**SEMCOM**

**Faculty Name: Dr. Jay Nanavati**
**Sem: 5 (TYBCA)**
**Subject: Software Engineering**
**Unit-1**

## Software (IEEE Definition)

- Software is the collection of programs, procedures, rules and documentation and data.
- It is used by many people instead of developer.
- It requires sufficient documentation as help to user.
- Programs are tested perfectly because users are not allowed to make changes.

### Difference between Program & software

| Program | Software |
|---|---|
| Small in size (LOC) | Large in size (KLOC) |
| Limited Functionality | Wide range of functionality |
| Simple or less complex | Complex |
| Used by the author of the program | Used by people other than developers |
| Little or no documentation | Sufficient documentation |
| Bugs can be tolerated | Bugs can not be tolerated |
| Testing is often not done. | Testing is always done. |
| Portability, reliability & usability are not so important. | Portability, reliability & usability are also important. |

## Software Engineering

### History

- The electronic computers started during 1940s.During this time, there was more concentration on the development of hardware. There were no operating systems and the programs were written on paper tapes or switches.
- During 1950, the concept of operating systems started and languages like FORTRAN and COBOL were introduced. Thus now the hardware was not much important, but the programs were important.
- In 1960s multiprogramming OS were found. Thus, prices were of hardware decreased.
- With the availability of cheaper and more powerful machines, higher-level languages, and more user-friendly operating systems, the applications of computers grew rapidly.
- The techniques for writing simple programs could not be scaled up for developing software systems. Now there was need of some standard techniques to write programs. This was called Software crisis.
- There were two meetings held by NATO (North Atlantic Treaty Organization) Science committee and the term “Software Engineering” was coined.
- With time, the cost of hardware decreased, and for software increased. This is because the development of software has many people involved and time taken is also more.

**Definition**

- **Software Engineering** is systematic approach to development, operation, maintenance and retirement of software.

- Goal of Software Engineering: To produce high-quality software at low cost.

**Software Process & its characteristics**

- **Software Process:** A software process specifies a method of developing software.
- **Software Project:** A software project is a development project in which a software process is used.
- **Software Product:** Software product is the outcome of a software project.

**Characteristics of Software Process:**

1. Predictability:
   It determines how accurately the outcome of a process in a project can be predicted before the project is completed. Past experience can ONLY be used if a process is predictable. A process is predictable only if it is under statistical control.
2. Testability & Maintainability:
   The process should produce software that is easy to test and maintain. The goal of the process should not be to reduce the effort of design and coding, but to reduce the cost of testing and maintenance. Testing and Maintenance depends on design and coding hence software must be designed and coded to make testing and maintenance easier.
3. Early Defect Removal and Defect Prevention:
   Errors can occur at any stage during development. The cost of correcting errors of different phases is not same and it depends on when it was detected and corrected. The greater the delay in detecting and correcting errors after it occurs the more it is expensive. Thus we should attempt to detect errors that occurs in a phase during the phase itself and should not wait until testing to detect errors.
4. Process improvement:
   In context of software as the productivity and quality are determined largely by the process. For quality improvement and cost reduction the software process must be improved.

**Phases in software development**

A software development process consists of various phases, each phase ending with a defined output.
The main reason for having phases is that it ensures that the cost of development is lower compared to if it was developed as a whole.
It allows proper checking for quality and progress at some defined points. Otherwise one will have to wait until entire software is developed completely. Developing software as a whole results in complexity.

Thus for managing the complexity, project tracking and quality, software development consists of following four phases.

1. Requirement Analysis
2. Software Design

3. Coding
4. Testing

**Requirement analysis:**
During this phase we try to understand the problem, which will be solved by the software. The problem may be to develop software from existing system, develop a new system or combination of two.
Goal: To produce Software Requirement Specification (SRS) document.
The person responsible for the requirements analysis is called the Analyst.
There are two parties involved in software development: Client and Developer. Now, both of them may not understand each other's problem. So this step will act as bridge of communication.

There are two parts of analysis:
1. Problem Identification
2. Requirement Specification.

