of software.

OR

Software Engineering is a discipline whose aim is the production of fault free software that satisfies the user's needs and that is delivered on time and within budget.

OR

The term Software Engineering refers to a movement, methods and techniques aimed at making software development more systematic. Software methodologies like the OMG' s UML and software tools (CASE tools) that help developer's model application designs and then generate code are all closely associated with Software Engineering.

OR

Software Engineering is an Engineering discipline which is concerned with all aspects of software production.

1.4.1. Software Engineering Principles

Ask any student who has had some programming experience the following question:

You are given a problem for which you have to build a software system that most students fed will be approximately 10,000 lines of (say C or Java) code. If you are working full time or it, how long will it take you to build this system?

The answer of students is generally 1 to 3 months. And, given the programming expertise of the students there is a good chance that they will be able to build a system and demo it to the progress within 2 months. With 2 months as the completion time, the productivity of the student will be 5,000 lines of code (LOC) per person month.

The principles deal with both the process of software engineering and the final product. The right process will help produce the right product, but the desired product will also affect the choice of which process to use. A traditional problem in software engineering has been the emphasis on either the process or the product to the exclusion of the other. Both are important.

The principles we develop are general enough to be applicable throughout the process of software construction and management. Principles, however, are not sufficient to drive software development. In fact, they are general and abstract statements describing desirable properties of software processes and products. But, to apply principles, the software engineer should be equipped with appropriate methods and specific techniques that help incorporate the desired properties into processes and products.

In principle, we should distinguish between methods and techniques. Methods are general guidelines that govern the execution of some activity; they are rigorous, systematic, and disciplined approaches. Techniques are more technical and mechanical than methods; often, they also have more technical and mechanical than methods; often, they also have more restricted applicability. In general, however, the difference between the two is not sharp. We will therefore use the two terms interchangeably.

Sometimes, methods and techniques are packaged together to form a methodology. The purpose of a methodology is to promote a certain approach to solving a problem by preselecting the methods and techniques to be used. Tools, in turn, are developed to support the application of techniques, methods, and methodologies.

Fig. 1.2 shows the relationship between principles, methods, methodologies, and tools.

Each layer in the figure is based on the layer(s) below it and is more susceptible to change, due to passage of time. This figure shows clearly that principles are the basis of all methods, techniques, methodologies, and tools.

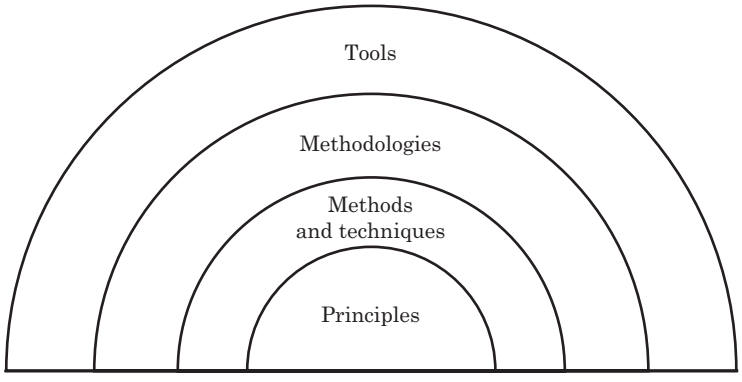

Fig. 1.2. Relationship Between Principles, Techniques, Methodologies, and Tools

In software engineering we are not dealing with programs that people build to illustrate something or for hobby (which we are referring to as student systems). Instead the problem domain is the software that solves some problem of some users where larger systems or businesses may depend on the software, and where problems in the software can lead to significant direct or indirect loss.

1.5. SOFTWARE COMPONENTS

A software component is a system element offering a predefined service and able to communicate with other components. Clemens Szyperski and David Messerschmitt give the following five criteria for what a software component shall be to fulfill the definition:

- Multiple-use
- Non-context-specific
- Cornposable with other components
- Encapsulated *i.e.*, non-investigable through its interfaces
- A unit of independent deployment and versioning.

A simpler definition can be: A component is an object written to a specification. It does not matter what the specification is: COM, Java Beans, etc., as long as the object adheres to the specification, It is only by adhering to the specification that the object becomes a component and, gains features like reusability and so forth.

Software components often take the form of objects or collections of objects (from object-oriented programming), in some binary or textual form, adhering to some Interface Description Language (IDL) so that the component may exist autonomously from other components in a computer.

