ST. XAVIER'S COLLEGE

MAITIGHAR, KATHMANDU

Software Engineering

Theory Assignment #1.2

Submitted by:

Aashish Raj Shrestha

013BSCCSIT002

Submitted to:

Er. Sanjay K Yadav Lecturer, Department of Computer Science St. Xavier's College	

Date of Submission: 24th August, 2016

Table of Contents

What is Software?

Sometimes abbreviated as **SW** and **S/W**, **software** is a collection of instructions that enable the user to interact with a computer, its hardware, or perform tasks. Software is a general term for the various kinds of programs used to operate computers and related devices [1]. Without software, computers would be useless. For example, without your Internet browser, you could not surf the Internet or read this page and without an operating system, the browser could not run on your computer. The picture to the right shows a Microsoft Excel box, an example of a spreadsheet software program [2].

Software can be thought of as the variable part of a computer and hardware the invariable part. Software is often divided into application software (programs that do work users are directly interested in) and system software (which includes operating systems and any program that supports application software). The term middleware is sometimes used to describe programming that mediates between application and system software or between two different kinds of application software (for example, sending a remote work request from an application in a computer that has one kind of operating system to an application in a computer with a different operating system).

An additional and difficult-to-classify category of software is the *utility*, which is a small useful program with limited capability. Some utilities come with operating systems. Like applications, utilities tend to be separately installable and capable of being used independently from the rest of the operating system [1].

What is Software Engineering?

A software engineer is a licensed professional engineer who is schooled and skilled in the application of engineering discipline to the creation of software.

A software engineer is often confused with a programmer, but the two are vastly different disciplines. While a programmer creates the codes that make a program run, a software engineer creates the designs the programmer implements [3].

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product [4].

IEEE defines software engineering as:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software [5].

Fritz Bauer,

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines [6].

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 documentations [6].

Fairley,

Software Engineering is the methodological and managerial discipline concerning the systematic production and maintenance of software products that are developed and maintained within anticipated and controlled time and cost limits [6].

Somerville,

Software Engineering is concerned with the theories, methods and tools that are needed to develop the software products in a cost effective way [6].

Other Definitions [6],

i. *Software Engineering deals with cost effective solutions to practical problems by applying scientific knowledge in building software artifacts in the service of mankind.*
ii. *Software Engineering is the application of methods and scientific knowledge to create practical cost-effective solutions for design, construction, operation and maintenance of the software.*
iii. *Software Engineering is the 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.*
iv. *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 helps developer's model application designs and then generate code are all closely associated with Software Engineering.*
v. *Software Engineering is an engineering discipline which is concerned with all aspects of software production.*

Software Engineer works for development of procedures and systematic applications that are used on electronic machines. Software engineering incorporates various accepted methodologies to design software. This particular type of engineering has to take into consideration what type of machine the software will be used on, how the software will work with the machine, and what elements need to be put in place to ensure reliability [7].

What is difference between Software Engineering and Computer Science?

The relationship 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 distinction exists [8].

Essentially, computer science is concerned with the theories and methods that underlie computer and software systems, whereas software engineering is concerned with the practical problems of producing software. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers.

Ideally, all of software engineering should be underpinned by theories of computer science, but in reality this is not the case. Software engineers must often use ad hoc approaches to developing the software. Elegant theories of computer science cannot always be applied to real, complex problems that require a software solution [22].

Issues	Software Engineering	Computer Science
Ideal	Constructing software applications for real-world use today	Finding eternal truths about problems and algorithms for prosperity
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 travelling salesman problem)
Budgets and Schedules	Projects (like upgrading an office suite) have fixed budgets and schedules	Projects (like Solving P=NP?) have open end budgets and schedules
Change	Applications evolve as user needs and expectations evolve, and as SE technologies and practices evolve	When computer science problem are solved, the solution will never change
Additional Skills	Domain knowledge	Mathematics
Notable Educators and Researchers	Barry Boehm, Fred Brooks and David Panas	Edsger Dijkstra, Donald Knuth and Aalan Turing
Notable Practitioners	Dan Bricklin, Steve McConnell	Not applicable
Practitioner in all over the world	Above 1,400,000	Above 50,000

What is difference between software engineering and system engineering?