In Problem Identification, each and every part of the system is first studied, and inputs and outputs are decided. If new system is built from existing system, we can also think of new features we can provide in automated system.
In Requirement specification, the developers will specify all the requirements in form of document. For that they can use: English, Regular Expression, tables or its combination. The requirements document must specify the functional and performance requirements, the formats of inputs and outputs and all the design constraints that exist due to political, economic, and environmental and security reasons.
This phase ends with Validation of SRS document. From Validation, we can make sure that all the requirements are given. This can be done with Requirement review technique, in which group of people of developers and also of clients will make the review.

**Software Design:**
In this phase we decide how to satisfy the requirements given in first step.
Goal: To plan solution for the problem given in requirement analysis.
There are two parts in this phase:
1. System Design
2. Detailed Design

System Design is also called Top-Level Design. It specifies all the modules used in system, and the relations between them. In this part we decide Data Structures, Files, Output Formats and main modules.

Detailed Design specifies internal logic of each module. This logic is given in high-level language, and not in a language in which software is developed. The output contains design document, which has separate two documents-for system design and for detailed design.
A design methodology is a systematic approach to creating a design by application of set of techniques and guidelines.

**Coding:**
This phase is related to the programming used for the system.
Goal: To translate the design in the code of required programming language in software.
This phase affect the testing and maintenance. If the code is well written, it can reduce the cost of testing and maintenance.
The standard used for coding is Structured programming. It means that programs should be written in sequence of statements and statements should be executed in sequence. Such statements are like single-entry-single-exit statements.

To verify this step, we may read the code by review technique. The testing of coding by executing it is called 'unit testing'. Here we run each module of system rather than running the whole system. So this phase is called "Coding and Unit Testing".
The output of this phase is unit-tested code of different modules.

**Testing:**
Testing is the measure to check the quality of software during the development.
Goal: To detect the errors in the requirement, design and coding phases.
This phase starts with test plan, which identifies all the activities that must be performed and specifies the schedule, allocates the resources and specifies guidelines for testing. It specifies conditions to be tested, different units to be tested and the manner in which the modules will be integrated. During testing a test case specification document is produced, which lists different test cases, together with expected outputs. The specified test cases are executed and actual result is compared with expected output.

Relations between modules are considered and are combined to perform Integration Testing. System is performed to check that system is working according to the requirement. Finally acceptance testing is done in which client will enter the actual data and check the working of software.
The final output of testing phase is Test Report and the Error report. Test report contains set of test cases along with actual output. Error report describes the error encountered.

**Maintenance:**
Maintenance is not part of software development but it is an important activity in life of software.
Major type of maintenance:
Adaptive Maintenance:
In this type of maintenance there is no error in the system but there may be some changes required by the Client in the input and output formats, or in the requirement analysis.

Corrective Maintenance:
In this case there are errors in software and they are solved.

**Effort Distribution:**
Software Development may take few months to few years. It can be used for about 5-20 years. The cost of software development is the total cost of its phases-Requirement analysis, Design, Coding and testing. The cost of maintenance after software is developed is more than development cost.
The ratio of development to cost is given as 40/60, 30/70 or even lower. The main reason for this is that developers want to reduce the cost of development and they are not concerned with the maintenance part, that part is done by the users.
A typical distribution effort is given as below:

| | |
|---|---|
| Requirements | 10% |
| Design | 20% |
| Coding | 20% |
| Testing | 50% |

These values can be different for the organizations and different projects.
The conclusion from this distribution:

- The Coding part contains only small percentage, because software development is not just writing programs.

- The testing part contains major part, because it is very important and quality of software depends on it.
- The goal of cost and design should not be to reduce its own cost, but to reduce the cost of Testing and Maintenance.

