***CONTENT***

1. Abstract
2. Introduction
	1. Software
	2. Engineering
	3. Software Engineering
3. History of Software Engineering
4. Why do we need Software Engineering?
5. Soft wares Used in Different Spheres :
	1. Bloomberg: Banking
	2. ArchiCAD: Architecture
	3. Oracle: Database Management
	4. Reference Software
	5. Antivirus Software: Computer Security
	6. Global Positioning System (GPS): Navigation Software
6. Software Development Processes
	1. Waterfall Model
	2. Rapid Prototyping Model
	3. Incremental Model
	4. Extreme Programming
	5. Spiral Model
	6. Rational Unified Process
	7. Choosing A Process Model
7. Importance of Software Engineering
	1. As a Profession
	2. As a Field of Education
8. Conclusion
9. Glossary

**ABSTRACT**

The title of the term paper is Software Engineering. This report deals with a very limited area of the topic starting with the definition of software, engineering and Software Engineering, followed by the past or the History of Software Engineering from its foundation leading to the development of different software to its present day state. From the first digital computers that appeared in the early 1940s to the development of the now famous Personal Computer (PC). Extract from “Why do we need of software engineering” an article by Edeh Chijioke an expert author about what lead to the requirement of Software Engineering and advancement in this field. Different softwares which are now a days used in the different spheres like Banking, Architecture, Database Management, Computer Security, etc. with their origin and uses. Softwares like ArchiCAD (Architectural Software), Oracle (Database Management Software), Reference Software, Accounting software (Software used in Banking), Antiviruses (Software used to secure the computer system from malware/harmful programs), GPS (Navigation Software), etc. This report also gives an overview of Software Development Process with the advantages and disadvantages of few of the processes like Build & Fix, Waterfall, Spiral, eXtreme Programming (XP), etc. Leading to the importance of Software Engineering as a profession, and as a field of Education in today’s world.

 **INTRODUCTION**

To understand what Software Engineering is, we first need to understand what software is and what is engineering in them.

SOFTWARE: Software, or Computer Software, is a set of instructions in machine language or most often in the form of a computer program that direct a computer's processor to perform specific functions/operations. It is a kind of interface between the user and computer hardware (Ex: Operating System). It interacts with the machine/hardware or the physical objects (processor and related devices) that carry out the instructions. Hardware and software require each other and neither has any value without the other. As hardware is useless without the software and also the software cannot function on its own without the hardware. Software is a very general term. It can refer to all computer instructions in general, or to any specific set of computer instructions directed to do a specific task. It comprises of both the machine instructions (the binary/machine language code that the processor understands) and source code (more human-understandable instructions that must be converted into machine code by compilers or interpreters before being executed as the processer understands only the machine language).

Software can broadly be classified in to three categories:

* System Software: It is computer software designed to operate and control the computer hardware and to provide a platform for running application software.

For example: Operating Systems like Microsoft Windows, Mac OS X and Linux.

* Application Software: It is all the computer software that causes a computer to perform specific tasks/operations (compare with computer viruses) except the running of the computer itself. A specific instance/object of such software is called a software application, application or app.

For example: accounting software, enterprise software, graphics software, media players, and office suites.

* Embedded Software: It is computer software, written to control machines or devices that are not typically thought of as computers. It is specially designed for the particular hardware/device on which it runs and has time and memory restrictions.

For example: electronics in cars, telephones, modems, robots, appliances, toys, security systems, pacemakers, televisions and set-top boxes, and digital watches.

ENGINEERING: Engineering is the application of scientific, economic, social, and practical knowledge in order to design, build, and maintain structures, machines, devices, systems, materials and processes. It may also include using understandings to comprehend, model and scale an appropriate solution to a problem or objective. The discipline of engineering is extremely broad, and covers a range of more specialized fields of engineering, each with a more specific emphasis on a particular area of technology and types of application.

Chemical, Civil, Electrical, Mechanical, Computer Science (Computer & Software), Aeronautical are few of the most popular engineering fields.

SOFTWARE ENGINEERING: Therefore, Software Engineering is the application of the instructions in the form of a computer program that direct a computer's processor to perform specific functions/operations.

‘Or’

Software Engineering (SE) is the application of a systematic and disciplined approach to the design, development, operation, and maintenance of software. And the study of these approaches is the application of engineering to software. Software development is a much used and more generic term but does not necessarily include the engineering model.

And the one who applies the principles of engineering to the design, development, maintenance, testing, and evaluation of the software and systems that make computers or anything contain software work is a Software Engineer.