The relationship between Software Engineering and the fields of Traditional engineering (for e.g. System Engineering) and Computer Science have been debated for decades. Software Engineering resembles all of these fields, but important distinction exists [9].

System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process [19].

According to Ian Sommerville, System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is therefore concerned with hardware development, policy and process design and system deployment as well as software engineering. System engineers are involved in specifying the system, defining its overall architecture and then integrating the different parts to create the finished system. They are less concerned with the engineering of the system components (hardware, software, etc.)[23]

System engineering is older discipline than software engineering. People have been specifying and assembling complex industrial systems such as aircraft and chemical plants for more than a hundred years. However, as the percentage of the software in systems has increased, software engineering techniques such as use-case modelling and configuration management are being used in the system engineering process [23].

Issues	Software Engineering	Engineering
Foundations	Based on computer science, information science and discrete math.	Based on science mathematics, and empirical knowledge
Costs	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-overruns can adversely affect the total project cost.	In some projects, constructions 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 downloading 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. Others kind 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 the software	Engineers generally try to apply known and tested principles, and limit the use of

	projects.	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 system engineering: manage manufacturing, or maintenance crew.
Blame	Software engineers must blame themselves for project problems.	Engineers in some fields can often blame construction, manufacturing or maintenance crews for project problems.
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 requirements.

What is software process?

A software process is a structured set of activities required to develop a software system [11]. Process defines a framework for a set of Key Process Areas (KPAs) that must be established for effective delivery of *software engineering* technology. This establishes the context in which technical methods are applied, work

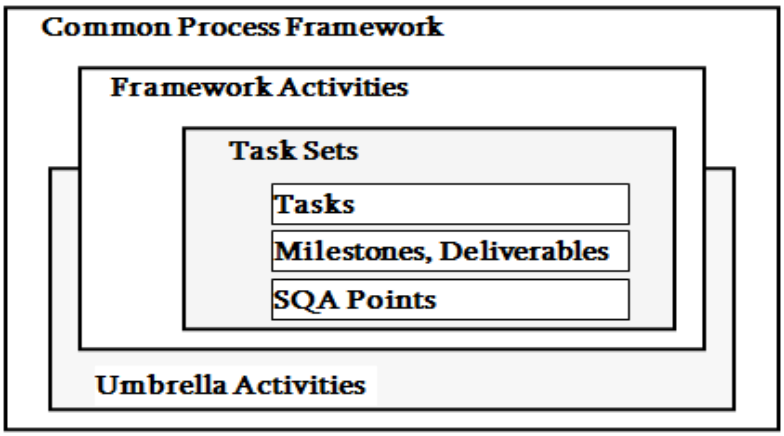

Figure 1.1: Chart of Process Framework

products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed [10].

Software process framework [10][15]: A process framework establishes the foundation for a complete *software process* by identifying a small *number of framework activities* that are applicable to all software projects, regardless of size or complexity. It also includes a set of *umbrella activities* that are applicable across the entire software process. Some most applicable framework activities are described below.

Communication:

This activity involves heavy communication with customers and other stakeholders in order to gather requirements and other related activities.

Planning:

Here a plan to be followed will be created which will describe the technical tasks to be conducted, risks, required resources, work schedule etc.

Modeling:

A model will be created to better understand the requirements and design to achieve these requirements.

Construction:

Here the code will be generated and tested.

Deployment:

Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

These above described five activities can be used in any kind of *software development*. The details of the software development process may become a little different, but the framework activities remain the same.

Like Above there are many different software processes but all involve [11]:

♣ Specification – defining what the system should do;
♣ Design and implementation – defining the organization of the system and implementing the system;
♣ Validation – checking that it does what the customer wants;
♣ Evolution – changing the system in response to changing customer needs.

What is software process model?

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective [11].

