Introduction

Software:

The *Institute of Electrical and Electronic Engineers (IEEE)* defines software as a collection of computer programs, procedures, rules and associated documentation and data.

Software does not wear out: Different things like clothes, shoes, ornaments do wear out after some time. But, software once created never wears out. It can be used for as long as needed and in case of need for any updating, required changes can be made in the same software and then it can be used further with updated features.

Software is not manufactured: Software is not manufactured but is developed. So, it does not require any raw material for its development.

Software controls, integrates, and manages the hardware components of a computer system. It also instructs the computer what needs to be done to perform a specific task and how it is to be done. For example, software instructs the hardware how to print a document, take input from the user, and display the output.

Software Characteristics:

Different individuals judge software on different basis. This is because they are involved with the software in different ways. For example, users want the software to perform according to their requirements. Similarly, developers involved in designing, coding, and maintenance of the software evaluate the software by looking at its internal characteristics, before delivering it to the user. Software characteristics are classified into six major components.

Efficiency - the software is produced in the expected time and within the limits of the available resources.

Reliability - This is the ability of the software to provide the desired functionalities under every condition. This means that our software should work properly in each condition.

Usability - The usability of the software is the simplicity of the software in terms of the user. The easier the software is to use for the user, the more is the usability of the software as more number of people will now be able to use it and also due to the ease will use it more willingly.

Flexibility - A software is flexible. What this means is that we can make necessary changes in our software in the future according to the need of that time and then can use the same software then also.

Portability - Portability of the software means that we can transfer our software from one platform to another that too with ease. Due to this, the sharing of the software among the developers and other members can be done flexibly.

Reusability - As the software never wears out, neither do its components, i.e. code segments. So, if any particular segment of code is required in some other software, we can reuse the existing code form the software in which it is already present. This reduced our work and also saves time and money.

Maintainability - Every software is maintainable. This means that if any errors or bugs appear in the software, then they can be fixed.

Interoperability - the software system can interact properly with other systems. This can apply to software on a single, stand-alone computer or to software that is used on a network.

Correctness - The program produces the correct output.

Building the Software Products:

Building a software product is an activity similar to the construction of an apartment. In a sample scenario, a software company notices that a software product needs to be developed because there is a market for that product. The motive for this project is depicted in below figure. The company then initiates a project. A project team is formed. A software project manager is appointed to create a project plan. The project plan includes details such as how much time and how many people are needed. Business analysts create a list of product features (i.e., the features that the proposed product should have). The software designers prepare the architecture of the product. Software developers then construct the product (i.e., write the source code using a programming language such as C++). Once the product is ready and tested, it is shipped to the market. The software project manager oversees all these activities to ensure that the things on the project go as per the project plan. Similar to the measures (such as earthquake resistance, fire alarms, and fire exits) taken while constructing the apartment, some considerations are needed to build a software product. These considerations include security, performance, and reliability of the product. These considerations will be addressed when the software product is developed.

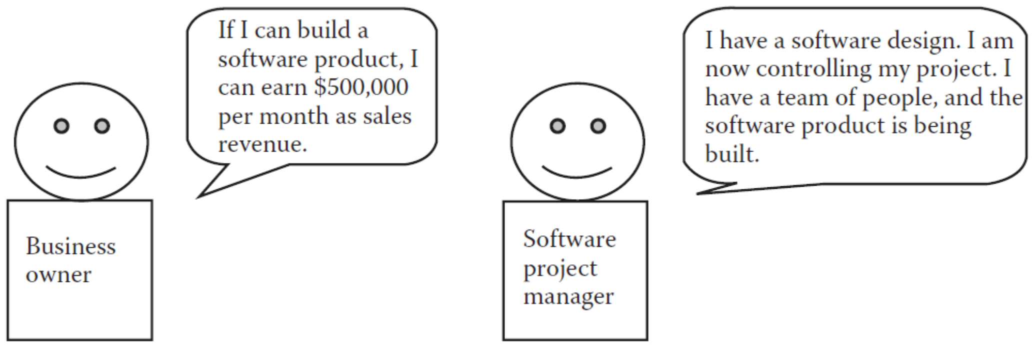

If I can build a software product, I can earn $500,000 per month as sales revenue.
Business owner
I have a software design. I am now controlling my project. I have a team of people, and the software product is being built.
Software project manager

Figure 1.2 A business owner initiates a project for building a software product and the software project manager plans and controls the software development project.