When a component is to be accessed or shared across execution contexts or network links, some form of serialization (also known as marshalling) is employed to turn the component or one of its interfaces into a bit stream.

It takes significant effort and awareness to write a software component that is effectively reusable. The component needs:

- to be fully documented;
- more thorough testing;
- robust input validity checking;
- to pass back useful error messages as appropriate;
- to be built with an awareness that it will be put to unforeseen uses;
- a mechanism for compensating developers who invest the (substantial) effort implied above.

1.6. SOFTWARE CHARACTERISTICS

The software has a very special characteristic e.g., “it does not wear out”. Its behavior and nature is quite different than other products of human life. Both activities require different processes and have different characteristics.

Now we have a better understanding of the problem domain that software engineering deals with, let us orient our discussion to software Engineerings itself. Software engineering is defined as systematic approach to the development, operation, maintenance, and retirement of the software.

The key characteristics of software are as under:

1. **Most software is custom-built, rather than being assembled from existing components.** Most software continues to be custom built, although recent developments tend

to be component based. Modern reusable components encapsulate both data and the processing applied to data, enabling the software engineer to create new applications from reusable part. For example, today GUI is built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms. The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction.

2. **Software is developed or engineered; it is not manufactured in the classical sense.** Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent for software. Both activities depend on people, but the relationship between people applied and work accomplished is entirely different. Both require the construction of a "product". But the approaches are different. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

3. **Software is flexible.** We all feel that software is flexible. A program can be developed to do almost anything Sometimes, this characteristic may be the' best and may help us to accommodate any kind of change. Reuse of components from the libraries help in reduction of effort. Now-days, we reuse not only algorithms but also data structures.

4. **Software doesn't wear out.** There is a well known "bath-tub curve" in reliability studies for the hardware products. Fig. 1.3 depicts failure rate as a function of time for hardware. The relationship, often called the "bath-tub curve". Note that, wear out means process of loosing the material.

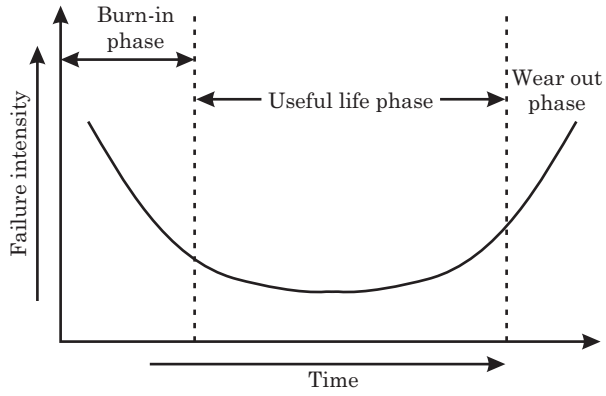

Fig. 1.3. Bath Tub Curve

There are three phases for the life of a hardware product. Initial phase is burn-in phase, where failure intensity is high. It is expected to test the product in the industry before delivery. Due to testing and fixing faults, failure intensity will come down initially and may stabilize after certain time. The second phase is the useful life phase where failure intensity is approximately constant and is called useful life of a product. After few years, again failure intensity will increase due to wearing out of components. This phase is called wear out phase. We do not have this phase for the software, as it does not wear out. The curve for software is given in Fig. 1.4

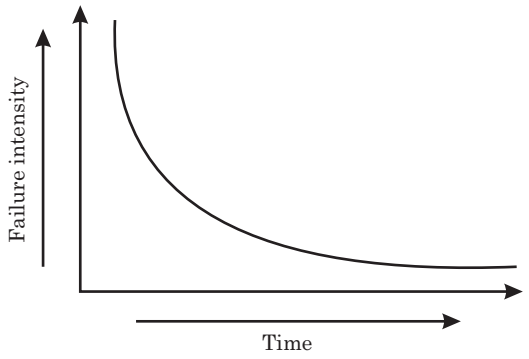

Fig. 1.4. Software Curve

Important point is software becomes reliable overtime instead of wearing out. It becomes obsolete, if the environment, for which it was developed, changes. Hence software may be retired due to environmental changes, new requirements, new expectations, etc.

1.7. SOFTWARE CRISIS

The software crisis has been with us since 1970s. As per the latest IBM report, "31% of the projects get cancelled before they are completed, 53% over-run their cost-estimates by an average of 189% and for every 100 projects, there are 94 restarts".