**HISTORY OF SOFTWARE ENGINEERING**

When the first digital computers appeared in the early 1940s, the instructions made to them to operate were wired into the machine. Users quickly realized that this design was not flexible and came up with the von Neumann architecture or the "stored program architecture". Hence the division between "hardware" and "software" began with the concept being used to deal with the complexity of computing.

Programming languages started to appear in the 1950s and this was also another major step in this notion. Major languages such as FORTRAN, ALGOL, and COBOL were released in the late 1950s to deal with scientific, algorithmic, and business problems respectively. E.W. Dijkstra wrote his significant paper, "Go to Statement Considered Harmful", in 1968 and David Parnas introduced the key concept of modularity and information hiding in 1972 to help programmers deal with the increasing complexity of software systems. A software system for managing the hardware is called an operating system was also introduced, by UNIX in 1969. In 1967, the Simula language also introduced the object-oriented programming standard.

These advances in software were met with more advances in computer hardware. In the mid-1970s, the microcomputer was introduced, making it economical for hobbyists to obtain a computer and programme for it. This in turn led to the now famous Personal Computer (PC). The Software Development Life Cycle or SDLC was also starting to appear as an agreement for centralized construction of software in the mid-1980s. The late 1970s and early 1980s saw the introduction of several new Simula - inspired object-oriented programming languages, including Smalltalk, Objective - C and C++.

For decades, solving the software crisis was supreme to researchers and for companies producing software tools. The cost of owning and maintaining software in the 1980s was twice as expensive as developing the software. During the 1990s, the cost of ownership and maintenance increased by 30% over the 1980s. In 1995, statistics showed that half of surveyed development projects were operational, but were not considered successful. The average software project overshoots its schedule by half. Three-quarters of all large software products delivered to the customer are failures that are either not used at all, or do not meet the customer’s requirements.

 Open-source software started to appear in the early 90s in the form of Linux and other software introducing the "bazaar" or distributed style of constructing software. Then the World Wide Web and the universalization of the Internet hit in the mid-90s, changing the engineering of software once again. Distributed systems gained dominance as a way to design systems, and the Java programming language was also introduced with its own virtual machine as another forward step in this notion. Programmers collaborated and wrote the Agile Manifesto, which favored more lightweight processes to create cheaper and well-timed software.

The rise of the Internet led to very rapid growth in the demand for international information display/e-mail systems on the World Wide Web. Programmers were required to handle illustrations, maps, photographs, and other images, plus simple animation, at a rate never before seen, with few well-known methods to optimize image display/storage (such as the use of thumbnail images).

The growth of browser usage, running on the HTML language, changed the way in which information-display and retrieval was organized. The widespread network connections led to the growth and prevention of international computer viruses on MS Windows computers, and the vast proliferation of spam e-mail became a major design issue in e-mail systems, flooding communication channels and requiring semi-automated pre-screening. Keyword-search systems evolved into web-based search engines, and many software systems had to be re-designed, for international searching, depending on search engine optimization (SEO) techniques. Human natural-language translation systems were also needed to attempt to translate the information flow in multiple foreign languages, with many software systems being designed for multi-language usage, based on design concepts from human translators. Typical computer-user bases went from hundreds, or thousands of users, to, often, many-millions of international users.

The current definition of software engineering is still being debated by practitioners today as they struggle to come up with ways to produce software that is "cheaper, better, faster". Cost reduction has been a primary focus of the IT industry since the 1990s. Total cost of ownership represents the costs of more than just acquisition. It includes things like productivity impediments, upkeep efforts, and resources needed to support infrastructure.

With the expanding demand for software in many smaller organizations, the need for inexpensive software solutions led to the growth of simpler, faster methodologies that developed running software, from requirements to deployment, quicker & easier.

**WHY DO WE NEED SOFTWARE ENGINEERING?**

To understand the necessity for software engineering, we must briefly look back at the history of computing. This history will help us to understand the problems that started to become obvious in the late sixties and early seventies, and the solutions that have led to the creation of the field of software engineering. These problems were referred to by some as "The Software Crisis," so named for the symptoms of the problem. The situation might also been called "The Complexity Barrier," so named for the primary cause of the problems. Some refer to the software crisis in the past tense. The crisis is far from over, but thanks to the development of many new techniques that are now included under the title of software engineering, the field has made and is continuing to make progress.

In the early days of computing the primary concern was with building or acquiring the hardware. Software was almost expected to take care of itself. The consensus held that "hardware" is "hard" to change, while "software" is "soft," or easy to change. According, most people in the industry carefully planned hardware development but gave considerably less forethought to the software.