Types of Software Products:

Software engineers are concerned with developing software products, that is, software that can be sold to a customer. There are two kinds of software product:

Generic products:

- ✓ Product developers own the specification.
- ✓ Developer can be able to change the specification due to some other external change.
- ✓ Generic product user cannot control the evolution of the product.
- ✓ User can get application immediately.
- ✓ Example : Microsoft Word, Microsoft power point, Microsoft Excel

Customized products:

- ✓ Customers own the specification and it also controlled by the customer.
- ✓ Customer is also involved; therefore for changing the specification, both customer and the developer must make discussion for the implementation.
- ✓ The customer and the developer will involve in the business process changes and they will implement if both are satisfied.
- ✓ The application will take some year for the process of development and it is not available immediately.
- ✓ Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Software Failure:

Software failures are a consequence of two factors:

a) *Increasing system complexity*

   As new software engineering techniques help us to build larger, more complex systems, the demands change. Systems have to be built and delivered more quickly; larger, even more complex systems are required; systems have to have new capabilities that were previously thought to be impossible.

b) *Failure to use software engineering methods*

   It is fairly easy to write computer programs without using software engineering methods and techniques. Many companies have drifted into software development as their products and services have evolved. They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be.

Software Product and Software Process:

These two words are the one which is mostly confused with each other. In this article, we are going to explain each of these in detail.

Software product:

The final software that is delivered to the customer is called the software product. It is the outcome of the entire software development process. It may include source code, data, user guides, reference manuals, installation manuals, specification documentation, other documentation, etc. The software product does not have any information regarding the software process, like how it was scheduled, how many people worked on it, how the work was divided, etc. It only consists of the final application that fulfills the user's requirements.

Software process:

Four fundamental activities are common to all software processes.

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software development, where the software is designed and programmed.
3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. Software evolution, where the software is modified to reflect changing customer and market requirements.

The software process is the entire way in which we produce the software. It is the entire journey from the idea of the Software to the final release of it. It includes all the activities that are performed to the form the final Software product, like the requirement analysis, designing of the software, coding, testing, documentation, Maintenance, etc. Hence, the software product can also be defined as the collection of all the activities that as a result leads to the formation of the software product. The Software Product may not contain details about the software process, but the software process has every detail about the final product from the very initial phase itself that how the software would be like.

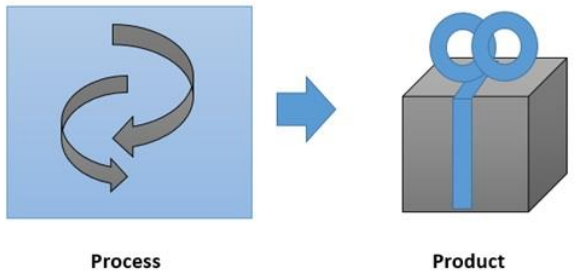


Fig: Relation between Software product and Software process

Of course, both of these, the software product and the software process are related to each other. An efficient process is very important to produce a good quality software product. If the software development process is weak, then the final product will undoubtedly suffer.

However, the software product is more dependent upon the software process. This can be understood in the following way: If we have a software product, then it would surely have its history which is the software process. But, the case is not the same with Software process. An ongoing software process doesn't need to lead to a final product. There are chances that some problems may occur in the projects development phase and it may be canceled. And if we take a look at the present software scenario, then this situation is seen in more than 60 percent of the cases.

What is Software Engineering?

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

Software engineering is not just programming.

The keyword here is *engineering*. Engineering always connotes two things: designing a product in detail and a methodical approach to carrying out the design and production so that the product is produced with the desired qualities and a large number of people can use it.

As a note, software engineering is not only limited to creating new products. Existing software products need maintenance. Software maintenance is also a part of software engineering.