When software is developing then during development many problems are raised up that set of problem is known as software crisis. When software is developing then on the different steps of development, problems are encountered associated with those steps. Now we will discuss the problem and causes of software crisis encounter on different stage of software development.

1.7.1. Problems

- Schedule and cost estimates are often grossly inaccurate.
- The "Productivity" of software people hasn't kept pace with the demand for their services.
- The quality of software is sometimes less than adequate.
- With no solid indication of productivity, we can't accurately evaluate the efficiency of new, tools, methods or standards.
- Communication between customer and software developer is often poor.
- The software maintenance task devours the majority of all software rupees.

1.7.2. Causes

- Quality of software is not good because most of the developer use the historical data to develop the software.
- If there is delay in any process or stage *i.e.*, analysis, design, coding and testing) then scheduling does not match with actual timing.
- Communication between managers and customers, software developers, support staff etc. can break down because the special characteristics of software and the problems associated with its development are misunderstood.
- The software people responsible for tapping that potential often change when it is discussed and resist change when it is introduced.

1.7.3. Software Crisis in the Programmer's Point of View

- Problem of compatibility.
- Problem of portability.
- Problem in documentation.
- Problem of piracy of software.
- Problem in co-ordination of work of different people.
- Problem of maintenance in proper manner.

1.7.4. Software Crisis in the User's Point of View

- Software cost is very high.
- Customers are moody or choosy.
- Hardware goes very down.
- Luck of specialization in development.
- Problem of different versions of software.
- Problem of views.
- Problem of bugs.

1.8. SOFTWARE MYTHS

There are Number of myths associated with software development community. Some of them really affect the way, in which software development should take place. In this section, we list few myths, and discuss their applicability to standard software development.

- If we get behind schedule, we can add more programmers and catch up.
- If I decide to outsource the software project to a third party, I can just relax and let that firm build it.
- Project requirement continuously changes, but changes can be easily accommodated because software is flexible.
- The only deliverable work product for a successful project is the working program.
- Software with more features is better software.
- Once we write the program and get it to work, our job is done.
- Until I get the program running, I have no way of assessing its quality.
- Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.
- A general statement of objectives is sufficient to begin writing programs; we can fill in the details later.
- We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

1.9. SOFTWARE APPLICATIONS

Software has become integral part of most of the fields of human life. We name a field and we find the usage of software.

Software applications are grouped into eight areas for convenience as shown in Fig. 1.5.

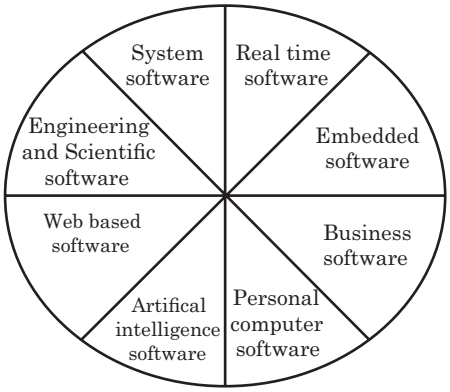

Fig. 1.5. Software Applications

1. **System Software.** System software is a collection of programs used to run the system as an assistance to use other software programs. The compilers, editors, utilities, operating system components, drivers and interfaces, assemblers, compilers, linkers and loaders are examples of system software. This software resides in the computer system and consumes its resources. A computer system without system software cannot function.

   System software directly interacts with the hardware, heavy usage by multiple users, concurrent operations that requires scheduling, resource sharing and sophisticated process management, complex data structures and multiple external interfaces.

2. **Real-time Software.** Real time software deals with changing environment. First it collects the input and convert it from analog to digital, control component that responds to the external environment, perform the action in the last.

   The software is used to monitor, control and analyze real world events as they occur. Examples are Rocket launching, games etc.

3. **Embedded Software.** Software, when written to perform certain functions under control conditions and further embedded into hardware as a part of large systems, is called embedded software.

   The software resides in Read-Only-Memory (ROM) and is used to control the various functions of the resident products. The products could be a car, washing machine, microwave oven, industrial processing products, gas stations, satellites and a host of other products, where the need is to acquire input, analyze, identify status, decide and take action that allows the product to perform in a predetermined manner. Because of their role and performance, they are also termed intelligent software.

4. **Business Software.** Software designed to process business applications is called business software. Business software could be a data- and information-processing application. It could drive the business process through transaction processing in on-line or in real-time mode.

   This software is used for specific operations such as accounting package, Management information system, pay roll package, inventory management. Business software restructures existing data in order to facilitate business operations or management decision making. It also encompasses interactive computing. It is an integrated software related to a particular field.