The cost of software amounted to such a small fraction of the cost of the hardware that no one considered it very important to manage its development. Everyone, however, saw the importance of producing programs that were efficient and ran fast because this saved time on the expensive hardware.

This approach proved satisfactory in the early days of computing, when the software was simple. However, as computing matured, programs became more complex and projects grew larger whereas programs had since been routinely specified, written, operated, and maintained all by the same person, programs began to be developed by teams of programmers to meet someone else's expectations.

Individual effort gave way to team effort. Communication and coordination which once went on within the head of one person had to occur between the heads of many persons, making the whole process very much more complicated. As a result, communication, management, planning and documentation became critical.

As programs became more complex, the early methods used to make blueprints (flowcharts) were no longer satisfactory to represent this greater complexity. And thus it became difficult for one person who needed a program written to convey to another person, the programmer, just what was wanted, or for programmers to convey to each other what they were doing. In fact, without better methods of representation it became difficult for even one programmer to keep track of what he or she is doing.

The times required to write programs and their costs began to exceed to all estimates. It was not unusual for systems to cost more than twice what had been estimated and to take weeks, months or years longer than expected to complete. The systems turned over to the client frequently did not work correctly because the money or time had run out before the programs could be made to work as originally intended. Or the program was so complex that every attempt to fix a problem produced more problems than it fixed. As clients finally saw what they were getting, they often changed their minds about what they wanted. At least one very large military software systems project costing several hundred million dollars was abandoned because it could never be made to work properly.

The quality of programs also became a big concern. As computers and their programs were used for more vital tasks, like monitoring life support equipment, program quality took on new meaning. Due to the increase in dependency on computers for day to day work, it was important that they work correctly.

Making a change within a complex program turned out to be very expensive. Often even to get the program to do something slightly different was so hard that it was easier to throw out the old program and start over. This, of course, was costly. Part of the evolution in the software engineering approach was learning to develop systems that are built well enough the first time so that simple changes can be made easily.

At the same time, hardware was growing ever less expensive. Tubes were replaced by transistors and transistors were replaced by integrated circuits until microcomputers costing less than three thousand dollars have become several million dollars. As an indication of how fast change was occurring, the cost of a given amount of computing decreases by one half every two years. Given this realignment, the times and costs to develop the software were no longer so small, compared to the hardware, that they could be ignored.

As the cost of hardware plummeted, software continued to be written by humans, whose wages were rising. The savings from productivity improvements in software development from the use of assemblers, compilers, and data base management systems did not proceed as rapidly as the savings in hardware costs. Indeed, today software costs not only can no longer be ignored, they have become larger than the hardware costs. Some current developments, such as nonprocedural (fourth generation) languages and the use of artificial intelligence (fifth generation), show promise of increasing software development productivity, but this is just the beginning of their potential.

Another problem was that in the past programs were often prepared before it was fully understood what the program needed to do. Once the program had been written, the client began to express dissatisfaction. And if the client is dissatisfied, ultimately the producer, too, was unhappy. As time went by software developers learned to layout with paper and pencil exactly what they intended to do before starting. Then they could review the plans with the client to see if they met the client's expectations. It is simpler and less expensive to make changes to this paper-and-pencil version than to make them after the system has been built. Using good planning makes it less likely that changes will have to be made once the program is finished.

**SOFTWARES USED IN VARIOUS SPHERES**

BLOOMBERG: Bloomberg L.P. is a privately held financial software, data and media company headquartered in New York City. Bloomberg L.P. was founded by Michael R. Bloomberg in 1981 with the help of Thomas Secunda, Duncan MacMillan, Charles Zegar and a 30% ownership investment by Merrill Lynch. Bloomberg L.P. provides financial software tools such as an analytics and equity trading platform, data services and news to financial companies and organizations through the Bloomberg terminal (via its Bloomberg Professional Service), its core money-generating product. Bloomberg L.P. also includes a wire service, a global television network, a radio station, websites, subscription-only newsletters and two magazines, Bloomberg Business week and Bloomberg Markets. Used in Banking Sector. Other software used in this sector are ATMs, Accounting Software, Banking Software,

ArchiCAD: ArchiCAD is architectural BIM CAD software for Macintosh and Windows developed by the Hungarian company Graphisoft. ArchiCAD offers specialized solutions for handling all common aspects of aesthetics and engineering during the whole design process of the built environment — buildings, interiors, urban areas, etc. Development of ArchiCAD started in 1982 for the original Apple Macintosh. ArchiCAD is recognized as the first CAD product on a personal computer able to create both 2D drawings and parametric 3D geometry. In its debut in 1987 ArchiCAD also became the first implementation of BIM under Graphisoft's "Virtual Building" concept. Today more than 100,000 architects are using it in the building design industry. Used in Architecture Sector. Other software used in this sector is Chief Architect, Punch Software, AutoCAD Architecture, Vector Works Architecture, etc.