Why Software Engineering?

Small programs such as a program for converting temperature from Celsius to Fahrenheit or an individual programming assignment given in a typical classroom do not need any software engineering. However, building large software products does require software engineering.

If you need to build a software product, then you need to take into consideration the cost economics, time factors, and quality factors. If a software product is built with excessive cost, then the company that is paying to build the product will incur losses. If a software product cannot be built within the accepted time limits, then the product cannot be launched into the market at the right time, and thus the company that owns the product may not make the expected profits. If a software product fails to meet the quality standards, then it cannot be used effectively by users. Maintaining a poor-quality software product is also costly. The company owning such a software product may incur heavy support costs in terms of time spent by the support personnel in fixing and addressing customer complaints. Software engineering is thus extremely important.

Reduction of Development Costs:

Most software products evolve over time by maintaining them or by developing new products based on the experience gained from the existing products. Software engineering helps reduce the cost of development by reusing the existing code, using better tools, and following sound software engineering practices.

Reduction of Development Time:

There are some software products and services available in the market that provide the needed functionality immediately; thus, the project team can avoid writing fresh source codes altogether to achieve the same functionality. The end result is that the productivity of the project team continues to improve, which allows the faster development of software products.

Increasing the Quality:

Quality is the single most important ingredient in making any product successful. Software engineering plays an important role in building quality software products. By using better software engineering methodologies and software engineering tools, it is possible to build better-quality software products. Making the quality assurance processes an intrinsic part of the software development processes ensures the development of quality software products.

Challenges in Software Engineering

Software engineering employs a well-defined and systematic approach to develop software. This approach is considered to be the most effective way of producing high-quality software. However, despite this systematic approach in software development, there are still some serious challenges faced by software engineering. Some of these challenges are listed below.

- ✓ The challenge in software engineering is to deliver high-quality software on time and on budget to customers.
- ✓ The accurate estimation of project cost, effort and schedule is a challenge in software engineering. Therefore, project managers need to determine how good their estimation process actually is and to make appropriate improvements.
- ✓ Risk management is an important part of project management, and the objective is to identify potential risks early and throughout the project and to manage them appropriately. The probability of each risk occurring and its impact is determined, and the risks are managed during project execution.
- ✓ Software quality needs to be properly planned to enable the project to deliver a quality product. Flaws with poor quality software lead to a negative perception of the company and may potentially lead to damage to the customer relationship with a subsequent loss of market share. The quality needs to be carefully considered when designing and developing software. The effect of software failure may be large costs to correct the software, loss of credibility of the company or even loss of life.
- ✓ The methods used to develop small or medium-scale projects are not suitable when it comes to the development of large-scale or complex systems.
- ✓ Changes in software development are unavoidable. In today's world, changes occur rapidly and accommodating these changes to develop complete software is one of the major challenges faced by the software engineers.
- ✓ Sometimes, changes are incorporated in documents without following any standard procedure. Thus, verification of all such changes often becomes difficult.
- ✓ The key challenges facing software engineering is - coping with increasing diversity, demands for reduced delivery times and developing trustworthy software

Software Engineering Diversity:

- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.
- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.

Software Application Types

- ✓ *Stand-alone applications:*
    - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- ✓ *Interactive transaction-based applications*
    - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- ✓ *Embedded control systems*
    - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.
- ✓ *Batch processing systems*
    - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
- ✓ *Entertainment systems*
    - These are systems that are primarily for personal use and which are intended to entertain the user.
- ✓ *Systems for modeling and simulation*
    - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
- ✓ *Data collection systems*
    - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- ✓ *Systems of systems*
    - These are systems that are composed of a number of other software systems.

Software Engineering vs Computer Science:

The primary difference is that computer science was originally a sub-branch of mathematics. Computer science deals with the basic structure of a computer and is more theoretical. Hence, it is more malleable in terms of specialization, with the emphasis on math and science. Software engineering is a field concerned with the application of engineering processes to the creation, maintenance, and design of software for a variety of different purposes. A software engineer designs customized applications per the requirements of an organization.