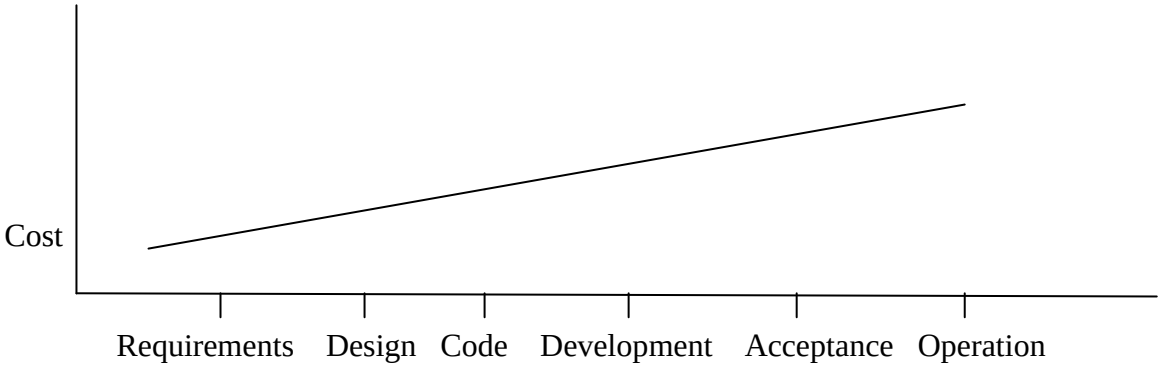
**Error Distribution:**

The programming part is very important because there is possibility of errors. These errors can occur at any stage in the software development.

A typical error distribution is given as:

| | |
|---|---|
| Requirement analysis | 20% |
| Design | 30% |
| Coding | 50% |

The cost of correction of errors with different phases is as shown below:




The greater the delay in detecting an error after it occurs, the more expensive it is to correct it. As shown in figure an error that occurs during the requirements phase, if corrected during acceptance testing, can cost up to 100 times more than correcting the error during requirements phase itself. The reason is that if there is an error in the requirements then the design and code will be affected by it. To correct the error after coding is done would require both the design and code to be changed, thereby increasing the cost of correction.

Thus we should attempt to detect errors that occur in a phase itself and should not wait until testing to detect errors.

**SOFTWARE DEVELOPMENT PROCESS MODELS:**

**Software Development Process** focuses on the activities related to production of software. for example, design, coding and testing.

**Process Model:**

Process Model specifies some activities that should be performed and the order in which they should be performed.

**Waterfall Model:** Waterfall Model is the simplest process model. The phases in this model are arranged in linear order.

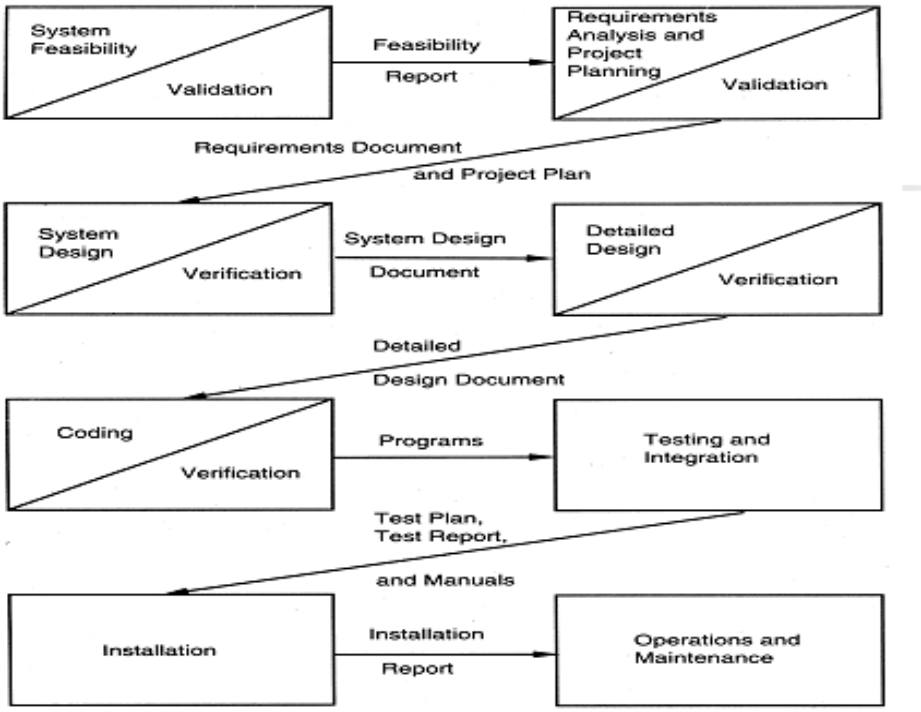
In this model, a project begins with feasibility analysis. If the project is found feasible then further activities are carried out.

The sequence of activities in waterfall model is:

Requirement Analysis, Project Planning, System Design, Detailed Design, Coding, Unit Testing, System Integration and Testing followed by Installation, Operations and Maintenance.

Here it is necessary to identify the end of a phase and beginning of next. This can be done by certification mechanism, which is usually done by verification or validation at the end of each phase. This will ensure that output of a phase is consistent with its input and with overall requirements.

