

PAPER NO. CT 52
SECTION 5

CERTIFIED
INFORMATION COMMUNICATION
TECHNOLOGISTS
(CICT)

SOFTWARE ENGINEERING

STUDY TEXT

KASNEB SYLLABUS

wLEARNING OUTCOMES

A candidate who passes this paper should be able to:

- Identify appropriate software system design tools
- Design appropriate software systems
- Describe software system testing
- Document and commission software
- Evaluate software acquisition techniques
- Maintain software

CONTENT

	page
1. Introduction to software systems development.....	4
- Software systems development concepts	
- Software development life cycle	
2. Software process models.....	10
- Linear/waterfall model	
- Rapid prototyping	
- Evolutionary models	
- Component based models	
- Other models	
3. Software requirements analysis.....	19
- Overview or requirements concepts	
- Requirement analysis process	
- Requirements specification	
4. Design tools and methods.....	25
- System flowcharts	
- Case tools	
- Functional decomposition	
- Modules design	
- Structured walkthrough	
- Decision tables	
- Structured charts	
- Data flow diagrams	
- Object Oriented design tools	
5. Software quality.....	63
- Quality control and assurance	
- Software quality factors and metrics	
- Formal technical reviews	
- Verification and validation	
- Cost of quality	
6. Software coding.....	82
- Coding styles and characteristics	
- Coding in high-level languages	
- Coding standards	
- User interface	

7. Software testing.....	94
- Software testing life cycle	
- Software testing methods (Black box testing and White box testing)	
- Software testing levels(unitintegration, system and acceptance testing)	
- Other forms of testing	
8. Software acquisition methods.....	102
- Software costing	
- Software outsourcing	
- Open-source software engineering and customization	
- In-house development	
- Commercial Off The Shelf software (COTS)	
- Budgeting for information systems	
- Financial cost benefit analysis	
- Business case approach	
- Total cost of ownership	
- Balanced scorecard/activity based costing and expected value	
- Tracking and allocating costs	
9. Conversion strategies.....	131
- Conversion planning	
- Parallel running	
- Direct cut over	
- Pilot study	
- Phased approach	
10. Documentation and commissioning.....	138
- Objectives of systems documentation	
- Use of systems documentation	
- Qualities of a good documentation	
- Types of documentation	
- Software commissioning	
11. Software maintenance and evolution.....	146
- Types or software changes	
- Software change identification	
- Software change implementation	
12. Auditing information systems.....	162
- Overview of information systems audit	
- Auditing computer resources	
- Audit techniques	
- Audit applications	
13 Emerging Issues and trends.....	

TOPIC 1

INTRODUCTION TO SOFTWARE SYSTEMS DEVELOPMENT

- *Software systems development concepts*

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.

Fritz Bauer, a German computer scientist, defines software engineering as:

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.

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as **software evolution**. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Evolution Laws

Lehman has given laws for software evolution. He divided the software into three different categories:

- **S-type (static-type)** - This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.
- **P-type (practical-type)** - This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.
- **E-type (embedded-type)** - This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

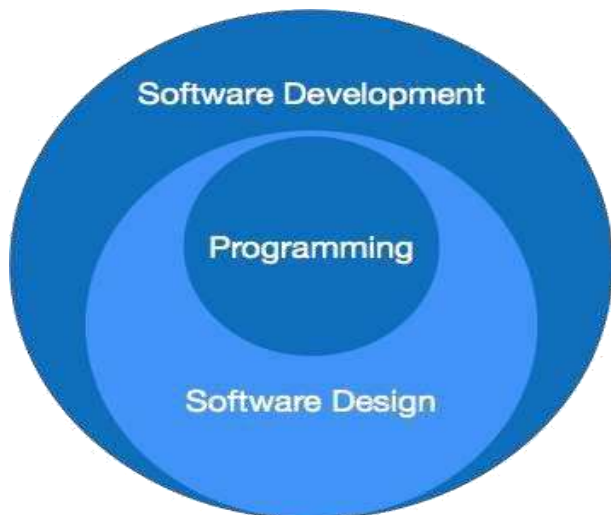
E-Type software evolution

Lehman has given eight laws for E-Type software evolution -

- **Continuing change** - An E-type software system must continue to adapt to the real world changes; else it becomes progressively less useful.
- **Increasing complexity** - As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.
- **Conservation of familiarity** - The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system.
- **Continuing growth**- In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.
- **Reducing quality** - An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment.
- **Feedback systems**- The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation** - E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability** - The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:



Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes :

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development.

This includes:

- Coding
- Testing
- Integration

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software** - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR

Email: info@masomomsingi.co.ke

www.masomomsingi.co.ke

TOPIC 2

SOFTWARE PROCESS MODELS

A software process is a coherent set of activities for specifying, designing, implementing and testing software systems. A structured set of activities required to develop a software system:

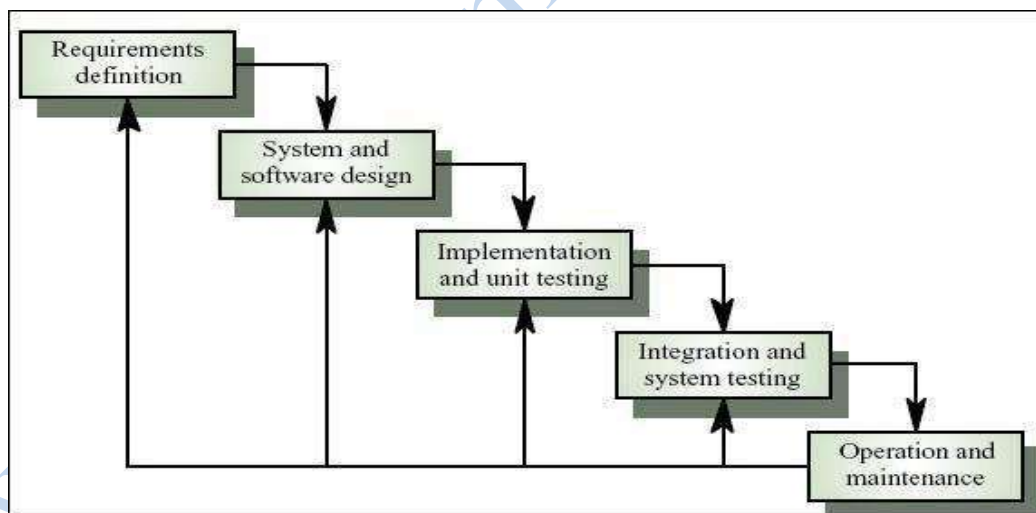
- Specification
- Design
- Validation
- Evolution