One should choose Computer Science if he wants to get into a specialized field in CS like artificial intelligence, machine learning, security, or graphics. On the other hand, one should choose Software Engineering if he wants to learn the overall life cycle of how specific software is built and maintained.

Software Engineering vs. Software Development:

- ✓ Engineers are thus much more like *architects* or even *designers*. Their techniques for doing so therefore are less concerned with *building quickly* than they are with finding and designing the *right* solution amidst trying out new things.
- ✓ An engineer designs and plans applying the principles of engineering to software development. Always aware of the "big picture", with talents in many areas. An engineer's core focus lies with architecture. A developer executes. Their talents often focused on a single area.
- ✓ Anyone can be a software developer. If you know a small amount of programming concept then you have the foundation to become Software Developer. They write code without any performance and scalability analysis. More of this element is completed by the Software Engineer, making that role different from a Software Developer.
- ✓ A software engineer is responsible for making system design and prototyping. So the software developer is mainly focused on developing code and testing; these are part of software development cycle. Software developer is responsible for coding.
- ✓ In general, developers are not responsible for designing the whole system. Whereas engineers design, liaise, and oversee the project, software developers write code to bring the project to life.
- ✓ A developer executes. Their talents often focused on a single area. Without the need for the "big picture".

Methodologies Used for Software Development:

A software product can be developed using any of the popular software engineering methodologies (or models or approaches). At one extreme, the entire product is developed under one project using a big project planning model. At the other extreme, you can develop the product incrementally. In between these two extremes, other models of software development exist.

Which model should be used for software development for a specific project boils down to the preference of the sponsors or clients of the project. If the clients prefer to have a product with minimal (or essential) features at the beginning and launch it into the market and, on the basis of the market response, continue adding additional features later on, then the incremental model is preferred. A software product can be incrementally built using any of the many popular agile methodologies such as eXtreme Programming, Scrum, and Rational Unified Process (used by Eclipse platforms). On the other hand, if the clients want the entire product to be delivered in one shot, then the big project planning model is preferred, better known as the Waterfall model.

There are many factors that need to be considered in deciding which model is best suited for a given project.

Best software engineering techniques and methods:

While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can’t, therefore, say that one method is better than another.

Seven Principles of Software Engineering:

Principles are common and conceptual statements describing desirable properties of software products and processes. Principles become practice through methods and techniques, often methods and techniques are packaged in a methodology. Methodologies can be enforced by tools.

Principles of Software Engineering have a good impact on the process of software engineering and also on the final product. These principles facilitate to develop software in such a manner that it posses all the qualities like: efficiency, functionality, adaptability, maintainability, and usability. Principles are general, abstract statements describing desirable properties of software processes and products. The principles are applicable throughout the lifecycle of the software. Principles are the set of statements which describe the advantageous features of the product and process. Focus on both process and product is needed to deliver software systems. These principles help in controlling process which in turn helps to control the quality of the product. Only the control of process will not guarantee a quality product therefore it is important to concentrate on both process and quality.

As said earlier there are no fundamentally recognized principles of Software Engineering but we can list down few that may be used in all phases of software development. Seven important principles that may be used in all phases of software development

Principle - 1: Rigor and formality:

- ✓ Software engineering is a creative design activity. Creativity often leads to imprecision and inaccuracy. Software development can tolerate neither imprecision nor inaccuracy.
- ✓ Rigor helps to produce more reliable products, to control cost, to increase confidence in products.
- ✓ Formality is rigor at the highest degree. It is software process driven and evaluated by mathematical laws.
- ✓ Examples:

  Rigorous- Rigorous documentation of development steps helps project management and assessment of timeliness.