ORACLE (RDBMS):

* The Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is an object-relational database management system produced and marketed by Oracle Corporation.
* Larry Ellison and his friends, former co-workers Bob Miner and Ed Oates, started the consultancy Software Development Laboratories (SDL) in 1977. SDL developed the original version of the Oracle software. The name Oracle comes from the codename of a CIA-funded project Ellison had worked on while previously employed by Ampex. The Oracle RDBMS stores data logically in the form of table spaces and physically in the form of data files. Table spaces can contain various types of memory segments, such as Data Segments, Index Segments, etc. Segments in turn comprise one or more extents. Extents comprise groups of contiguous data blocks. Data blocks from the basic units of data storage. A DBA can impose maximum quotas on storage per user within each table space. Other examples in this sector are SQLBase, SQLite, txtSQL, SmallSQL, etc.

REFERENCE SOFTWARE: Reference software is software which emulates and expands upon print reference forms including the dictionary, translation dictionary, encyclopedia, thesaurus, and atlas. Like print references, reference software can either be general or specific to a domain, and often includes maps and illustrations, as well as bibliography and statistics. Reference software may include multimedia content including animations, audio, and video, which further illustrate a concept. Well-designed reference software improves upon the navigability of print references, through the use of search functionality and hyperlinks. Other examples of this type of software are Classroom Management, Survey Management, Sales readiness software, etc.

ANTIVIRUS: Antivirus or anti-virus software (usually written with the acronym of AV) is software used to prevent, detect and remove malware (of all descriptions), such as: computer viruses, malicious BHOs, hijackers, ransom ware, key loggers, backdoors, rootkits, Trojan horses, worms, malicious LSPs, dialers, fraud tools, adware and spyware. Computer security, including protection from social engineering techniques, is commonly offered in products and services of antivirus software companies.

There are several methods which antivirus software can use to identify malware:

* Signature based detection is the most common method. To identify viruses and other malware, antivirus software compares the contents of a file to a dictionary of virus signatures. Because viruses can embed themselves in existing files, the entire file is searched, not just as a whole, but also in pieces.
* Heuristic-based detection like malicious activity detection, can be used to identify unknown viruses.
* File emulation is another heuristic approach. File emulation involves executing a program in a virtual environment and logging what actions the program performs. Depending on the actions logged, the antivirus software can determine if the program is malicious or not and then carry out the appropriate disinfection actions.
* Ex. COMODO, AVG, Avast, Norton, etc.

GPS: GPS navigation software usually falls into one of the following two categories:

* Navigation with route calculation and directions from the software to the user of the route to take based on a vector-based map, normally for motorized vehicles with some motorized forms added on as an afterthought.
* Navigation tracking, often with a map "picture" in the background, but showing where you have been, and allowing "routes" to be preprogrammed, giving a line you can follow on the screen. This type can also be used for geocaching.
* Other navigation software are [Navigon](http://en.wikipedia.org/wiki/Navigon), [iGO](http://en.wikipedia.org/wiki/IGO_%28software%29), [NDrive](http://en.wikipedia.org/wiki/NDrive), [ROUTE 66](http://en.wikipedia.org/wiki/Route_66_%28company%29), etc.

**SOFTWARE DEVELOPMENT PROCESS**

A software development process, also known as a software development life-cycle (SDLC), is a structure imposed on the development of a software product. Similar terms include software life cycle and software process. It is often considered a subset of systems development life cycle. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

Software Development Activities:

* Planning is an objective of each and every activity, where we want to discover things that belong to the project. An important task in creating a software program is extracting the requirements or requirements analysis.
* Implementation is the part of the process where software engineers actually program the code for the project.
* Software testing is an integral and important phase of the software development process. This part of the process ensures that defects are recognized as soon as possible.
* Documenting the internal design of software for the purpose of future maintenance and enhancement is done throughout development.
* Deployment starts directly after the code is appropriately tested, approved for release, and sold or otherwise distributed into a production environment. This may involve installation, customization (such as by setting parameters to the customer's values), testing, and possibly an extended period of evaluation.
* Software training and support is important, as software is only effective if it is used correctly.
* Maintaining and enhancing software to cope with newly discovered faults or requirements can take substantial time and effort, as missed requirements may force redesign of the software.