5. **Personal Computer Software.** The personal computer software market has burgeoned over the past two decades. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network or database access are only a few of hundreds of applications.
6. **Artificial Intelligence Software.** Artificial Intelligence software uses non-numerical algorithms, which use the data and information generated in the system, to solve the complex problems. These problem scenarios are not generally amenable to problem-solving procedures, and require specific analysis and interpretation of the problem to solve it as shown in Fig. 1.6.

   Application within this area include robotics, expert system, pattern recognition (image and voice), artificial neural networks, theorem proving and game playing, signal processing software.

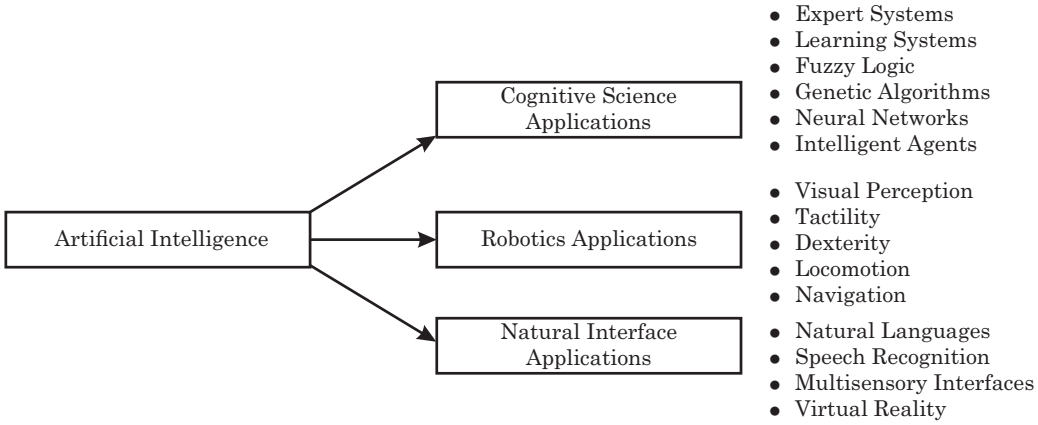

Fig. 1.6. Application Areas of Artificial Intelligence

7. **Web-based Software.** Web-based software is the browsers by which web pages are processed *i.e.*, HTML, Java, CCl, Perl, DHTML etc.
8. **Engineering and Scientific Software.** Design, engineering of scientific software's deal with processing requirements in their specific fields. They are written for specific applications using the principles and formulae of each field. In this type, application areas are:

Astronomy, Volcanology, Molecular biology, Computer Aided Design (*e.g.*, AutoCAD software) system simulations.

These software's service the need of drawing, drafting, modelling. lead calculations, specifications-building and so on. Dedicated software's are available for stress analysis or for analysis of engineering data, statistical data for interpretation and decision-making. CAD / CAM/ CAE packages, SPSS, MATLAB, circuit analyzers are typical examples of such software.

1.10. SOFTWARE ENGINEERING PROCESSES

According to Webster, the term process means "a particular method of doing something, generally involving a number of steps or operations". In software engineering, the phrase software process refers to the methods of developing software.

A software process is a set of activities, together with ordering constraints among them, such that if the activities are performed properly and in accordance with the ordering constraints, the desired result is produced. The basic desired result is, as stated earlier, high quality and productivity.

1.10.1 Process

A process is a series of steps involving activities, constraints and resources that produce an intended output of some kind.

Any process has the following characteristics:

- The process prescribes all of the major process activities.
- The process uses resources, subject to a set of constraints (such as a schedule), and produces intermediate and final products.
- The process may be composed of sub processes that are linked in some way. The process may be defined as a hierarchy of processes, organized so that each sub process has its own process model.
- Each process activity has entry and exit criteria, so that we know when the activity begins and ends.
- The activities are organized in a sequence, so that it is clear when one activity is performed relative to the other activities.
- Every process has a set of guiding principles that explain the goals of each activity.
- Constraints or controls may apply to an activity, resource, or product. For example, the budget or schedule may constrain the length of time an activity may take or a tool may limit the way in which a resource may be used.

1.10.2. What is a Software Process?

The software process is the way in which we produce software. This differs from organization to organization. Surviving in the increasingly competitive software business requires more than hiring smart, knowledgeable developers and buying the latest development tools. We also need to use effective software development processes, so that developers can systematically use the best technical and managerial practices to successfully complete their projects. Many software organizations are looking at software process improvement as a way to improve the quality, productively predictability of their software development, and maintenance efforts.