  Formal- automated transformational process to derive program from formal specifications

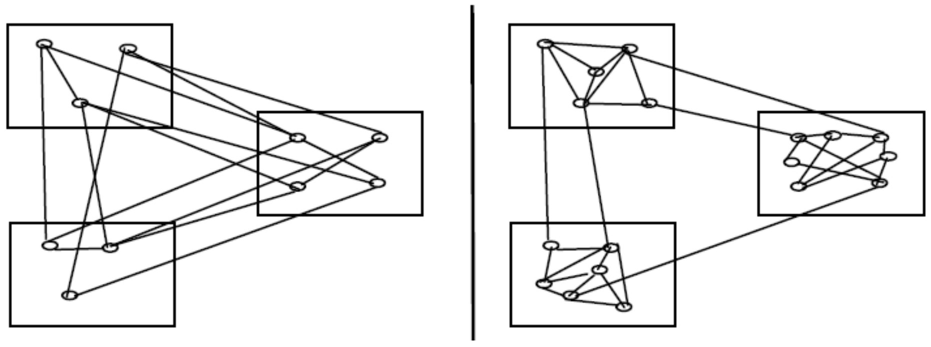
Principle - 2: Separation of concerns:

- ✓ It tackles with the complexity of large systems by separating different issues to concentrate on one at a time (Divide & Conquer)
- ✓ Supports parallelization of efforts and separation of responsibilities.
- ✓ Various types of separation
    - In terms of time
    - In terms of qualities
    - In terms of views of an artifact.
- ✓ Separation of concerns allow separation of responsibilities
    - Separation of managerial and technical issues
    - Separation of requirements and design

Principle - 3: Modularity and decomposition:

- ✓ A complex system may be divided into simpler pieces called modules
- ✓ A system that is composed of modules is called modular It supports application of separation of concerns
- ✓ When dealing with a module we can ignore details of other modules
- ✓ Each separated module should be highly cohesive/ interconnected with the rest of the modules so that each module should be understandable as a meaningful unit. Components of each module should be closely related to one another.
- ✓ Modules should exhibit low coupling means they should have low interactions with other modules.
- ✓ Too much interaction with other modules makes a module dependent on other, thus making it less reusable.

Example:



high coupling low cohesion low coupling high cohesion

Principle - 4: Abstraction

- ✓ Abstraction is a mechanism to' hide irrelevant details and represent only the essential features so that one can focus on important things at a time; It allows managing complex systems by concentrating on the essential features only.

- ✓ For example, while driving a car, a driver only knows the essential features to drive a car such as how to use clutch, brake, accelerator, gears, steering, etc., and least bothers about the internal details of the car like motor, engine, wiring, etc.
- ✓ Abstraction can be of two types, namely, data abstraction and control abstraction. Data abstraction means hiding the details about the data and control abstraction means hiding the implementation details. In object-oriented approach, one can abstract both data and functions.

Principle - 5: Anticipation of change

- ✓ Anticipation of change helps to
  - create a software infrastructure that absorbs changes easily
  - enhance reusability of components
  - control cost in the long runExamples:
- ✓ Not anticipating change often leads to high cost and unmanageable software

  – Software development deals with inherently changing requirements

Principle - 6: Generality

- ✓ Generality leads to
  - increased reusability
  - increased reliability
  - faster development
  - reduced cost
- ✓ Not generalizing often leads to continuous redevelopment of similar solutions. Software development cannot tolerate building the same thing over and over again

Principle – 7: Incremental Development:

- ✓ Incremental development means development in a stepwise fashion. According to this principle a subset of the system is delivered early to get the feedback from the users in the early stages of the development.
- ✓ New features or changes , if any, can thus be added incrementally during the early stages without doing much of changes in the structure of the system.
- ✓ It focuses first, more on the functionality, then turn to performance.
- ✓ Incrementality leads to
  - the development of better products
  - early identification of problems
  - an increase in customer satisfaction

Software Engineering Ethics:

Software products can be misused to harm people. For example, malware can be created to steal money from bank accounts of other people. Similarly, viruses can be created to infect the computer systems of other people. Thus, a software engineer needs to abide by a code of ethics to shy away from such activities. It is quite possible to create powerful software systems that can harm a large number of people. All software engineers should abide by a code of ethics to work for the betterment of humanity and not to harm others.

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 outwith 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).

Rationale for the Code of Ethics:

- Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large. Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems.
- Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession.