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

Generic software process models

- a) The waterfall model - Separate and distinct phases of specification and development.
- b) Evolutionary development - Specification and development are interleaved.
- c) Formal systems development - A mathematical system model is formally transformed to an implementation.
- d) Reuse-based development - The system is assembled from existing components

• *Linear/waterfall model*



This model assumes that everything is carried out and taken place perfectly as planned in the previous stage and there is no need to think about the past issues that may arise in the next phase. This model does not work smoothly if there are some issues left at the

previous step. The sequential nature of model does not allow us go back and undo or redo our actions.

This model is best suited when developers already have designed and developed similar software in the past and is aware of all its domains. The drawbacks of the waterfall model are:

- The difficulty of accommodating change after the process is underway.
- Inflexible partitioning of the project into distinct stages.
- This makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood.
- ***Rapid prototyping***

Rapid Prototyping (RP) can be defined as a group of techniques used to quickly fabricate a scale model of a part or assembly using three-dimensional computer aided design (CAD) data. What is commonly considered to be the first RP technique, Stereolithography, was developed by 3D Systems of Valencia, CA, USA. The company was founded in 1986, and since then, a number of different RP techniques have become available.

Rapid Prototyping has also been referred to as solid free-form manufacturing; computer automated manufacturing, and layered manufacturing. RP has obvious use as a vehicle for visualization. In addition, RP models can be used for testing, such as when an airfoil shape is put into a wind tunnel. RP models can be used to create male models for tooling, such as silicone rubber molds and investment casts. In some cases, the RP part can be the final part, but typically the RP material is not strong or accurate enough. When the RP material is suitable, highly convoluted shapes (including parts nested within parts) can be produced because of the nature of RP.

The reasons of Rapid Prototyping are

- To increase effective communication.
- To decrease development time.
- To decrease costly mistakes.
- To minimize sustaining engineering changes.
- To extend product lifetime by adding necessary features and

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomdingi.co.ke

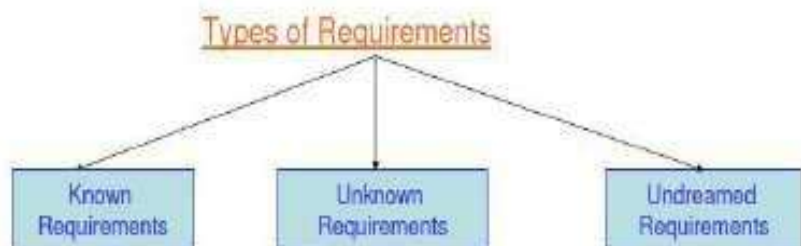
TOPIC 3

SOFTWARE REQUIREMENTS ANALYSIS

- *Overview or requirements concepts*

Software Requirement

1. A condition of capability needed by a user to solve a problem or achieve an objective
2. A condition or a capability that must be met or possessed by a system to satisfy a contract, standard, specification, or other formally imposed document."



- 1 **Known Requirements:** -Something a stakeholder believes to be implemented
- 2 **Unknown requirements:**-Forgotten by the stakeholder because they are not needed right now or needed only by another stakeholder
- 3 **Undreamt requirements:**- stakeholder may not be able to think of new requirement due to limited knowledge

A Known, Unknown and Undreamt requirements may be functional or non-functional.

- **Functional requirements:** - describe what the software has to do. They are often called product features. It depends on the type of software, expected users and the type of system where the software is used.
- **Non Functional requirements:** - are mostly quality requirements. That stipulate how well the software does, what it has to do. These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- **User requirements:** -Statements in natural language plus diagrams of the services the system provides and its operational constraints.
- **System requirements:** - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

THIS IS A SAMPLE.
 TO GET COMPLETE NOTES,
 CALL|TEXT|WHATSAPP 0728 776 317
 OR
 Email: info@masomomsingi.co.ke

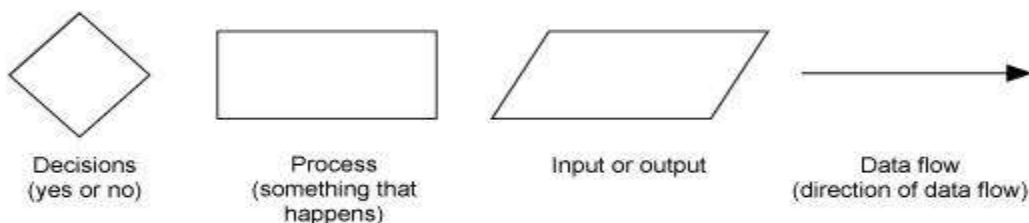
TOPIC 4

DESIGN TOOLS AND METHODS

- **System flowcharts**

System flowcharts are a way of displaying how data flows in a system and how decisions are made to control events.