The waterfall model is using linear order because of the following two assumptions:
For successful project resulting in successful product, all the phases listed in waterfall model must be performed anyway. Any different order of phases will result in less successful software product.
A successful software product is one that satisfies all the objectives of the development project. These objectives include satisfying requirements and performing development within time and cost constraints.




Project outputs in waterfall model:

- Requirements documents
- Project plan
- System Design Document
- Detailed Design Document
- Test Plan and Test Reports
- Final Code
- Software Manuals
- Review Reports

To certify an output product of a phase before next phase begins, reviews are often held. Reviews are formal meetings to uncover deficiencies in a product. The review reports are the outcomes of these reviews.

Limitations of Waterfall Model:

The waterfall model assumes that requirements can be frozen i.e. fixed.

This is possible for systems designed to automate an existing manual system. But for new systems, determining requirements is difficult, as the user does not even know the requirements.

Freezing requirements usually requires choosing the hardware. A large software project may take few years to complete. It may happen that by time hardware technology may become obsolete because hardware technology is changing fast.

This model suggests that the requirement must be completely specified before the rest of development can proceed. Sometimes it is desirable to first develop a part of system completely and then later enhance the systems in phases. This can be done in general marketing systems in which the requirements are likely determined by the developers.

It is document driven process that requires formal documents at the end of each phase. This approach tends to make the process documentation-heavy and is not suitable for many applications, particularly interactive applications, where developing elaborate documentation of user interfaces is not feasible. Also if development is done using fourth generation languages or modern development tools, developing elaborate specifications before implementation is sometimes unnecessary.

It is well suited for routine types of projects, where requirements are well understood. That is waterfall model is suitable if we are familiar with the problem domain and requirements are clear.

**Prototyping model:**

The goal of this model is to overcome the first two limitations of the waterfall model. In this model requirements are not freezed, but a prototype is created as per the current requirements. Development of prototype obviously undergoes design, coding and testing, but each of these phases not done very formally and thoroughly. By this model the client gets the actual feel of the system, because this enables the client to better understand the requirements.

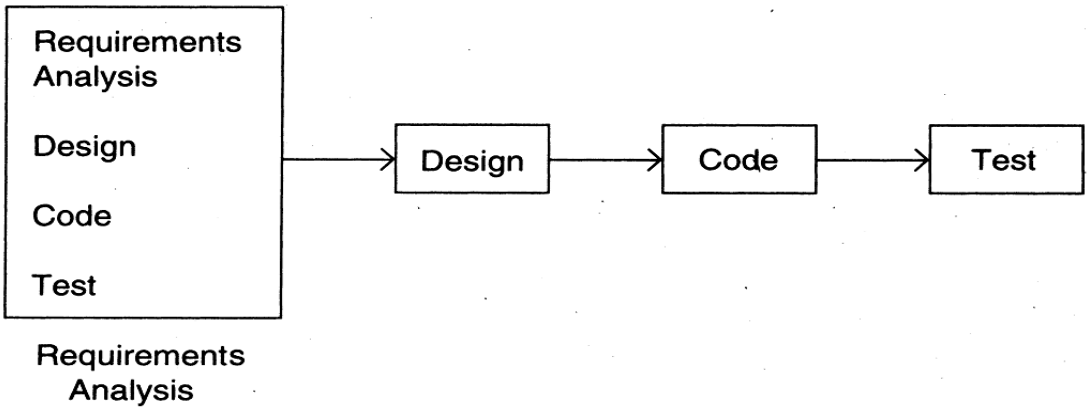
This approach is good for complicated and large systems for which there is no existing system or manual process to help determine the requirements. It helps to know the feasibility of certain approach.

The development of prototype starts when preliminary version of the requirements specification has been developed. After the prototype has been developed, the end users and clients use the prototype and based on their experience, they provide feedback to the developers. Based on the feedback the prototype is modified and the clients are again allowed to use the system. The cycle repeats until the users and clients are completely satisfied. The initial requirements specification is modified to final requirement specification.

In Prototype only those features which are not properly understood are included.

Exception Handling, Recovery and formats are not included in prototype.

This model is useful for the large and complicated system, which is totally new, and there is no existing system for it. The concept of the model is shown as below:




Prototype model is generally not used for following reasons:
The cost of development is more, because it is similar to building software twice.