Principles of the Software Engineering Code of Ethics and Professional Practice:

In 1999, the Institute for Electrical and Electronics Engineers, Inc. (IEEE) and the Association for Computing Machinery, Inc (ACM) published a code of eight Principles related to the behavior of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles based on IEEE/ACM Code of Ethics:

***Principle 1:* PUBLIC**

Software engineers shall act consistently with the public interest. In particular, software engineers shall, as appropriate:

- Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.
- Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.
- Cooperate in efforts to address matters of grave public concern caused by software, its installation, maintenance, support or documentation.
- Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools.
- Be encouraged to volunteer professional skills to good causes and to contribute to public education concerning the discipline.

***Principle 2:* CLIENT AND EMPLOYER**

Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest. In particular, software engineers shall, as appropriate:

- Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education.
- Use the property of a client or employer only in ways properly authorized, and with the client's or employer's knowledge and consent.
- Keep private any confidential information gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law.
- Identify, document, collect evidence and report to the client or the employer promptly if, in their opinion, a project is likely to fail, to prove too expensive, to violate intellectual property law, or otherwise to be problematic.
- Identify, document, and report significant issues of social concern, of which they are aware, in software or related documents, to the employer or the client.
- Promote no interest adverse to their employer or client, unless a higher ethical concern is being compromised; in that case, inform the employer or another appropriate authority of the ethical concern.

Principle 3: ***PRODUCT***

Software engineers shall ensure that their products and related modifications meet the highest professional standards possible. In particular, software engineers shall, as appropriate:

- Strive for high quality, acceptable cost, and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.
- Ensure that they are qualified for any project on which they work or propose to work, by an appropriate combination of education, training, and experience.
- Ensure that an appropriate method is used for any project on which they work or propose to work.
- Work to follow professional standards, when available, that are most appropriate for the task at hand, departing from these only when ethically or technically justified.
- Ensure that specifications for software on which they work have been well documented, satisfy the users' requirements and have the appropriate approvals.
- Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work and provide an uncertainty assessment of these estimates.
- Ensure adequate testing, debugging, and review of software and related documents on which they work.
- Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.
- Work to develop software and related documents that respect the privacy of those who will be affected by that software.
- Treat all forms of software maintenance with the same professionalism as new development.

Principle 4: ***JUDGMENT***

Software engineers shall maintain integrity and independence in their professional judgment. In particular, software engineers shall, as appropriate:

- Temper all technical judgments by the need to support and maintain human values.
- Only endorse documents either prepared under their supervision or within their areas of competence and with which they are in agreement.
- Not engage in deceptive financial practices such as bribery, double billing, or other improper financial practices.
- Disclose to all concerned parties those conflicts of interest that cannot reasonably be avoided or escaped.

Principle 5: ***MANAGEMENT***

Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance. In particular, those managing or leading software engineers shall, as appropriate:

- Ensure good management for any project on which they work, including effective procedures for promotion of quality and reduction of risk.
- Ensure that software engineers know the employer's policies and procedures for protecting passwords, files and information that is confidential to the employer or confidential to others.
- Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work, and provide an uncertainty assessment of these estimates.
- Attract potential software engineers only by full and accurate description of the conditions of employment.
- Offer fair and just remuneration.
- Not unjustly prevent someone from taking a position for which that person is suitably qualified.
- Ensure that there is a fair agreement concerning ownership of any software, processes, research, writing, or other intellectual property to which a software engineer has contributed.
- Not ask a software engineer to do anything inconsistent with this Code.

Principle 6: ***PROFESSION***

Software engineers shall advance the integrity and reputation of the profession consistent with the public interest. In particular, software engineers shall, as appropriate:

- Extend software engineering knowledge by appropriate participation in professional organizations, meetings and publications.
- Support, as members of a profession, other software engineers striving to follow this Code.
- Take responsibility for detecting, correcting, and reporting errors in software and associated documents on which they work.
- Ensure that clients, employers, and supervisors know of the software engineer's commitment to this Code of ethics, and the subsequent ramifications of such commitment.
- Report significant violations of this Code to appropriate authorities when it is clear that consultation with people involved in these significant violations is impossible, counter-productive or dangerous.