Software process is the related set of activities and processes that are involved in developing and evolving a software system.

OR

A set of activities whose goal is the development or evolution of software.

OR

A software process is a set of activities and associated results, which produce a software product.

These activities are mostly carried out by software engineers. There are four fundamental process activities (covered later in the book), which are common to all software processes. These activities are:

1. **Software specification:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the specification must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing customer needs. Different software processes organize these activities in different ways and are described at different levels of detail. The timing of the activities varies, as does the

results of each activity. Different organizations may use different processes to produce the same type of product. However, some processes are more suitable than others for some types of application. If an inappropriate process is used, this will probably reduce the quality or the usefulness of the software product to be developed.

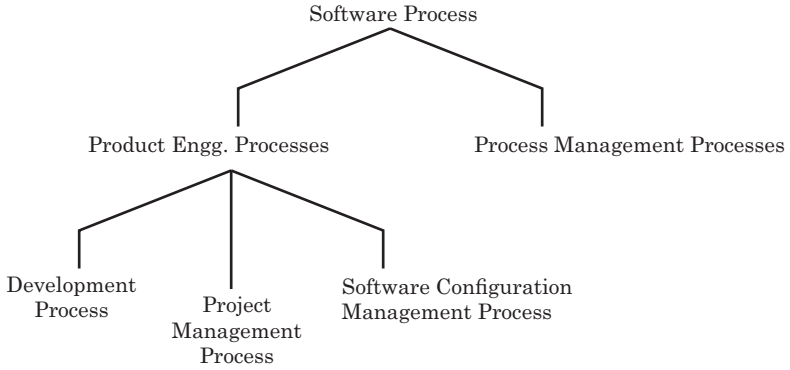

Fig. 1.7. The Software Process

A software process can be characterized as shown in Fig. 1.7. A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. A number of task sets-each a collection of software engineering work tasks, project milestones, software work products and deliverables, and quality assurance points-enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities–such as software quality assurance, software configuration management and measurement-overlay the process model. Umbrella activities are independent of any one-framework activity and occur throughout the process.

It seems straight forward, and literature has number of successes stories of companies that substantially improved their software development and project management capabilities. However, many other organizations do not manage to achieve significant and lasting improvements in the way they conduct their projects.

Thus, the software industry considers software development as a process. According to Booch and Rumbaugh, *A process defines who is doing what, when and how to reach a certain goal*? Software engineering is a field, which combines process, methods and tools for the development of software. The concept of process is the main step in the software engineering approach. Thus, a software process is a set of activities. When those activities are performed in specific sequence in accordance with ordering constraints, the desired results are produced.

1.11. EVOLUTION OF SOFTWARE

The software has seen many changes since its inception. After all, it has evolved over the period of time against all odds and adverse circumstance. computer industry has also progressed at a break-neck speed through the computer revolution, and recently, the network revolution triggered and/or accelerated by the expansive, spread of the Internet and most recently the web computer industry has been delivering exponential improvement in price-performance, but the problems with software have not been decreasing. Software still come late, exceed budget and are full of residual faults. As per the latest IBM report, "31% of the projects get canceled before they are completed, 53% over-run their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts".

Software engineering principles have evolved over the past more than fifty years from art to an engineering discipline. It can be shown with the help of the following Fig. 1.8.

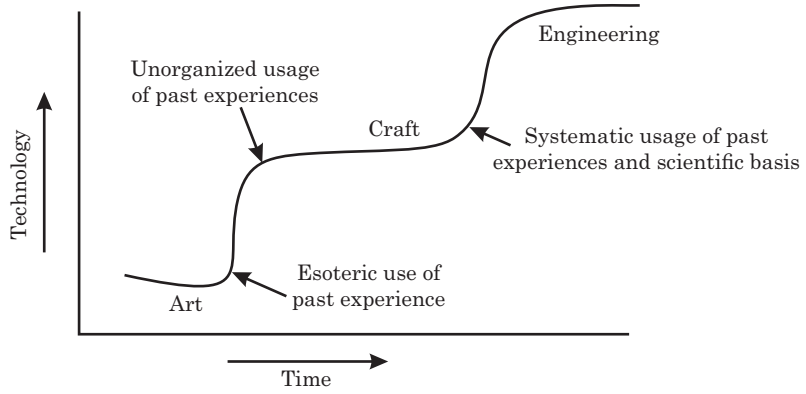

Fig. 1.8. Evolution of Art to an Engineering Discipline

Development in the field of software and hardware computing make a significant change in the twentieth century. We can decide the software development process into four eras:

1. **Early Era.** During the early eras general-purpose hardware became commonplace. Software, on the other hand, was custom-designed for each application and had a relatively limited distribution. Most software was developed and ultimately used by the same person or organization.