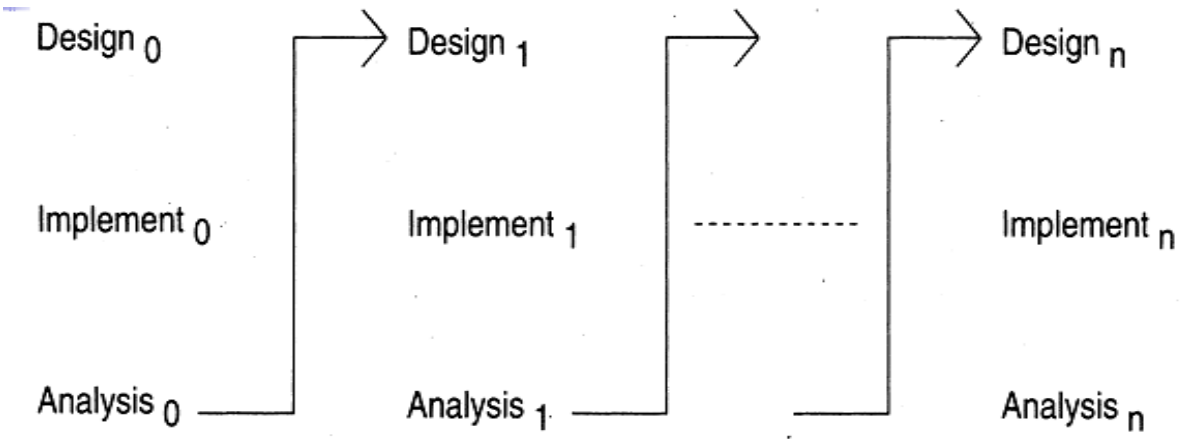
Advantages:
However, in some situations, the cost of software development without prototyping may be more than with prototyping.
Even if it is built twice, the overall cost of development is less, as this model is precise. The cost of testing and documentation is reduced.
The experience of the development is useful to the developers, and thus developing the similar projects of experience will have the less cost.
Prototype model is well suited for projects where requirements are hard to determine and the confidence in obtained requirements is low. In such projects requirements may change and rework is required. Requirements frozen after experience with the prototype model are more stable.

**Iterative Enhancement Model:**
The goal of Iterative Enhancement Model is to remove the third limitation of the waterfall model and combining the benefits of both prototyping model and waterfall model.
According to this model, the software is developed in terms of the increments, and each increment adds some functionality in the system. This process is continued until the full system is implemented.

Advantages:

This model gives better testing, because there is testing of each phase like waterfall model. The requirements of system are provided as feed back like prototyping.

This model is represented as below:




In the first step, the simple initial implementation is done for overall system. Project Control list is prepared, which gives the list of all tasks that are performed. The phases in this system as shown above are Design, Implementation and analysis. The process continued until the project control list becomes empty. The advantage of using control list is that it minimizes the chance of error, and reduces the redesigning.