***Principle 7:* COLLEAGUES**

Software engineers shall be fair to and supportive of their colleagues. In particular, software engineers shall, as appropriate:

- Encourage colleagues to adhere to this Code.
- Assist colleagues in professional development.
- Credit fully the work of others and refrain from taking undue credit.
- Give a fair hearing to the opinions, concerns, or complaints of a colleague.
- Assist colleagues in being fully aware of current standard work practices including policies and procedures for protecting passwords, files and other confidential information, and security measures in general.

***Principle 8:* SELF**

Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession. In particular, software engineers shall continually endeavor to:

- Improve their ability to create safe, reliable, and useful quality software at reasonable cost and within a reasonable time.
- Improve their ability to produce accurate, informative, and well-written documentation.
- Improve their knowledge of relevant standards and the law governing the software and related documents on which they work.
- Improve their knowledge of this Code, its interpretation, and its application to their work.
- Not give unfair treatment to anyone because of any irrelevant prejudices.

Other Organizations Governing Software Code of Ethics:

Apart from the IEEE/ACM Code of Ethics, there are other professional organisations that have created ethical codes that are followed by their members. This includes:

1. Institute of certification of computing professionals (ICCP). which was created in 1973 to promote certification and professionalism in the industry (ICCP.org).
2. The Association of Professional Engineers and Geoscientists of British Columbia (APEGBC). which is a licensing and regulatory body.

Ethical Dilemmas:

- ✓ Disagreement in principle with the policies of senior management.
- ✓ Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- ✓ Participation in the development of military weapons systems or nuclear systems.

Some myths and realities of Software:

The development of software requires dedication and understanding on the developers' part. Many software problems arise due to myths that are formed during the initial stages of software development. Unlike ancient folklore that often provides valuable lessons, software myths propagate false beliefs and confusion in the minds of management, users and developers.

Managers, who own software development responsibility, are often under strain and pressure to maintain a software budget, time constraints, improved quality, and many other considerations. Common management myths are listed below.

- **Management Myths**

Myth: If the project is behind schedule, increasing the number of programmers can reduce the time gap.
Reality: Adding more people to a development team won't necessarily speed up the delivery, rather it can slow it down even more. New workers take longer to learn about the project as compared to those already working on the project.

Myth: Using the latest, cutting-edge tools or technology guarantees better results then the quality of the software produced by the company will also be great.
Reality: It is not true because the quality of the software depends moreover on its developers and the techniques and logic used to build it. Although the infrastructure and quality of computers and tools is a factor on which the quality of the software depends, it cannot be completely determined by these factors.

Myth: If the project is outsourced to a third party, the management can relax and let the other firm develop software for them.
Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

- **Customer myths**

Myth*:* Brief requirement stated in the initial process is enough to start development; detailed requirements can be added at the later stages.
Reality: Starting development with incomplete and ambiguous requirements often leads to software failure. Instead, a complete and formal description of requirements is essential before starting development.

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.
Reality: Adding requirements at a later stage often requires repeating the entire development process. Change, when requested after software is in production, can be much more expensive than the same change requested earlier. This is because incorporating changes later may require redesigning and extra resources

Myth: Software with more features is better software.
Reality: It is a complete myth. Efficient and useful features in the software make it better from others.

➢ **Developer Myths:**

Myth: Software development is considered complete when the code is delivered.
Reality: 50% to 70% of all the efforts are expended after the software is delivered to the user.

Myth*:* The success of a software project depends on the quality of the product produced.
Reality: The quality of programs is not the only factor that makes the project successful instead the documentation and software configuration also playa crucial role.

Myth*:* Software engineering requires unnecessary documentation, which slows down the project.
Reality: Software engineering is about creating quality at every level of the software project. Proper documentation enhances quality which results in reducing the amount of rework.

Myth*:* Software quality can be assessed only after the program is executed.
Reality: The quality of software can be measured during any phase of development process by applying some quality assurance mechanism. One such mechanism is formal technical review that can be effectively used during each phase of development to uncover certain errors.

Myth*:* Once the software is created, the job is completed.
Reality: This is the biggest myth of all. Software always requires maintenance throughout its lifetime until it gets retired. There’s always something to fix, change, or improve.