To illustrate this, symbols are used. They are connected together to show what happens to data and where it goes. The basic ones include:



Symbols used in flow charts

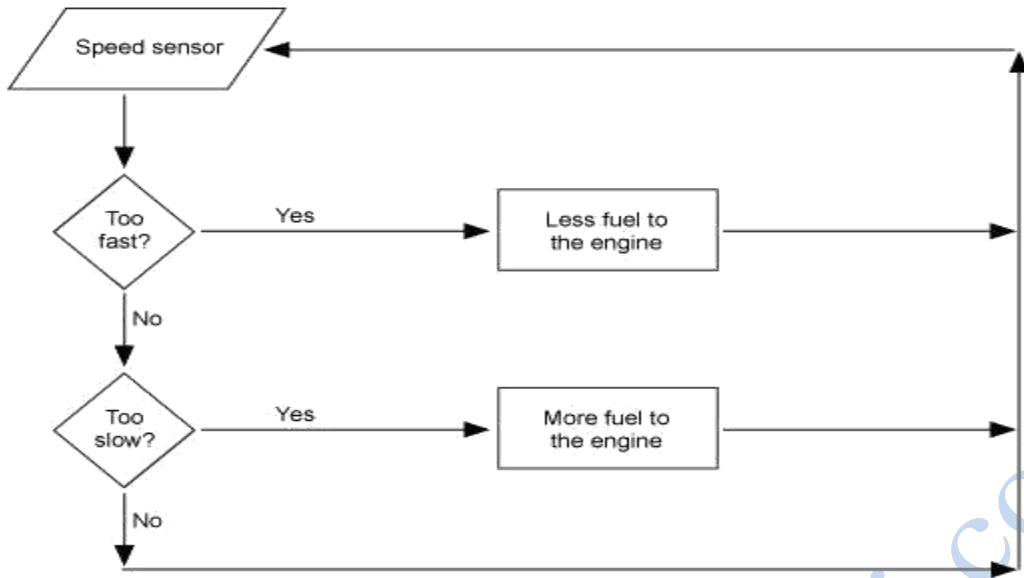
Note that system flow charts are very similar to data flow charts. Data flow charts do not include decisions, they just show the path that data takes, where it is held, processed, and then output.

Using system flowchart ideas

This system flowchart is a diagram for a 'cruise control' for a car. The cruise control keeps the car at a steady speed that has been set by the driver.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

www.masomomsingi.co.ke



A system flowchart for cruise control on a car

The flowchart shows what the outcome is if the car is going too fast or too slow. The system is designed to add fuel, or take it away and so keep the car's speed constant. The output (the car's new speed) is then fed back into the system via the speed sensor.

Other examples of uses for system diagrams include:

- aircraft control
- central heating
- automatic washing machines
- booking systems for airlines

Types of flowchart

Sterneckert (2003) suggested that flowcharts can be modeled from the perspective of different user groups (such as managers, system analysts and clerks) and that there are four general types:

- *Document flowcharts*, showing controls over a document-flow through a system
- *Data flowcharts*, showing controls over a data-flow in a system
- *System flowcharts* showing controls at a physical or resource level
- *Program flowchart*, showing the controls in a program within a system

Notice that every type of flowchart focuses on some kind of control, rather than on the particular flow itself.

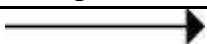

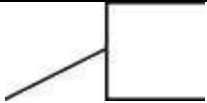
However, there are several of these classifications. For example, Andrew Veronis (1978) named three basic types of flowcharts: the *system flowchart*, the *general flowchart*, and the *detailed flowchart*. That same year Marilyn Bohl (1978) stated "in practice, two kinds of flowcharts are used in solution planning: *system flowcharts* and *program flowcharts*. More recently Mark A. Fryman (2001) stated that there are more differences: "Decision flowcharts, logic flowcharts, systems flowcharts, product flowcharts, and process flowcharts are just a few of the different types of flowcharts that are used in business and government".


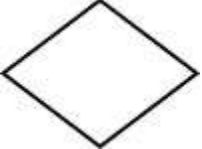





In addition, many diagram techniques exist that are similar to flowcharts but carry a different name, such as UMLactivity diagrams.

Flowchart building blocks

Common Shapes

The following are some of the commonly used shapes used in flowcharts. Generally, flowcharts flow from top to bottom and left to right.

Shape	Name	Description
	Flow Line	An arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to. The line for the arrow can be solid or dashed. The meaning of the arrow with dashed line may differ from one flowchart to another and can be defined in the legend.
	On-Page Connector	Generally represented with a circle, showing where multiple control flows converge in a single exit flow. It will have more than one arrow coming into it, but only one going out. In simple cases, one may simply have an arrow point to another arrow instead. These are useful to represent an iterative process (what in Computer Science is called a loop). A loop may, for example, consist of a connector where control first enters, processing steps, a conditional with one arrow exiting the loop, and one going back to the connector. For additional clarity, wherever two lines accidentally cross in the drawing, one of them may be drawn with a small semicircle over the other, showing that no connection is intended.
	Annotation	Annotations represent comments or remarks about the flowchart. Like comments found in high-level programming languages, they have no effect on the interpretation or behavior of the flowchart. Sometimes, the shape consists of a box with dashed (or dotted) lines.

	Terminal	Represented as circles, ovals, stadiums or rounded (fillet) rectangles. They usually contain the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit inquiry" or "receive product".
	Decision	Represented as a diamond (rhombus) showing where a decision is necessary, commonly a Yes/No question or True/False test. The conditional symbol is peculiar in that it has two arrows coming out of it, usually from the bottom point and right point, one corresponding to Yes or True, and one corresponding to No or False. (The arrows should always be labeled.) More than two arrows can be used, but this is normally a clear indicator that a complex decision is being taken, in which case it may need to be broken-down further or replaced with the "predefined process" symbol. Decision can also help in the filtering of data.
	Input/output	Represented as a parallelogram. Involves receiving data and displaying processed data. Can only move from input to output and not vice versa. Examples: Get X from the user; display X.
	Predefined Process	Represented as rectangles with double-struck vertical edges; these are used to show complex processing steps which may be detailed in a separate flowchart. Example: PROCESS-FILES. One subroutine may have multiple distinct entry points or exit flows (see coroutine). If so, these are shown as labeled 'wells' in the rectangle, and control arrows connect to these 'wells'.
	Process	Represented as rectangles. This shape is used to show that something is performed. Examples: "Add 1 to X", "replace identified part", "save changes", etc....
	Preparation	Represented as a hexagon. May also be called initialization. Shows operations which have no effect other than preparing a value for a subsequent conditional or decision step. Alternatively, this shape is used to replace the Decision Shape in the case of conditional looping.
	Off-Page Connector	Represented as a home plate-shaped pentagon. Similar to the on-page connector except allows for placing a connector that connects to another page.