A Process Model describes the sequence of phases for the entire lifetime of a product. Therefore it is sometimes also called Product Life Cycle. This covers everything from the initial commercial idea until the final de-installation or disassembling of the product after its use.

Usually there are three main phases [12]:

- concept phase
- implementation phase
- maintenance phase

Each of these main phases usually has some sub-phases, like a requirements engineering phase, a design phase, a build phase and a testing phase. The sub-phases may occur in more than one main phase each of them with a specific peculiarity depending on the main phase.

Besides the phases a Process Model shall also define at least:

- The **activities** that have to be carried out in each of the sub-phases, including the sequence in which these activities have to be carried out.
- The **roles** of the executors that have to carry out the activities, including a description of their responsibilities and required skills.
- The **work products** that have to be established or updated in each of the activities. Besides the final product there are usually several other items that have to be generated during the development of a product. These are for example requirements and design document, test specifications and test reports, etc.

Therefore, a Process Model provides a fixed framework that guides a project in:

- Development of the product
- Planning and organizing the project
- Tracking and running the project

Software Process Models [11]:

♣ The waterfall model

- Plan-driven model. Separate and distinct phases of specification and development.

♣ Incremental development

- Specification, development and validation are interleaved. May be plan-driven or agile.

♣ Reuse-oriented software engineering

- The system is assembled from existing components. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

Also some examples of the types of software process model that may be produced are [24]:

1. A Workflow model: This shows the sequence of activities in the process along with their inputs, outputs and dependencies. The activities in this model represents human actions.
2. A dataflow or activity model: This represents the process as a set of activities, each of which carries out some data transformation. It shows how the input to the process, such as specification, is transformed to an output, such as a design. The activities here may represent transformations carried out by people or by computers.
3. A role/action model: This represents the roles of the people involved in the software process and the activities for which they are responsible.

Most of the models are based on one of three general models or paradigms of software development [24]:

1. The Waterfall Approach: This takes the above activities and represent them as separate process phase such as requirements specification, software design, implementation, testing and so on. After each stage is defined it is 'signed-off', and development goes on to the following stage.
2. Iterative Development: This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from very abstract specifications. This is then refined with customer input to produce a system that satisfies the customer's needs. The system may then be delivered. Alternatively, it may be re-implemented using a more structured approach to produce a more robust and maintainable system.
3. Component-based Software Engineering (CBSE): This techniques assumes that parts of the system already exist. The system development process focuses on integrating these parts rather than developing them from scratch.

What are the costs of software engineering?

Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs [19].

Software Cost [17]: Despite the terminology, software engineering cost does not refer directly to monetary value associated with software development. Such a value is almost impossible to arrive at and not always useful. The questions are "What's the effort involved?" and "How long will it take?" The answer to these questions can then be translated to the monetary value. This leads to the following definition of software cost.

Software cost consists of three elements:

- Manpower loading is the number of engineering and management personnel allocated to the project as a function of time.
- Effort is defined as the engineering and management effort required to complete a project, usually measured in units such as person-month. The types and the levels of skills for the resources influence the cost of the project.
- Duration is the amount of the time (usually measured in months) required to complete the project.

Arriving at the cost estimate involves using a number of different factors to try to determine the overall cost of a system. Deciding which factors to include and combine them to arrive at the estimate make up the engineering cost estimation process that is defined as follows:

Direct costs includes items such as analysis, design, coding, testing and integration. Depending on who is doing the engineering and why software cost may also include a number of other items such as training, customer support, installation, level of documentation, configuration management, and quality assurance.

What is CASE (Computer Aided Software Engineering)?

According to IEEE, Definition of CASE is: The use of computers to aid in the software engineering process. May include the application of software tools to software design, requirements tracing, code production, testing, document generation, and other software engineering activities [16].

The acronym CASE stands for Computer Aided Software Engineering. It covers a wide range of different types of program that are used to support software process activities such as requirements analysis, system modelling, debugging and testing. All methods now come with associated CASE technology such as editors for the notations used in the method, analysis modules which check they system model according to the method rules and report generators to help create system documentation. The CASE tools may also include a code generator that automatically generates source code from the system model and some process guidance for software engineers [25].