Software Development Models: A (software/system) process model is a description of the sequence of activities carried out in a Software Engineering project, and the relative order of their activities.

Several models exist to streamline the development process. Each one has its pros and cons, and it is up to the development team to adopt the most appropriate one for the project. Sometimes a combination of the models may be more suitable.

Waterfall Model:

* The waterfall model is a sequential design process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance.
* The waterfall development model originates in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.

When to use the waterfall model:

* Requirements are very well known, clear and fixed.
* Product definition is stable.
* Technology is understood.
* There are no ambiguous requirements.
* Ample resources with required expertise are available freely.
* The project is short.

Advantages of Waterfall Model:

* Easy to understand and implement.
* Widely used and known (in theory).
* Fits other engineering process models: civil, mechanical etc.
* Reinforces good habits: define-before- design, design before-code.
* Identifies deliverables and milestones.
* Document driven: People leave, documents don’t published documentation standards: URD, SRD, etc., e.g. ESA PSS-05.
* Works well on large/mature products and weak teams.

Disadvantages of Waterfall Model:

* The model implies that you should attempt to complete a given stage before moving on to the next stage
* Does not account for the fact that requirements constantly change.
* It also means that customers cannot use anything until the entire system is complete.
* The model makes no allowances for prototyping.
* It implies that you can get the requirements right by simply writing them down and reviewing them.
* The entire functionality is developed and then tested all together at the end. Major design problems may not be detected till very late.
* The model implies that once the product is finished, everything else is maintenance.

Rapid Prototyping Model: Rapid application development (RAD) is a software development model/methodology that uses minimal planning in favor of rapid prototyping. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements. RAD is not appropriate when technical risks are high. This process is divided into four phases as follows:

* Requirements planning phase – combines elements of the system planning and systems analysis phases of the Systems Development Life Cycle (SDLC). Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.
* User design phase – during this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. User Design is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.
* Construction phase – focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.
* Cutover phase – resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.

When to use RAD Model:

* RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
* It should be used if there’s high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
* RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

Advantages of Rapid Prototyping Model:

* Reduces risk of incorrect user requirements.
* Good where requirements are changing/uncommitted.
* Regular visible progress aids management.
* Supports early product marketing.

Disadvantages of Rapid Prototyping Model:

* An unstable/badly implemented prototype often becomes the final product.
* Requires extensive customer collaboration:
	+ Costs customers time/money.
	+ Needs committed customers.
	+ Difficult to finish if customer withdraws.
	+ May be too customer specific, no broad market.
* Difficult to know how long project will last.
* Easy to fall back into code-and-fix without proper requirements analysis, design, customer evaluation and feedback.

Incremental Model:

* Development occurs as a succession of releases with increasing functionality.
* Customers provide feedback on each release.
* Used in deciding requirements and improvements for next release.
* There is no “maintenance” phase – each version includes both problem fixes as well as new features.
* May also include “re-engineering” – changing the design and implementation of existing functionality, for easier maintainability.
* The “phased release” and “evolutionary” models discussed in the text are incremental development approaches.

When to use the Incremental Model:

* Requirements of the complete system are clearly defined and understood.
* Major requirements must be defined; however, some details can evolve with time.
* There is a need to get a product to the market early.
* A new technology is being used.
* Resources with needed skill set are not available.
* There are some high risk features and goals.

Advantages of Incremental Model:

* Customers get usable functionality earlier than with waterfall.
* Getting early feedback improves likelihood of producing a product that satisfies customers.
* Reduces market risk: if customers hate the product, find out early before investing too much effort and money.
* The quality of the final product is better.
* The core functionality is developed early and tested multiple times (during each release).
* Only a relatively small amount of functionality added in each release: easier to get it right and test it thoroughly.
* Detect design problems early and get a chance to redesign.

Disadvantages of Incremental Model:

* Needs good planning and design.
* Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
* Total cost is higher than waterfall.

eXtreme Programming (XP): Extreme Programming (XP) is a software development model/methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of Agile Software Development, it advocates frequent releases in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted. Other elements of Extreme Programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, simplicity and clarity in code, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers.

 When to use Agile Model:

* When new changes are needed to be implemented. The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
* To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.
* Unlike the waterfall model in agile model very limited planning is required to get started with the project. Agile assumes that the end users’ needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly affected or removed based on feedback. This effectively gives the customer the finished system they want or need.
* Both system developers and stakeholders alike, find they also get more freedom of time and options than if the software was developed in a more rigid sequential way. Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available; meaning the project can continue to move forward without fear of reaching a sudden standstill.