Other Shapes

A typical flowchart from older basic computer science textbooks may have the following kinds of symbols:

Labeled connectors

Represented by an identifying label inside a circle. Labeled connectors are used in complex or multi-sheet diagrams to substitute for arrows. For each label, the "outflow" connector must always be unique, but there may be any number of "inflow" connectors. In this case, a junction in control flow is implied.

Concurrency symbol

Represented by a double transverse line with any number of entry and exit arrows. These symbols are used whenever two or more control flows must operate simultaneously. The exit flows are activated concurrently, when all of the entry flows have reached the concurrency symbol. A concurrency symbol with a single entry flow is a *fork*; one with a single exit flow is a *join*.

Data-flow extensions

A number of symbols have been standardized for data flow diagrams to represent data flow, rather than control flow. These symbols may also be used in control flowcharts (e.g. to substitute for the parallelogram symbol).

- A *Document* represented as a rectangle with a wavy base;
- A *Manual input* represented by quadrilateral, with the top irregularly sloping up from left to right. An example would be to signify data-entry from a form;
- A *Manual operation* represented by a trapezoid with the longest parallel side at the top, to represent an operation or adjustment to process that can only be made manually.
- A *Data File* represented by a cylinder.
- *Case tools*

Computer-aided software engineering (CASE) is the domain of software tools used to design and implement applications. CASE tools are similar to and were partly inspired by Computer Aided Design (CAD) tools used to design hardware products. CASE tools are used to develop software that is high-quality, defect-free, and maintainable. CASE software is often associated with methodologies for the development of information systems together with automated tools that can be used in the software development process.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

TOPIC 5

SOFTWARE QUALITY

- The degree to which a system, component, or process meets specified requirements.
- The degree to which a system, component or process meets customer or user needs or expectations

- *Quality control and assurance*

Software Quality Control is the set of procedures used by organizations to ensure that a software product will meet its quality goals at the best value to the customer, and to continually improve the organization's ability to produce software products in the future.

Software quality control refers to specified functional requirements as well as non-functional requirements such as supportability, performance and usability. It also refers to the ability for software to perform well in unforeseeable scenarios and to keep a relatively low defect rate.

Definition 2

Software Quality Control is a function that checks whether a software component or supporting artifact meets requirements, or is "fit for use". Software Quality Control is commonly referred to as Testing.

Quality Control Activities

- Check that assumptions and criteria for the selection of data and the different factors related to data are documented.

- Check for transcription errors in data input and reference.
- Check the integrity of database files.
- Check for consistency in data.
- Check that the movement of inventory data among processing steps is correct.
- Check for uncertainties in data, database files etc.
- Undertake review of internal documentation.
- Check methodological and data changes resulting in recalculations.
- Undertake completeness checks.
- Compare Results to previous Results.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomssingi.co.ke

www.masomomssingi.co.ke

Software Control Methods

- Rome laboratory Software framework
- Goal Question Metric Paradigm
- Risk Management Model
- The Plan-Do-Check-Action Model of Quality Control
- Total Software Quality Control
- Spiral Model Of Software Developments

Software development requires quality control.

These specified procedures and outlined requirements leads to the idea of Verification and Validation and software testing.

It is distinct from software quality assurance which encompasses processes and standards for ongoing maintenance of high quality of products, e.g. software deliverables, documentation and processes - avoiding defects. Whereas software quality control is a validation of artifacts compliance against established criteria - finding defects

- **Software quality assurance (SQA)** consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.

SQA encompasses the entire software development process, which includes processes such as requirements definition, software design, coding, source code control, code reviews, software configuration management, testing, release management, and product integration. SQA is organized into goals, commitments, abilities, activities, measurements, and verifications

Differences between **Software Quality Assurance (SQA)** and **Software Quality Control (SQC)**:

Many people still use the term Quality Assurance (QA) and Quality Control (QC) interchangeably but this should be discouraged.

Criteria	Software Quality Assurance (SQA)	Software Quality Control (SQC)
<i>Definition</i>	SQA is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products). The activities establish and evaluate the processes that produce products.	SQC is a set of activities for ensuring quality in software products. The activities focus on identifying defects in the actual products produced.
<i>Focus</i>	Process focused	Product focused
<i>Orientation</i>	Prevention oriented	Detection oriented
<i>Breadth</i>	Organization wide	Product/project specific
<i>Scope</i>	Relates to all products that will ever be created by a process	Relates to specific product
<i>Activities</i>	<ul style="list-style-type: none"> • Process Definition and Implementation • Audits • Training 	<ul style="list-style-type: none"> • Reviews • Testing

- **Software quality factors and metrics**

Quality Factors

- **Functionality** - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
 - Suitability
 - Accuracy
 - Interoperability
 - Compliance
 - Security
- **Reliability** - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
 - Maturity
 - Recoverability

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR

Email: info@masomomsingi.co.ke

www.masomomsingi.co.ke

TOPIC 6

SOFTWARE CODING

Coding

- Coding is undertaken once the design phase is complete and the design documents have been successfully reviewed.
- In the coding phase every module identified and specified in the design document is independently coded and unit tested.
- Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called coding standards.
- Most software development organizations formulate their own coding standards that suit them most, and require their engineers to follow these standards rigorously.

Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop.

- representative coding standards
- representative coding Guidelines

Coding Standards

Programmers spend more time reading code than writing code

- They read their own code as well as other programmers' code.
- Readability is enhanced if some coding conventions are followed by all.
- Coding standards provide these guidelines for programmers.
- Generally are regarding naming, file organization, statements/declarations,
- Naming conventions.