What are the attributes of good software?

A software component is a system element offering a predefined service able to communicate with other components [14]. Clemens Szyperski and David Messerschmitt give the following five criteria for what a software component shall be to fulfill the definition of good software.

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

In order for the software to be a good software it should have following characteristics:

The key characteristics of software are as under [14]:

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 exists 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 and 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 products.
3. **Software is flexible.** We all feel the software is flexible. A program can be developed to do almost anything. Sometimes, this characteristics 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 well known "bath-tub curve" in reliability studies for the hardware products. Figure 1.2 depicts failure rate as a function of time for hardware. The relationship, often called the "bath-tub curve".

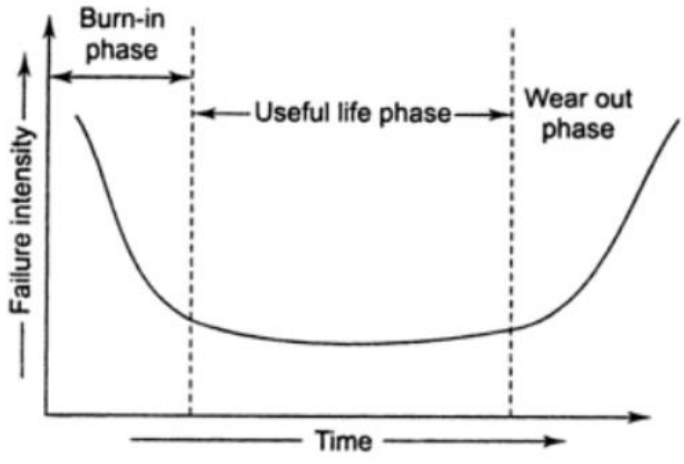

Figure 1.2: Bath-Tub Curve

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 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 Figure 1.3

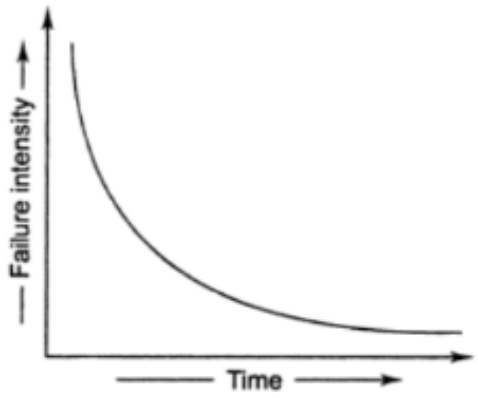

Figure 1.3: Software Curve

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.

Attributes of a good software:

Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable [19].

Basically there are four attributes of a good software.

1. **Maintainability:** The software should be written in a way that it can be evolved to meet changing needs of the customer. Since the change of needs (requirement change of customer) is unavoidable, this is a critical attribute.
2. **Dependability:** A software must be trustworthy (Can we trust the software?), reliable, safe and secured (Are we secured using this software?). Use of the software should not be harmful to the customer in anyway (even something goes wrong). Even at a system failure, a dependable software should not cause physical or economic damage. Also malicious users should not be able to access or damage the system.
3. **Efficiency:** A software should be efficient in every way. The software should not make wasteful of system resources (ex. Memory, processing cycles). Responsiveness should be there, and the memory utilization should be minimum and also the processing time is a consideration.
4. **Acceptability:** The software must be acceptable to the group of users for which it's designed for. Software should be understandable, reliable and compatible with other systems they use.

What are the key challenges facing software engineering?

There are some unique and pressing issues to deal with in the software industry. Several of these are now discussed:

• **Tractable Medium:** Engineer is a logical and tractable medium, not physical medium. The constraints of physical medium can serve to simplify alternatives. For example, in a house design you can't put a kitchen and a bathroom in the same place; batteries have standard voltages. Frederick Brooks, notable software engineer and author of the legendary book The Mythical Man Month, expresses an analogy,

The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination [18].