In this era the software are mainly based on (1950-1960)

- Limited Distribution
- Custom Software
- Batch Orientation

2. **Second Era.** The second era to computer system evolution introduced new concepts of human machine interaction. Interactive techniques opened a new world of application and new levels of hardware and software sophistication. Real time software deals with the changing environment and one other is multi-user in which many users can perform or work on a software at a time.

In this era the software are mainly based on (1960-1972)

- Multi-user
- Data base
- Real time
- Product software
- Multiprogramming

3. **Third Era.** In the earlier age the software was custom designed and limited distribution but in this era the software was consumer designed and the distribution is also not limited. The cost of the hardware is also very low in this era.

In this era the software are mainly based on (1973-1985)

- Embedded intelligence
- Consumer Impact
- Distributed Systems
- Low Cost Hardware

4. **Fourth Era.** The fourth era of computer system evolution moves us away from individual computers and computer programs and toward the collective impact of computers and software. As the fourth era progresses, new technologies have begun to emerge.

In this era the software are mainly based on (1985-)

- Powerful desktop systems
- Expert systems
- Artificial Intelligence
- Network computers
- Parallel computing
- Object oriented technology

At this time the concept of software making is object oriented technology or network computing etc.

Software engineering principles have evolved over the last sixty years with contributions from numerous researchers and software professionals. Over the years, it has emerged from a pure art to a craft, and finally to an engineering discipline.

The early programmers used an ad hoc programming style. This style of program development is now variously being referred to as exploratory, build and fix, and code and fix styles. In a build and fix style, a program is quickly developed without making any specification, plan or design. The different imperfections that are subsequently noticed are fixed.

1.12. COMPARISON OF SOFTWARE ENGINEERING AND RELATED FIELDS

The relationships between software engineering and the fields of computer science, and traditional engineering have been debated for decades. Software engineering resembles all of these fields, but important distinctions exist.

1.12.1. Comparing Computer Science

Many compare software engineering to computer science and information science like they compare traditional engineering to physics and chemistry.

About half of all software engineers earn computer science degrees. Yet on the job, practitioners do applied software engineering, which differs from doing theoretical computer science, as illustrated in Table 1.1.

TABLE 1.1

Issue	*Software Engineering*	*Computer Science*
Ideal	Constructing software applications for real-world use for today	Finding eternal truths about problems and algorithms for posterity
Results	Working applications (like office suites and video games) that deliver value to users.	Computational complexity, and correctness of algorithms (like shell sort) and analysis of problems (like the travelling salesman problem)
Budgets and Schedules	Projects (like upgrading an office suite) have fixed budgets and schedules	Projects (like solving P=NP'?) have open ended budgets and schedules

Change	Applications evolve as user needs and expectations evolve, and as SE technologies and Oractices evolve.	When computer science problems are solved, the solution will never change
Additional Skills	Domain knowledge	Mathematics
Notable Educators and Researchers	Barry Boehm, Fred Brooks, and David Parnas	Edsger Dijkstra, Donald Knuth, and Alan Turing
Notable Practitioners	Dan Bricklin, Steve McConnell	Not Applicable
Practitioners in U.S.	680,000	25,000
Practitioners in Rest of World	1,400,000?	50,000?

1.12.2. Comparing Engineering

The software engineering community is about 60% as large as the rest of engineering community combined.

Software engineers aspire to build low-cost, reliable, safe products; much like engineers in other disciplines do. Software engineers borrow many metaphors and techniques from other engineering disciplines, including requirements analysis, quality control, and project management techniques. Engineers in other disciplines also borrow many tools and practices from software engineers. Yet, there are also some differences between SE and other engineering disciplines as illustrated in Table 1.2.