Coding Guidelines

- Package name should be in lower case (mypackage, edu.iitk.maths)
- Type names should be nouns and start with uppercase (Day, DateOfBirth,...)
- Var names should be nouns in lowercase; vars with large scope should have long names; loop iterators should be i, j, k...
- Const names should be all caps
- Method names should be verbs starting with lower case (eg getValue())
- Prefix *is* should be used for Boolean method
- *Coding styles and characteristics*

Programming style is a set of rules or guidelines used when writing the source code for a computer program. It is often claimed that following a particular programming style will help

programmers to read and understand source code conforming to the style, and help to avoid introducing errors.

A classic work on the subject was *The Elements of Programming Style*, written in the 1970s, and illustrated with examples from the FORTRAN and PL/I languages prevalent at the time.

The programming style used in a particular program may be derived from the coding conventions of a company or other computing organization, as well as the preferences of the author of the code. Programming styles are often designed for a specific programming language (or language family): style considered good in C source code may not be appropriate for BASIC source code, and so on. However, some rules are commonly applied to many languages.

Elements of good style

Good style is a subjective matter, and is difficult to define. However, there are several elements common to a large number of programming styles. The issues usually considered as part of programming style include the layout of the source code, including indentation; the use of white space around operators and keywords; the capitalization or otherwise of keywords and variable names; the style and spelling of user-defined identifiers, such as function, procedure and variable names; and the use and style of comments.

Code appearance

Programming styles commonly deal with the visual appearance of source code, with the goal of readability. Software has long been available that formats source code automatically, leaving coders to concentrate on naming, logic, and higher techniques. As a practical point, using a computer to format source code saves time, and it is possible to then enforce company-wide standards without debates.

Indentation

Indent styles assist in identifying control flow and blocks of code. In some programming languages indentation is used to delimit logical blocks of code; correct indentation in these cases is more than a matter of style. In other languages indentation and white space do not affect function, although logical and consistent indentation makes code more readable

Coding styles- Coding guidelines provide only general suggestions regarding the coding style to be followed.

- 1) **Do not use a coding style that is too clever or too difficult to understand:** Code should be easy to understand. Clever coding can obscure meaning of the code and hamper understanding. It also makes maintenance difficult.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317

OR

Email: info@masomomsingi.co.ke

www.masomomsingi.co.ke

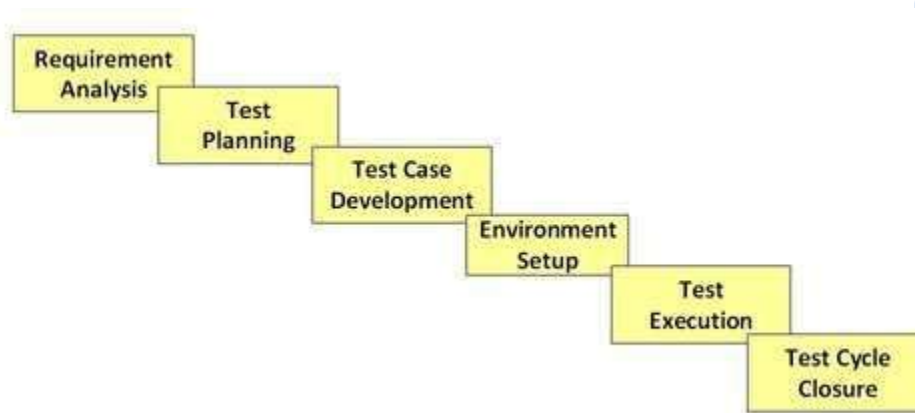
TOPIC 7

SOFTWARE TESTING

- *Software testing life cycle*

Contrary to popular belief, Software Testing is not a just a single activity. It consists of series of activities carried out methodologically to help certify your software product. These activities (stages) constitute the Software Testing Life Cycle (STLC).

The different stages in Software Test Life Cycle -



Each of these stages has a definite Entry and Exit criteria, Activities & Deliverables associated with it.

In an Ideal world you will not enter the next stage until the exit criteria for the previous stage is met. But practically this is not always possible. So for this tutorial , we will focus of activities and deliverables for the different stages in STLC

What is Software Testing Life Cycle (STLC)

Software Testing Life Cycle refers to a testing process which has specific steps to be executed in a definite sequence to ensure that the quality goals have been met. In STLC process, each activity is carried out in a planned and systematic way. Each phase has different goals and deliverables. Different organizations have different phases in STLC; however the basis remains the same.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

TOPIC 8

SOFTWARE ACQUISITION METHODS

- *Software costing*

Introduction

It has been surveyed that nearly one-third projects overrun their budget and late delivered and two-thirds of all major projects substantially overrun their original estimates. The accurate prediction of software development costs is a critical issue to make the good management decisions and accurately determining how much effort and time a project required for both project managers as well as system analysts and developers. Without reasonably accurate cost estimation capability, project managers cannot determine how much time and manpower cost the project should take and that means the software portion of the project is out of control from its beginning; system analysts cannot make realistic hardware-software tradeoff analyses during the system design phase; software project personnel cannot tell managers and customers that their proposed budget and schedule are unrealistic. This may lead to optimistic over promising on software development and the inevitable overruns and performance compromises as a consequence. But, actually huge overruns resulting from inaccurate estimates are believed to occur frequently.

The overall process of developing a cost estimate for software is not different from the process for estimating any other element of cost. There are, however, aspects of the process that are peculiar to software estimating. Some of the unique aspects of software estimating are driven by the nature of software as a product. Other problems are created by the nature of the estimating methodologies. Software cost estimation is a continuing activity which starts at the proposal stage and continues through the life time of a project. Continual cost estimation is to ensure that the spending is in line with the budget.

Cost estimation is one of the most challenging tasks in project management. It is to accurately estimate needed resources and required schedules for software development projects. The software estimation process includes estimating the size of the software product to be produced, estimating the effort required, developing preliminary project schedules, and finally, estimating overall cost of the project.