This tractability has its own pros and cons. On the positive side, as programmers we have the ultimate creative environment. We can create grandiose programs chock full of beautiful algorithms and impressive user interfaces. And we can completely change this functionality or the look of the interface in mere seconds and have a new creation! Conversely, because we are only dealing with "thought-stuff," our profession has a limited scientific and/or mathematical basis. In other fields, the scientific and mathematic basis of physical, intractable mediums constrain the solution to a problem -- only certain materials can withstand the weight of a car, only certain paints can take the intensity of the UV rays on the top of a mountain, etc. With software, the sky's the limit!

Quite often programmers are also asked to fix hardware product problems because people think that it is cheaper to fix the problems in the (tractable) software than it is to re-design and re-manufacture physical parts. This presents software engineers with the need to design and coding changes, often at the last minute.

The software industry has been trying to formulate a sort of scientific/mathematical basis for itself. Formal notations have been proposed to specify a program; mathematical proofs have been defined using these formal notations. The software community is also establishing analysis and design patterns. (Gamma, 1995; Fowler, 1997) These patterns are general solutions to recurring analysis and design problems; the patterns are proposed, proven and documented by experts in the field. Engineers can become familiar with these general solutions and learn to apply them appropriately in the systems and programs under development.

• **Changing requirements:** Adapting for hardware changes is only one source of requirements churn for software engineers. Unfortunately, requirements changes come from many sources. It is often very hard for customers to express exactly what they want in a product (software is only thought-stuff for them too!). They often don't know what they want until they see some of what they've asked for. Requirements analysts may not understand the product domain as completely as they need to early in the product lifecycle. As a result, the analysts might not know the right questions to ask the customer to elicit all their requirements. Lastly, the product domain can be constantly changing during the course of a product development cycle. New technology becomes available. Competitors release new products

that have features that weren't thought of. Innovators think of wonderful new ideas that will make the product more competitive.

• **Schedule Optimism:** Software engineers are an optimistic crew. In most organizations, it is the software engineers who estimate how long it will take to develop a product. No matter how many times we've taken longer than we thought in the past, we still believe "Next time, things will go more smoothly. We know so much more now." As a result, we often end up committing to a date we have no business committing to, giving the software industry a "never on time" reputation.

• **Schedule Pressure:** We often make these aggressive commitments because of the intensity of the people asking us for commitment. It seems that every product is late before it's even started, every feature is critical or the business will fold. Products need to be created and updated at a constant, rapid pace lest competitors take over the business.

What are the Software Engineering Methods?

Software engineering is gravely hampered today by immature practices. Specific problems include:

- The prevalence of fads more typical of fashion industry than of an engineering discipline.
- The lack of a sound, widely accepted theoretical basis.
- The huge number of methods and method variants, with differences little understood and artificially magnified.
- The lack of credible experimental evaluation and validation.
- The split between industry practice and academic research.

Software engineering methodology should be based on a solid theory, proven principles and best practices that [1]:

- Include a kernel of widely-agreed elements, extensible for specific uses
- Addresses both technology and people issues
- Are supported by industry, academia, researchers and users
- Support extension in the face of changing requirements and technology

A software engineering method is a structured approach to software development whose aim is to facilitate the production of high-quality software in a cost-effective way. Methods such as Structured Analysis (DeMarco, 1978) and JSD (Jackson, 1983) were first developed in the 1970s. These methods attempted to identify the basic functional components of a system; function oriented methods are still used. In the 1980s and 1990s, these function-oriented methods were supplemented by object-oriented (OO) methods such as those proposed by Booch (Booch, 1994) and Rumbaugh (Rumbaugh, 1991). These different approaches have now been integrated into a single unified approach built around Unified Modeling Language (UML) [27].

Also 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 [19].

Professional and Ethical Responsibility of Software Engineering

Like other engineering disciplines, software engineering is carried out within a legal and social framework that limits the freedom of engineers. Software engineers must accept that their job involves wider responsibilities than simply the application of technical skills. They must also behave in an ethical and morally responsible way if they are to be respected as professionals.