TABLE 1.2

Issue	*Software Engineering*	*Engineering*
Foundations	Based on computer science, information science, and discrete math.	Based on science, mathematics, and empirical knowledge.
Cost	Compilers and computers are cheap, so software engineering and consulting are often more than half of the cost of a project. Minor software engineering cost-over runs can adversely affect the total project cost.	In some projects, construction and manufacturing costs can be high, so engineering may only be 15 of the cost of a project. Major engineering cost overruns may not affect the total project cost.
Replication	Replication (copying CDs or down loading files) is trivial. Most development effort goes into building new (unproven) or changing old designs and adding features.	Radically new or one-of-a-kind systems can require significant development effort to create a new design or change an existing design. Other kinds of systems may require less development effort, but more attention to issues such as manufacturability.
Innovation	Software engineers often apply new and untested elements in software projects.	Engineers generally try to apply known and tested principles, and limit the use of untested innovations to only those necessary to create a product that meets its requirements.

Duration	Software engineers emphasize projects that will live for years or decades.	Some engineers solve long-ranged problems (bridges and dams) that endure for centuries.
Management Status	Few software engineers manage anyone.	Engineers in some disciplines, such as civil engineering, manage construction, manufacturing, or maintenance crews.
Blame	Software engineers must blame themselves for project problems.	Engineers in some fields can often blame construction, manufacturing, or maintenance crews for project problems.
Practitioners in U.S.	611,900 software engineers	1,157,020 total non-software engineers
Age	Software engineering is about 50 years old.	Engineering as a whole is thousands of years old.
Title Regulations	Software engineers are typically self-appointed. A computer science degree is common but not at all a formal requirement.	In many jurisdictions it is illegal to call yourself an engineer without specific, formal education and/ or accreditation by! governmental or engineering association bodies.
Analysis Methodology	Methods for formally verifying correctness are developed in computer science, but they are rarely used by software engineers. The issue remains controversial.	Some engineering disciplines are based on a closed system theory and can in theory prove formal correctness of a design. In practice, a lack of computing power or input data can make such proofs of correctness intractable, leading many engineers to use a pragmatic mix of analytical approximations and empirical test data to ensure that a product will meet its requirement
Synthesis Methodology	SE struggles to synthesize (build to order) a result according to requirements.	Engineers have nominally refined synthesis techniques over the ages to provide exactly this. However, this has not prevented some notable engineering failures, such as the collapse of the Tacoma Narrows Bridge, the sinking of the Titanic, and the Pentium FDlV bug. In addition, new technologies inevitably result in new challenges that cannot be met using existing techniques.

Research during Projects	Software engineering is often busy with researching the unknown (*e.g.*, to derive an algorithm) right in the middle of a project.	Traditional engineering nominally separates these activities. A project is supposed to apply research results in known or new clever ways to build the desired result. However, ground-breaking engineering projects such as Project Apollo often include a lot of research into the unknown.
Codified	Software engineering has just recently started to codify and teach best practice in the form of design patterns.	Some engineering disciplines have thousands of years of best practice experience handed over from generation to generation via a field's literature, standards, rules and regulations. Newer disciplines such as electronic engineering and computer engineering have codified their own best practices as they have developed.

1.13. SOME TERMINOLOGIES

Some terminologies are discussed in these sections which are frequently used in the field of Software Engineering.

1. **Deliverables and Milestones.** Different deliverables are generated during software development. The examples are source code, user manuals, operating procedure manuals etc. The milestones are the events that are used to ascertain the status of the project. Finalization of specification is a milestone. Completion of design documentation is another milestone. The milestones are essential for project planning and management.
2. **Product and Process.** What is delivered to the customer is called a product. It may include source code specification document, manuals, documentation etc. Basically, it is nothing but a set of deliverables only.

   Process is the way in which we produce software. It is the collection of activity that leads to (a part of) a product. An efficient process is required to produce good quality products. If the process is weak, the ends product will undoubtedly suffer, but an obsessive over reliance on process is also dangerous.
3. **Measures, Metrics and Indicators.** In software engineering measures provides a quantitative indication of amount, dimension, capacity or size of given attribute of a product.

   The metrics is a quantitative measures of the degree to which a system, component, or process possesses a given attribute of a product. An indicator is a combination of metrics.

   Measurement occurs as the result of the collection of one or more data points (*e.g.*, a number of module reviews are investigated to collect measures of the number of errors in each module.)

1.14. PROGRAMS VERSUS SOFTWARE PRODUCTS

Before discussing about the various types of development projects that are being undertaken by software development companies, let us first understand the important ways in which professional software differs from toy software such as those written by a student in this first programming assignment.

1.14.1. Programs

A program is a subset of software and it becomes software only if documentation and operating procedure manuals are prepared. Program is a combination of source code and object code as shown in Fig. 1.9.