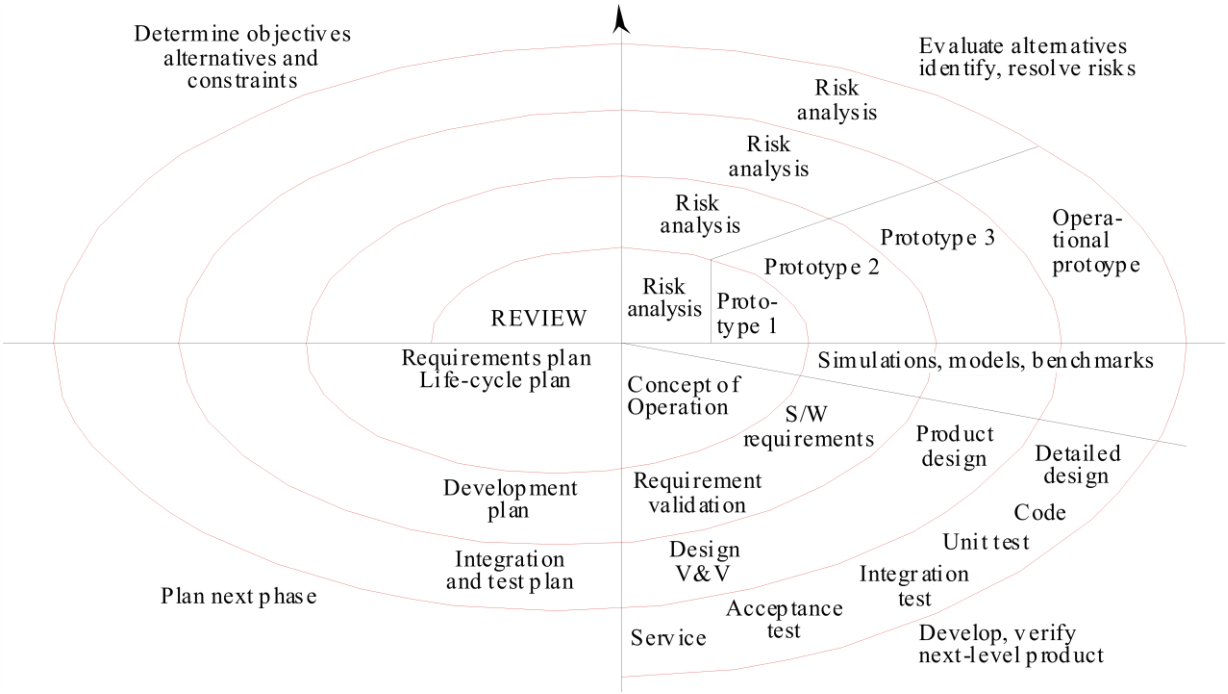
This type model is applicable for product development, in which the developers themselves provide specifications and therefore have a lot of control on what specifications go in the system and what stay out. First a version is released that contains some capability. Based on feedback from users and experience from users and experience from this version a list of additional desirable features and capabilities are generated. These features forms the basis of the software, and are included in next version.
It is also applicable in client-oriented projects, here the client's organization does not have to pay for the entire software together, it can get main part of the software developed and perform cost-benefit analysis for it before enhancing the software with more capabilities.
This process model can be useful if the core of the application to be developed is well understood and the increments can be easily defined.

**Spiral Model:**
This is recent model proposed by Boehm. It combines features of Prototype model and Waterfall model. It is used for large, complicated and expensive projects. Hence there can be many risks associated with such projects like cost overruns, less satisfactory software developed at the end.
The activities in this model are organized like spiral that has many cycles. The radial dimension represents cumulative cost incurred so far. The angular dimension represents progress made in completing each cycles.

Each cycle begins with
1. Identification of objectives
2. Identification of different alternatives.
3. Identification of constraints

This is the first quadrant of the cycle. In second quadrant we evaluate different alternatives to find risk associated with it. Risks are the chances that some of the objectives may not be met. The next step is to develop strategies that resolve the uncertainties and risks. This step may involve benchmarking, simulation and prototyping.

Next the software is developed keeping in mind the risks. Finally the next phase is planned. Here each of the cycle of spiral model is completed by review of that covers all the products developed during that cycle, including plans for next cycle.
In a typical model, application of the spiral model, it starts with an extra round zero, in which the feasibility of the basic project objectives is studied. In round one, a concept is developed. Here the objectives are stated more precisely and the cost and other constraints are defined precisely. In round two, top-level requirements are developed. In succeeding rounds actual development may be done.
This model is <u>useful for high-risk projects.</u>




**<u>This figure is for understanding only, the students are not supposed to draw it in exam.</u>**

**Role of Metrics:**
Software metrics is the measure to quantify the characteristics of software or its development process. The software does not have any physical characteristics, so the metrics used in engineering or other fields cannot be used for software.
There are special metrics used for software, which can quantify the things like size, complexity and reliability.
Metrics and measurement are necessary aspects of managing a software development project.
For effective monitoring, the management needs to get information regarding the project: how far it has progressed, how much development has taken place, how far behind the schedule it is, the quality of development so far, etc. Based on this information decisions can be made about the project. Without proper metrics to quantify the required information, subjective opinion would have to be used which is often unreliable and often goes against the fundamental goals of engineering. Hence metrics based management is key component in the software engineering.

There are two types of metrics:

***Product Metrics:***

It is used to quantify the characteristics of software.

e. g. size, complexity, reliability, efficiency, usability, correctness etc.

***Process Metrics:***

It is used to quantify the characteristics of the process, which is used to develop software.

e.g. productivity, cost and resource requirements, effectiveness of quality assurance measures and the effect of development techniques.