It is very difficult to estimate the cost of software development. Many of the problems that plague the development effort itself are responsible for the difficulty encountered in estimating that effort. One of the first steps in any estimate is to understand and define the system to be estimated. Software, however, is intangible, invisible, and intractable. It is inherently more

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

difficult to understand and estimate a product or process that cannot be seen and touched. Software grows and changes as it is written. When hardware design has been inadequate, or when hardware fails to perform as expected, the "solution" is often attempted through changes to the software. This change may occur late in the development process, and sometimes results in unanticipated software growth.

After 20 years research, there are many software cost estimation methods available including algorithmic methods, estimating by analogy, expert judgment method, price towin method, top-down method, and bottom-up method. No one method is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other. To understand their strengths and weaknesses is very important when you want to estimate your projects.

Expert Judgment Method

Expert judgment techniques involve consulting with software cost estimation expert or a group of the experts to use their experience and understanding of the proposed project to arrive at an estimate of its cost.

Generally speaking, a group consensus technique, Delphi technique, is the best way to be used. The strengths and weaknesses are complementary to the strengths and weaknesses of algorithmic method.

To provide a sufficiently broad communication bandwidth for the experts to exchange the volume of information necessary to calibrate their estimates with those of the other experts, a wideband Delphi technique is introduced over standard Deliphi technique.

The estimating steps using this method:

1. Coordinator present each expert with a specification and an estimation form.
2. Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
3. Experts fill out forms anonymously
4. Coordinator prepares and distributes a summary of the estimation on an iteration form.
5. Coordinator calls a group meeting, specially focusing on having the experts discuss points where their estimates varied widely.
6. Experts fill out forms, again anonymously, and steps 4 and 6 are iterated for as many rounds as appropriate.

The wideband Delphi Technique has subsequently been used in a number of studies and cost estimation activities. It has been highly successful in combining the free discuss advantages of the group meeting technique and advantage of anonymous estimation of the standard Delphi Technique.

The advantages of this method are:

- The experts can factor in differences between past project experience and requirements of the proposed project. The experts can factor in project impacts caused by new technologies, architectures, applications and languages involved in the future project and can also factor in exceptional personnel characteristics and interactions, etc.

The disadvantages include:

- This method cannot be quantified.
- It is hard to document the factors used by the experts or experts-group.
- Expert may be some biased, optimistic, and pessimistic, even though they have been decreased by the group consensus.
- The expert judgment method always compliments the other cost estimating methods such as algorithmic method.

Estimating by Analogy

Estimating by analogy means comparing the proposed project to previously completed similar project where the project development information is known. Actual data from the completed projects are extrapolated to estimate the proposed project. This method can be used either at system-level or at the component-level.

Estimating by analogy is relatively straightforward. Actually in some respects, it is a systematic form of expert judgment since experts often search for analogous situations so as to inform their opinion.

The steps using estimating by analogy are:

1. Characterizing the proposed project.
2. Selecting the most similar completed projects whose characteristics have been stored in the historical data base.
3. Deriving the estimate for the proposed project from the most similar completed projects by analogy.

The main advantages of this method are:

- The estimation are based on actual project characteristic data.
- The estimator's past experience and knowledge can be used which is not easy to be quantified.
- The differences between the completed and the proposed project can be identified and impacts estimated.

However there are also some problems with this method,

- Using this method, we have to determine how best to describe projects. The choice of variables must be restricted to information that is available at the point that the prediction require. Possibilities include the type of application domain, the number of inputs, the number of distinct entities referenced, the number of screens and so forth.
- Even once we have characterized the project, we have to determine the similarity and how much confidence can we place in the analogies. Too few analogies might lead to maverick projects being used; too many might lead to the dilution of the effect of the closest analogies. Martin Shepperd etc. introduced the method of finding the analogies by measuring Euclidean distance in n-dimensional space where each dimension corresponds to a variable. Values are standardized so that each dimension contributes equal weight to the process of finding analogies. Generally speaking, two analogies are the most effective.
- Finally, we have to derive an estimate for the new project by using known effort values from the analogous projects. Possibilities include means and weighted means which will give more influence to the closer analogies.

It has been estimated that estimating by analogy is superior technique to estimation via algorithmic model in at least some circumstances. It is a more intuitive method so it is easier to understand the reasoning behind a particular prediction.

Top-Down and Bottom-Up Methods

Top-Down Estimating Method

Top-down estimating method is also called Macro Model. Using top-down estimating method, an overall cost estimation for the project is derived from the global properties of the software project, and then the project is partitioned into various low-level components. The leading method using this approach is Putnam model. This method is more applicable to early cost estimation when only global properties are known. In the early phase of the software development, It is very useful because there are no detailed information available.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

TOPIC 9

CONVERSION STRATEGIES

- *Conversion planning*

Overview

The Conversion Plan describes the strategies involved in converting data from an existing system to another hardware or software environment. It is appropriate to reexamine the original system's functional requirements for the condition of the system before conversion to determine if the original requirements are still valid. An outline of the Conversion Plan is shown below.

INTRODUCTION

This section provides a brief description of introductory material.

Purpose and Scope

This section describes the purpose and scope of the Conversion Plan. Reference the information system name and provide identifying information about the system undergoing conversion.

Points of Contact

This section identifies the System Proponent. Provide the name of the responsible organization and staff (and alternates, if appropriate) who serve as points of contact for the system conversion. Include telephone numbers of key staff and organizations.

Project References

This section provides a bibliography of key project references and deliverables that have been produced before this point in the project development. These documents may have been produced in a previous development life cycle that resulted in the initial version of the system undergoing conversion or may have been produced in the current conversion effort as appropriate.

Glossary

This section contains a glossary of all terms and abbreviations used in the plan. If it is several pages in length, it may be placed in an appendix.

CONVERSION OVERVIEW

This section provides an overview of the aspects of the conversion effort, which are discussed in the subsequent sections.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

TOPIC 9

CONVERSION STRATEGIES

- *Conversion planning*

Overview

The Conversion Plan describes the strategies involved in converting data from an existing system to another hardware or software environment. It is appropriate to reexamine the original system's functional requirements for the condition of the system before conversion to determine if the original requirements are still valid. An outline of the Conversion Plan is shown below.