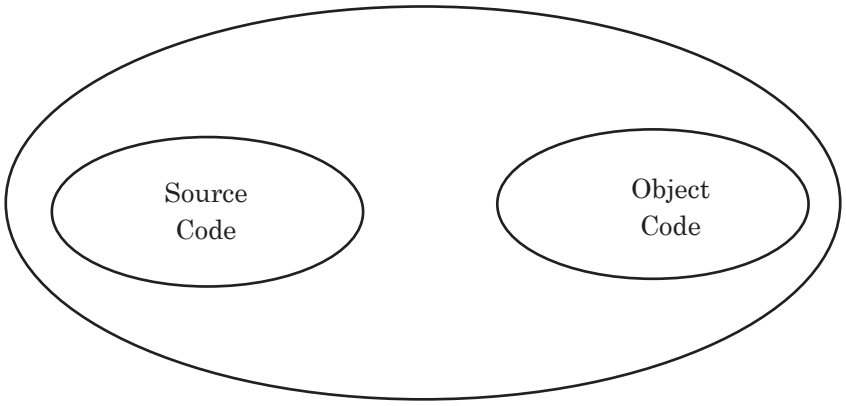

Fig. 1.9. Program = Source Code + Object Code

1.14.2. Software Products

A software product consists not only of the program code but also of all the associated documents such as the requirements specification documents, the design documents, the test document, the operating procedures which includes user manuals and operational manuals as shown in Fig. 1.10.

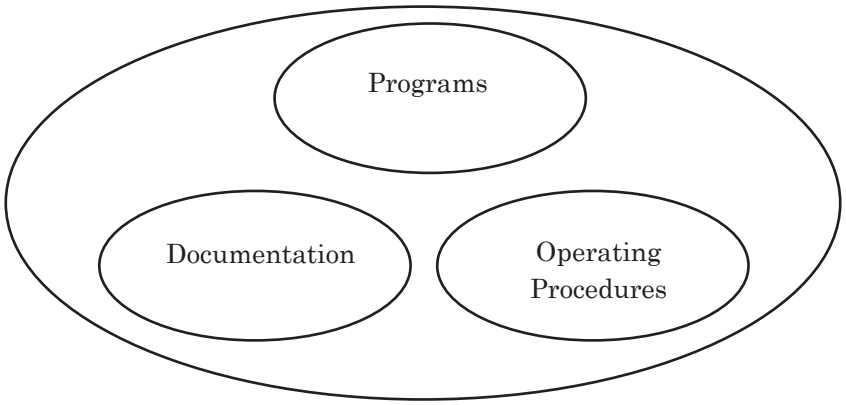

Fig. 1.10. Software = Program + Documentation + Operating Procedures

1.14.3. Programs Versus Software Products

The various differences between a program product and a software product are given in the tabular form, as illustrated in Table 1.3:

TABLE 1.3

Programs	*Software Products*
1. Programs are developed by individuals for their personal use	1. A software product is usually developed by a group of engineers working in a team
2. Usually small in size	2. Usually large in size
3. Single user	3. Large number of users
4. Single developer	4. Team of developers
5. Lacks proper documentation	5. Good documentation support
6. Adhoc development	6. Systematic development
7. Lack of user interface	7. Good user interface
8. Have limited functionality	8. Exhibit more functionality

Self-Assessment Exercises

1. Define software.
2. What is software engineering?
3. What do you mean by the term "Software Engineering"? Describe the evolving role of software.
4. What are the different myths and realities about the software?
5. Give the various application areas of the software.
6. What is bath tub curve?
7. Discuss the characteristics of the software.
8. What characteristics of software' make it different from other engineering products (for example hardware)?
9. Explain some characteristics of software. Also discuss some of the software components.
10. Comment on the statement "software does not wear out".
11. Discuss about the evolution of software engineering as a subject in the last 50 years.
12. What are the different software components?
13. What are the symptoms of the present software crisis? What factors have contributed to the making of the present software crisis? What are possible solutions to the present software crisis?
14. What do you understand by software crisis?
15. What is software crisis? Give the problems of software crisis.
16. What do you mean by software myths?
17. Explain in detail software engineering process.
18. Why has software become too much important in modern days desktop publishing?
19. Distinguish between a program and a software product.
20. Discuss the two well-known principles used in software engineering to tackle the complexity of development of large programs.
21. What is the difference between software engineering and conventional engineering?