Advantages of Agile Model:

* Customer satisfaction by rapid, continuous delivery of useful software.
* People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
* Working software is delivered frequently (weeks rather than months).
* Face-to-face conversation is the best form of communication.
* Close, daily cooperation between business people and developers.
* Continuous attention to technical excellence and good design.
* Regular adaptation to changing circumstances.
* Even late changes in requirements are welcomed.

Disadvantages of eXtreme Programming:

* In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
* There is lack of emphasis on necessary designing and documentation.
* The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
* Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

Spiral Model: The spiral model is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. Also known as the spiral lifecycle model (or spiral development), it is a systems development method (SDM) used in information technology (IT). This model of development combines the features of the prototyping and the waterfall model. The spiral model is intended for large, expensive and complicated projects. The spiral model is based on continuous refinement of key products for requirements definition and analysis, system and software design, and implementation (the code). At each iteration around the cycle, the products are extensions of an earlier product. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.

Starting at the center, each turn around the spiral goes through several task regions:

* Determine the objectives, alternatives, and constraints on the new iteration.
* Evaluate alternatives and identify and resolve risk issues.
* Develop and verify the product for this iteration.
* Plan the next iteration.

When to use Spiral Model:

* When costs and risk evaluation is important.
* For medium to high-risk projects.
* Long-term project commitment unwise because of potential changes to economic priorities.
* Users are unsure of their needs.
* Requirements are complex.
* New product line.
* Significant changes are expected (research and exploration).

**Advantages of Spiral Model:**

* High amount of risk analysis hence, avoidance of Risk is enhanced.
* Good for large and mission-critical projects.
* Strong approval and documentation control.
* Additional Functionality can be added at a later date.
* Software is produced early in the [software life cycle](http://istqbexamcertification.com/what-are-the-software-development-life-cycle-phases/).

**Disadvantages of Spiral model:**

* Can be a costly model to use.
* Risk analysis requires highly specific expertise.
* Project’s success is highly dependent on the risk analysis phase.
* Doesn’t work well for smaller projects.

Rational Unified Process: The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM since 2003. RUP is not a single concrete prescriptive process, but rather an adaptable process framework, intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the Unified Process. RUP is based on a set of building blocks and content elements, describing what is to be produced, the necessary skills required and the step-by-step explanation describing how specific development goals are to be achieved. The main building blocks, or content elements, are the following:

* Roles (who) – A Role defines a set of related skills, competencies and responsibilities.
* Work Products (what) – A Work Product represents something resulting from a task, including all the documents and models produced while working through the process.
* Tasks (how) – A Task describes a unit of work assigned to a Role that provides a meaningful result.

Advantages of RUP Software Development:

* This is a complete methodology in itself with an emphasis on accurate documentation.
* It is proactively able to resolve the project risks associated with the client's evolving requirements requiring careful [change request management](http://www.my-project-management-expert.com/change-request-management.html)
* Less time is required for integration as the process of integration goes on throughout the [software development life cycle](http://www.my-project-management-expert.com/software-development-life-cycle-model.html).
* The development time required is less due to reuse of components.
* There is online training and tutorial available for this process.

Disadvantages of RUP Software Development:

* The team members need to be expert in their field to develop a software under this methodology.
* The development process is too complex and disorganized.
* On cutting edge projects which utilize new technology, the reuse of components will not be possible. Hence the time saving one could have made will be impossible to fulfill.
* Integration throughout the [process of software development](http://www.my-project-management-expert.com/process-of-software-development.html), in theory sounds a good thing. But on particularly big projects with multiple development streams it will only add to the confusion and cause more issues during the stages of testing.

Choosing a Process Model:

* Waterfall approach is OK for small projects, and when requirements & design can be done near-perfectly upfront.
* Most complex real-world projects these days use an incremental approach with multiple phases.
* There are other process models that bring in other ideas.
* The “spiral” model is risk-driven. Risks are analyzed after each iteration to figure out whether to continue and what to do in the next iteration to reduce risks.
* “Concurrent engineering” brings in the idea of doing different project activities concurrently.
* We can borrow ideas from different process models and create an approach that is suited to the characteristics of our particular project.

**IMPORTANCE OF SOFTWARE ENGINEERING**

Software engineering is the discipline of designing, writing, testing, implementing and maintaining software. It forms the basis of operational design and development of virtually all computer systems. The discipline extends to application software on personal computers, connectivity between computers, operating systems and includes software for micro-controllers, small computers embedded in all types of electronic equipment.