INTRODUCTION

This section provides a brief description of introductory material.

Purpose and Scope

This section describes the purpose and scope of the Conversion Plan. Reference the information system name and provide identifying information about the system undergoing conversion.

Points of Contact

This section identifies the System Proponent. Provide the name of the responsible organization and staff (and alternates, if appropriate) who serve as points of contact for the system conversion. Include telephone numbers of key staff and organizations.

Project References

This section provides a bibliography of key project references and deliverables that have been produced before this point in the project development. These documents may have been produced in a previous development life cycle that resulted in the initial version of the system undergoing conversion or may have been produced in the current conversion effort as appropriate.

Glossary

This section contains a glossary of all terms and abbreviations used in the plan. If it is several pages in length, it may be placed in an appendix.

CONVERSION OVERVIEW

This section provides an overview of the aspects of the conversion effort, which are discussed in the subsequent sections.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317

OR

Email: info@masomomsingi.co.ke

www.masomomsingi.co.ke

TOPIC 10

DOCUMENTATION AND COMMISSIONING

- *Objectives of systems documentation*
- *Use of systems documentation*

Requirements

The question of what is the purpose of system documentation is difficult to answer. Below are some possible uses of system documentation.

a) Introduction / overview

System documentation can provide an introduction and overview of systems. New Administrators, contractors and other staff may need to familiarize themselves with a system; the first thing that will be requested is any system documentation. To avoid staff having to waste time discovering the purpose of a system, how it is configured etc. system documentation should provide an Introduction.

b) Disaster Recovery

Many systems are supported by disaster recovery arrangements, but even in such circumstances, the recovery can still fail. There may be a need to re-build a system from scratch at least to the point where a normal restore from backup can be done. To make a rebuild possible it will be necessary to have documentation that provides answers to the configuration choices. For example it is important to re-build the system with the correct size of file systems to avoid trying to restore data to a file system that has been made too small. In some circumstances certain parameters can be difficult to change at a later date. When rebuilding a system it may be important to configure networking with the original parameters both to avoid conflicts with other systems on the network and because these may be difficult to change subsequently.

c) OS or Application re-load

Even when a disaster has not occurred, there may be times when it is necessary to reload an Operating System or Application; this can either be as part of a major version upgrade or a drastic step necessary to solve problems. In such circumstances it is important to know how an OS or Application has been configured.

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317

OR

Email: info@masomomsingi.co.ke

www.masomomsingi.co.ke

TOPIC 11

SOFTWARE MAINTENANCE AND EVOLUTION

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

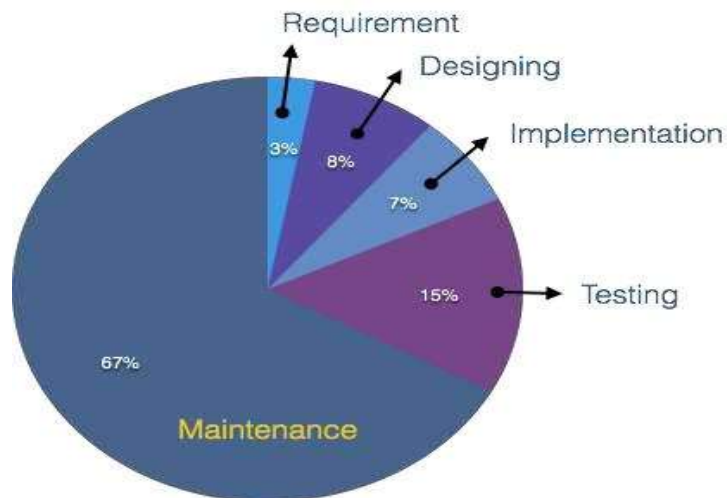
Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost goes high, such as:

Real-world factors affecting Maintenance Cost

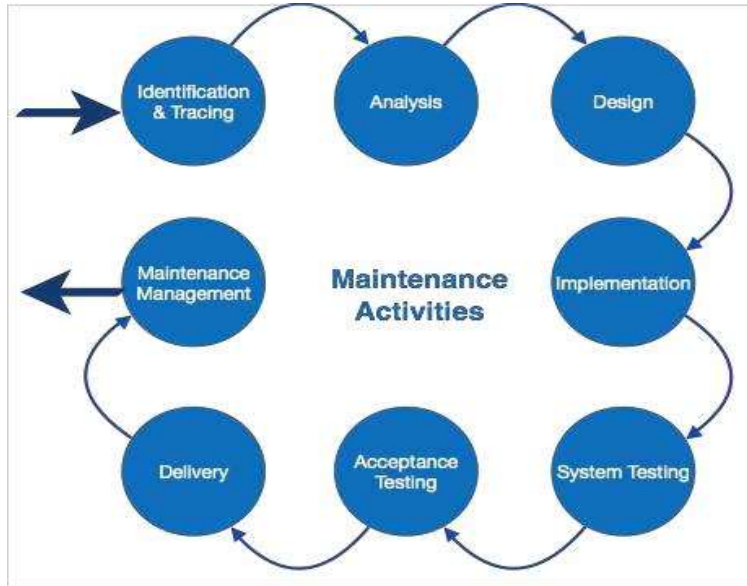
- The standard age of any software is considered up to 10 to 15 years.
- Older software, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced software on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

- **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

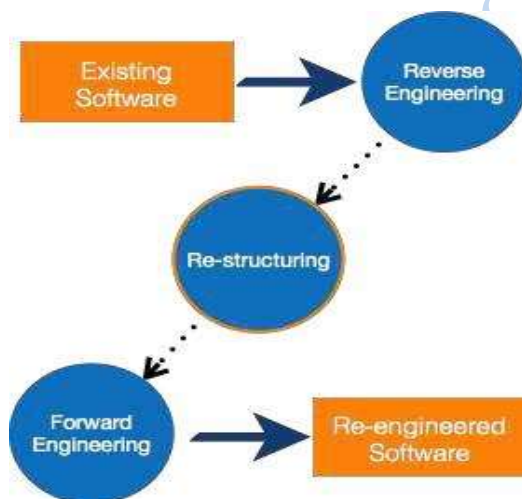
Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