It goes without saying that you should always uphold standards of honesty and integrity. You should not use your skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession. However, there are areas where standards of acceptable behavior are not bounded by laws but by the more tenuous notion of professional responsibility. Some of these are [26]:

1. Confidentiality: You should normally respect the confidentiality of your employers or clients irrespective of whether a formal confidentiality agreement has been signed.
2. Competence: You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. Intellectual property rights: You should be aware of local laws governing the use of intellectual property such as patents and copyrights. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. Computer Misuse: You should not use your 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).

Professional societies and institutions have an important role to play in setting ethical standards. Organizations such as the ACM, the IEEE (Institute of Electrical and Electronic Engineers) and the British Computer Society publish a code of professional conduct or code of ethics. Members of these organizations undertake to follow that code when they sign up for membership. These code of conduct are generally concerned with fundamental ethical behavior.

The ACM and IEEE have cooperated to produce a joint code of ethics and professional practice. This code exists in both a short form and a longer form that adds detail and substance to the shorter version. The rationale behind this code is summarized in the first two para graph of the longer form.

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

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:

1. PUBLIC – Software engineers shall act consistently with the public interest.
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.
3. PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT – Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.
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.

Software Engineering Ethics [19]:

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

Issues of Professional Responsibility [19]:

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is out with their competence.
- Intellectual property rights

        - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
    - Computer misuse
        - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

Ethical Dilemmas [19]:

- 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.

Software Engineering is an engineering discipline that is concerned with all aspect of software production. Why?

As a discipline, software engineering has progressed very far in a very short period of time, particularly when compared to classical engineering field (like civil or electrical engineering). In the early days of computing, not much more than 50 years ago, computerized systems were quite small. Most of the programming was done by scientists trying to solve specific, relatively small mathematical problems. Errors in those systems generally had only "annoying" consequences to the mathematician who was trying to find "the answer." Today we often build monstrous systems, in terms of size and complexity. What is also notable is the progression in the past 50 years of the visibility of the software from mainly scientists and software developers to the general public of all ages. "Today, software is working both explicitly and behind the scenes in virtually all aspects of our lives, including the critical systems that affect our health and well-being." [21]

Despite our rapid progress, the software industry is considered by many to be in a crisis. Some 40 years ago, the term "Software Crisis" emerged to describe the software industry's inability to provide customers with high quality products on schedule. "The average software development project overshoots its schedule by half; larger projects generally do worse. And, some three quarters of all large systems are "operating failures" that either do not function as intended or are not used at all." (Gibbs, 1994) While the industry can celebrate that software touches nearly all aspects of our daily lives, we can all relate to software availability dates (such as computer games) as moving targets and to computers crashing or locking up. We have many challenges we need to deal with as we continue to progress into a more mature engineering field, one that predictably produces high-quality products

Software Engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use. In this definition there are two key phrases [20]:

1. Engineering discipline: Engineers makes things work. They apply theories, methods and tools where these are appropriate, but they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work to organizational and financial constraints, so they look solutions within these constraints.
2. All aspects of software production: Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production.

In general, software engineers adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software. However engineering is all about selecting the most appropriate method for a set of circumstances and a more creative, less formal approach to development may be effective in some circumstances. Less formal development is particularly appropriate for the development of web-based systems, which requires blend of software and graphical design skills.

Discuss weather professional engineers should be certified in the same way as doctors or lawyers?

The economies of all developed nations are dependent on software and more and more systems are software controlled (like aircraft, rocket launchers etc.) in the developed countries. Since software engineering is concerned with theories, methods and tools for the professional software development and to find the solution of the problem, expenditure of the software represents a significant fraction of GNP (Gross National Product) in all developed countries [19].

Saying that more and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly and that is a hell out of a job and for that, professional engineer have to be very good at what they do. So if we have to maintain the standard for engineers to keep the quality of the work we should do it. We should analyze and certify the professional engineers as their work is felt important in the today's era.

What are the differences between Generic Software Product Development and Custom Software Product Development?