Without software engineering, computers would have no functionality. Although hardware is just as important, no software means no computers. It is a fundamental part of today's information systems and engineering and our lives would be very different without it.

**AS A PROFESSION**

Software engineering (SE) is the profession, practiced by software engineers, concerned with specifying, designing, developing and maintaining software applications by applying technologies and practices from computer science (CS), project management, and other fields, which improves the productivity of developers and the quality of the application they develop.

Most software engineers work as employees or contractors. Software engineers work with businesses, government agencies (civilian or military), and non-profit organizations. Some software engineers work on their own as Consulting Software Engineers. Some organizations have specialists to perform each of the tasks in the software development process. Other organizations required software engineers to do many or all of them. Entry-Level Software Engineer or Associate Software Engineer may be best. Some companies offer Software Engineer as an entry level position. In large projects, people may specialize in only one role. In small projects, people may fill several or all roles at the same time. Specializations include: in industry (analysts, architects, developers, testers, technical support, managers) and in academia (educators, researchers).

There is considerable debate over the future employment prospects for Software Engineers and other IT Professionals. For example, an online futures market called the Future of IT Jobs in America attempts to answer whether there will be more IT jobs, including software engineers, in 2012 than there were in 2002. Possible opportunities for Advancement can be as a Software Engineer, then to a Senior Software Engineer, or straight to a Senior Software Engineer, depending on skills and reputation.

There has been a healthy growth in the number of India's IT professionals over the past few years. From a base of 6,800 knowledge workers in 1985-86, the number increased to 522,000 software and services professionals by the end of 2001-02. It is estimated that out of these 522,000 knowledge workers, almost 170,000 are working in the IT software and services export industry; nearly 106,000 are working in the IT enabled services and over 220,000 in user organizations.

 **AS A FIELD IN EDUCATION**

About half of all practitioners today have [degrees](http://en.wikipedia.org/wiki/Academic_degree) in [computer science](http://en.wikipedia.org/wiki/Computer_science), [information systems](http://en.wikipedia.org/wiki/Information_systems), or [information technology](http://en.wikipedia.org/wiki/Information_technology). A small, but growing, number of practitioners have software engineering degrees. In 1987, [Imperial College London](http://en.wikipedia.org/wiki/Imperial_College_London) introduced the first three-year software engineering [Bachelor's degree](http://en.wikipedia.org/wiki/Bachelor%27s_degree) in the UK and the world; in the following year, the [University of Sheffield](http://en.wikipedia.org/wiki/University_of_Sheffield) established a similar program. In 1996, the [Rochester Institute of Technology](http://en.wikipedia.org/wiki/Rochester_Institute_of_Technology) established the first software engineering Bachelor's degree program in the United States, however, it did not obtain [ABET](http://en.wikipedia.org/wiki/ABET) accreditation until 2003, the same time as [Rice University](http://en.wikipedia.org/wiki/Rice_University), [Clarkson University](http://en.wikipedia.org/wiki/Clarkson_University), [Milwaukee School of Engineering](http://en.wikipedia.org/wiki/Milwaukee_School_of_Engineering) and [Mississippi State University](http://en.wikipedia.org/wiki/Mississippi_State_University) obtained theirs. In 1997, PSG College of Technology in Coimbatore, India was the first to start a five-year integrated Master of Science degree in Software Engineering.

Since then, software engineering undergraduate degrees have been established at many universities. A standard international curriculum for undergraduate software engineering degrees was recentlydefined by the [CCSE](http://en.wikipedia.org/wiki/CCSE). As of 2004, in the U.S., about 50 universities offer software engineering degrees, which teach both computer science and engineering principles and practices. The first software engineering [Master's degree](http://en.wikipedia.org/wiki/Master%27s_degree) was established at [Seattle University](http://en.wikipedia.org/wiki/Seattle_University) in 1979. Since then graduate software engineering degrees have been made available from many more universities. Likewise in Canada, the Canadian Engineering Accreditation Board (CEAB) of the [Canadian Council of Professional Engineers](http://en.wikipedia.org/wiki/Canadian_Council_of_Professional_Engineers) has recognized several software engineering programs.