Firstly in defining the differences between generic and custom software it is important to define what generic and custom software actually is; according to Dr A Capiluppi (2008, slide 5) generic software can be defined as: 'stand-alone systems produced for an open market – production controlled by the development organization', whereas custom software can be defined as: 'commissioned systems by a particular customer – production controlled by the customer organization'. Now that these two types of software have briefly been defined, a comparison between them is that; generic software development differs from custom software development mainly in the intended users and functions for those users that will be contained within the software. Generic software has to include as many functions as possible as it will have a very wide use base, and will need to provide functionality and usability to that wider range of users. However because of the wide use base of generic software it also needs to be quite concise with the functions that are included; as having too many functions available to users, who may not be the best experts on computers, would probably affect the ease of use of the software. Generic software also has to be developed to be very reusable and have components that can be easily modified or added to at a later date, making the software as upgradable as possible so it can have a longer useful lifetime [28].

The development of custom made software differs from this because the uses of the software are for a lot more specific of a customer base, and usually would include components that would need to be freshly written for the software; as if generic components already existed to perform the requirements of the users then they would have brought off the shelf software. Custom software development is usually a lot more based around communication with the specific needs of the users the software is being created for, to insure the finish product will meet the needs of the end user, whilst still being made of separate components that make the software as upgradable as possible. However as custom software is usually developed by one company/group of developers for a specific user, the upgradability and maintainability of the software will always be limited as if the knowledge from the original creators is lost, possibly through hiring of new IT support staff, then the new staff would take a long time to get to grips with the custom software and would risk damage to it when making modifications or upgrades to it at a later date [28].

What are the differences between Software Process Model and Software Process? Suggest two way to development in which a software process model might be helpful in identifying possible process improvements?

- A software process is what actually goes on when software is developed.
- A software process model is an abstraction and simplification of a process.
- A software process may contain various activities such as :
    - Specification
    - Design and Implementation
    - Validation
    - Evolution
- A software process model contains various models/paradigms such as:
    - Waterfall
    - Evolutionary Development
    - Formal Transformation
    - Integration from useful components
- Process models can be used to help understand real processes and to identify which aspects of these processes could be supported by CASE tools.
- A software process model is an abstract representation of a process methodology. Waterfall[1] is a process model. Agile is a process model. They don't specify how to do things, but outline the types of things that are done. For example, Waterfall identifies the phases that a project goes through - requirements, design, implementation/unit testing, integration testing, system testing, deployment - without saying what artifacts to produce or what tools to use (although the output of code is implied). Agile defines core values in the form of the Agile manifesto, time-boxed iterations, and continuous response to change, but it doesn't say how long your iterations should be or how you go about responding to change. The Spiral model is a third software process model.

References

[1] "Software", searchsoa.techtarget.com, 2016. [Online]. Available: http://searchsoa.techtarget.com/definition/software [Accessed: 20- August- 2016].

[2] "Software", computerhope.com, 2016. [Online]. Available: http://www.computerhope.com/jargon/s/software.htm [Accessed: 20- August- 2016].

[3] "Software engineer", webopedia.com, 2016. [Online]. Available: http://www.webopedia.com/TERM/S/software_engineer.html [Accessed: 20- August- 2016].

[4] "Software Engineering Overview", tutorialspoint.com, 2016. [Online]. Available: http://www.tutorialspoint.com/software_engineering/software_engineering_overview.htm [Accessed: 20- August- 2016].

[5] "IEEE standard glossary of software engineering terminology - IEEE Std 610.12-1990", idi.ntnu.no, 2016. [Online]. Available: http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf [Accessed: 20- August- 2016].

[6] Bharat B. Agrawal and Sumit P. Tayal,"Introduction to Software Engineering," in *Software Engineering, 1st* Ed. New Delhi, 2007, pp. 5-6.

[7] "software engineering: definition", www.businessdictionary.com, 2016. [Online]. Available: http://www.businessdictionary.com/definition/software-engineering.html [Accessed: 20- August- 2016].

[8] Bharat B. Agrawal and Sumit P. Tayal,"*COMPARISION OF SOFTWARE ENGINEERING AND RELATED FIELD*," in *Software Engineering, 1st* Ed. New Delhi, 2007, pp. 16-17.

[9] Bharat B. Agrawal and Sumit P. Tayal,"*COMPARISION OF SOFTWARE ENGINEERING AND RELATED FIELD*," in *Software Engineering, 1st* Ed. New Delhi, 2007, pp. 17-19.

[10] Roger S. Pressman,"The *Software Process*," in *Software Engineering: A Practitioner's Approach, 7th* Ed. NY, 2010, pp. 14-15.

[11] "Ch2.pptx", cs.ccsu.edu, 2016. [Online]. Available: http://www.cs.ccsu.edu/~stan/classes/CS530/Slides11/Ch2.pdf [Accessed: 20- August- 2016].

[12] "What is a Software Process Model?", the-software-experts.com, 2016. [Online]. Available: http://www.the-software-experts.com/e_dta-sw-process-model.php [Accessed: 20- August- 2016].

[13] "SEMAT - Software Engineering Method and Theory", infoq.com, 2016. [Online]. Available: https://www.infoq.com/news/2010/04/semat [Accessed: 20- August- 2016].

[14] Bharat B. Agrawal and Sumit P. Tayal,"*SOFTWARE COMPONENTS*," in *Software Engineering, 1st* Ed. New Delhi, 2007, pp. 7-8.

[15] "What is software process and software process framework? Explain.", onlineclassnotes.com, 2016. [Online]. Available: http://www.onlineclassnotes.com/2013/01/what-is-software-process-and-software.html [Accessed: 20- August- 2016].

[16] "IEEE SE Glossary-610.12-1990", idi.ntnu.no, 2016. [Online]. Available: http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf [Accessed: 20- August- 2016].

[17] "Ladeira", ujdigispace.uj.ac.za, 2016. [Online]. Available: http://ujdigispace.uj.ac.za/bitstream/handle/10210/4500/Ladeira.pdf?sequence=3&isAllowed=y [Accessed: 20- August- 2016].

[18] Frederick P Brooks, *The Mythical Man Month,* 20th Anniversary Ed. Addison Wesley 1995

[19] "Stan_Classes-CS530", cs.ccsu.edu, 2016. [Online]. Available: http://www.cs.ccsu.edu/~stan/classes/cs530/slides11/ch1.pdf [Accessed: 20- August- 2016].

[20] Ian Sommerville, "What is software Engineering?" *in Software Engineering*, 8th Ed. Boston, Massachusetts 02116, 2009, pp. 31-41

[21] Shari Lawrence Pfleeger, *Software Engineering: Theory and Practice, 2010*

[22] Ian Sommerville, "What's the difference between software engineering and computer science?" *in Software Engineering*, 8th Ed. Boston, Massachusetts 02116, 2009, pp. 31

[23] Ian Sommerville, "What's the difference between software engineering and system engineering?" *in Software Engineering*, 8th Ed. Boston, Massachusetts 02116, 2009, pp. 31-32

[24] Ian Sommerville, "What is a software process model?" *in Software Engineering*, 8th Ed. Boston, Massachusetts 02116, 2009, pp. 32-33

[25] Ian Sommerville, "What is CASE?" *in Software Engineering*, 8th Ed. New Delhi, 2009, pp. 36

[26] Ian Sommerville, "Professional and ethical responsibility," *in Software Engineering*, 8th Ed. Boston, Massachusetts 02116, 2009, pp. 36

[27] Ian Sommerville, "What are software engineering methods?" *in Software Engineering*, 8th Ed. Boston, Massachusetts 02116, 2009, pp. 35

[28] Dr Andrea Capiluppi. 2008. *lecture 01- Characteristics of Software.*[Online] University of Lincoln blackboard site. Available at: http://blackboard.lincoln.ac.uk/webapps/blackboard/content/listContent.jsp?mode=reset&course_id=_25772_1&content_id=_141234_1#_209998_1 [Accessed: 3- August- 2016].