In 1998, the US [Naval Postgraduate School](http://en.wikipedia.org/wiki/Naval_Postgraduate_School) (NPS) established the first [doctorate](http://en.wikipedia.org/wiki/Doctorate) program in Software Engineering in the world. Additionally, many online advanced degrees in Software Engineering have appeared such as the Master of Science in Software Engineering (MSE) degree offered through the Computer Science and Engineering Department at [California State University, Fullerton](http://en.wikipedia.org/wiki/California_State_University%2C_Fullerton). Steve McConnell opines that because most universities teach computer science rather than software engineering, there is a shortage of true software engineers.  ETS University and UQAM were mandated by IEEE to develop the Software Engineering Body of Knowledge ([SWEBOK](http://en.wikipedia.org/wiki/SWEBOK)), which has become an ISO standard describing the body of knowledge covered by a software engineer.

 **CONCLUSION**

This paper deals with Software Engineering. What Software is, what Engineering is and what the world understands by Software Engineering. Why was it needed or what lead to its development. How the world has shared the knowledge of soft wares and benefited from it. While numerous approaches, including methods and frameworks have been introduced and adopted as industry standards and/or best practices that have mitigated many of these issues. Software used in the various spheres/fields in the world like ArchiCAD, Oracle, GPS, Antivirus, etc. which have made work simpler for those in their respective fields. Software development process, also known as a software development life-cycle (SDLC), is a structure imposed on the development of a software product has also been dealt with in the paper. Followed by the importance of Software Engineering or the value carried by it in today’s world in our lives be it as a profession or as a field of education.

**GLOSSARY**

1. FORTRAN: developed by [IBM](https://en.wikipedia.org/wiki/IBM) at their campus in south San Jose, California in the 1950s for scientific and engineering applications.
2. ALGOL (short for ALGOrithmic Language) is a family of [imperative](https://en.wikipedia.org/wiki/Imperative_programming) computer [programming languages](https://en.wikipedia.org/wiki/Programming_language) originally developed in the mid-1950s which greatly influenced many other languages.
3. COBOL is one of the oldest programming languages, primarily designed by Grace Hopper. Its name is an acronym for COmmon Business-Oriented Language, defining its primary domain in business, finance, and administrative systems for companies and governments.
4. Edsger Wybe Dijkstra; 11 May 1930 – 6 August 2002) was a Dutch computer scientist. He received the 1972 Turing Award for fundamental contributions to developing programming languages, and was the Schlumberger Centennial Chair of Computer Sciences at The University of Texas at Austin from 1984 until 2000.
5. David Lorge Parnas (born February 10, 1941) is a Canadian early pioneer of software engineering, who developed the concept of information hiding in modular programming, which is an important element of object-oriented programming today. He is also noted for his advocacy of precise documentation.
6. Simula is a name for two simulation programming languages, Simula I and Simula 67, developed in the 1960s at the Norwegian Computing Center in Oslo, by Ole-Johan Dahl and Kristen Nygaard. Syntactically, it is a fairly faithful superset of ALGOL 60.
7. A microcomputer is a small, relatively inexpensive computer with a microprocessor as its central processing unit (CPU).[2] It includes a microprocessor, memory, and input/output (I/O) facilities.
8. A software development process, also known as a software development life-cycle (SDLC), is a structure imposed on the development of a software product.
9. Smalltalk is an object-oriented, dynamically typed, reflective programming language.
10. Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. It is the main programming language used by Apple for the OS X and iOS operating systems and their respective APIs, Cocoa and Cocoa Touch.
11. C++ (pronounced "see plus plus") is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises both high-level and low-level language features.[3] Developed by Bjarne Stroustrup starting in 1979 at Bell Labs, C++ was originally named C with Classes, adding object oriented features, such as classes, and other enhancements to the C programming language.
12. Open-source software (OSS) is computer software with its source code made available and licensed with a license in which the copyright holder provides the rights to study, change and distribute the software at no cost to anyone and for any purpose.
13. The World Wide Web (abbreviated as WWW or W3 commonly known as the web), is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia, and navigate between them via hyperlinks.
14. The Internet is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to serve several billion users worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless and optical networking technologies.
15. Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another.
16. A virtual machine (VM) is a software based, fictive computer. Virtual machines may be based on specifications of a hypothetical computer or emulate the computer architecture and functions of a real world computer.
17. February 2001, 17 software developers[6] met at the Snowbird, Utah and published the Manifesto for Agile Software Development[1] to define the approach now known as agile software development.
18. HyperText Markup Language (HTML) is the main markup language for creating web pages and other information that can be displayed in a web browser.
19. Search engine optimization (SEO) is the process of affecting the visibility of a website or a web page in a search engine's "natural" or un-paid ("organic") search results.
20. Hobbyist: A person who pursues a particular hobby: "a computer hobbyist".
21. Article Source: http://EzineArticles.com/?expert=Edeh\_Chijioke.