TOPIC 12

AUDITING INFORMATION SYSTEMS

- *Overview of information systems audit*

An IT audit is different from a financial statement audit. While a financial audit's purpose is to evaluate whether an organization is adhering to standard accounting practices, the purposes of an IT audit are to evaluate the system's internal control design and effectiveness. This includes, but is not limited to, efficiency and security protocols, development processes, and IT governance or oversight. Installing controls are necessary but not sufficient to provide adequate security. People responsible for security must consider if the controls are installed as intended, if they are effective if any breach in security has occurred and if so, what actions can be done to prevent future breaches. These inquiries must be answered by independent and unbiased observers. These observers are performing the task of information systems auditing. In an Information Systems (IS) environment, an audit is an examination of information systems, their inputs, outputs, and processing.

The primary functions of an IT audit are to evaluate the systems that are in place to guard an organization's information. Specifically, information technology audits are used to evaluate the organization's ability to protect its information assets and to properly dispense information to authorized parties. The IT audit aims to evaluate the following:

Will the organization's computer systems be available for the business at all times when required? (known as availability) Will the information in the systems be disclosed only to authorized users? (known as security and confidentiality) Will the information provided by the system always be accurate, reliable, and timely? (Measures the integrity) In this way, the audit

hopes to assess the risk to the company's valuable asset (its information) and establish methods of minimizing those risks.

Also Known As: Information Systems Audit, ADP audits, EDP audits, computer audits

Types of IT audits

Various authorities have created differing taxonomies to distinguish the various types of IT audits. Goodman & Lawless state that there are three specific systematic approaches to carry out an IT audit:

- **Technological innovation process audit.** This audit constructs a risk profile for existing and new projects. The audit will assess the length and depth of the

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317
OR
Email: info@masomomsingi.co.ke

company's experience in its chosen technologies, as well as its presence in relevant markets, the organization of each project, and the structure of the portion of the industry that deals with this project or product, organization and industry structure.

- **Innovative comparison audit.** This audit is an analysis of the innovative abilities of the company being audited, in comparison to its competitors. This requires examination of company's research and development facilities, as well as its track record in actually producing new products.
- **Technological position audit:** This audit reviews the technologies that the business currently has and that it needs to add. Technologies are characterized as being either "base", "key", "pacing" or "emerging".

Others describe the spectrum of IT audits with five categories of audits:

- **Systems and Applications:** An audit to verify that systems and applications are appropriate, are efficient, and are adequately controlled to ensure valid, reliable, timely, and secure input, processing, and output at all levels of a system's activity.
- **Information Processing Facilities:** An audit to verify that the processing facility is controlled to ensure timely, accurate, and efficient processing of applications under normal and potentially disruptive conditions.
- **Systems Development:** An audit to verify that the systems under development meet the objectives of the organization and to ensure that the systems are developed in accordance with generally accepted standards for systems development.
- **Management of IT and Enterprise Architecture:** An audit to verify that IT management has developed an organizational structure and procedures to ensure a controlled and efficient environment for information processing.
- **Client/Server, Telecommunications, Intranets, and Extranets:** An audit to verify that telecommunications controls are in place on the client (computer receiving services), server, and on the network connecting the clients and servers.

And some lump all IT audits as being one of only two types: "**general control review**" audits or "**application control review**" audits.

A number of IT Audit professionals from the Information Assurance realm consider there to be three fundamental types of controls regardless of the type of audit to be performed, especially in the IT realm. Many frameworks and standards try to break controls into different disciplines or arenas, terming them —Security Controls—, —Access Controls—, —IA Controls— in an effort to define the types of controls involved. At a more fundamental level, these controls can be shown to consist of three types of fundamental controls: Protective/Preventative Controls, Detective Controls and Reactive/Corrective Controls.

In an IS system, there are two types of auditors and audits: internal and external. IS auditing is usually a part of accounting internal auditing, and is frequently performed by corporate internal auditors. An external auditor reviews the findings of the internal audit as well as the inputs, processing and outputs of information systems. The external audit of information systems is frequently a part of the overall external auditing performed by a Certified Public Accountant (CPA) firm.

IS auditing considers all the potential hazards and controls in information systems. It focuses on issues like operations, data, integrity, software applications, security, privacy, budgets and expenditures, cost control, and productivity. Guidelines are available to assist auditors in their jobs, such as those from Information Systems Audit and Control Association.

IT Audit process

The following are basic steps in performing the Information Technology Audit Process:

1. Planning
2. Studying and Evaluating Controls
3. Testing and Evaluating Controls
4. Reporting
5. Follow-up
6. reports

Security

Auditing information security is a vital part of any IT audit and is often understood to be the primary purpose of an IT Audit. The broad scope of auditing information security includes such topics as data centers (the physical security of data centers and the logical security of databases, servers and network infrastructure components), networks and application security. Like most technical realms, these topics are always evolving; IT auditors must constantly continue to expand their knowledge and understanding of the systems and environment & pursuit in system company.

Several training and certification organizations have evolved. Currently, the major certifying bodies, in the field, are the Institute of Internal Auditors (IIA), the SANS Institute (specifically, the audit specific branch of SANS and GIAC) and ISACA. While CPAs and other traditional auditors can be engaged for IT Audits, organizations are well advised to require that individuals with some type of IT specific audit certification are employed when validating the controls surrounding IT systems.

History of IT Auditing

The concept of IT auditing was formed in the mid-1960s. Since that time, IT auditing has gone through numerous changes, largely due to advances in technology and the incorporation of technology into business.

Currently, there are many IT dependent companies that rely on the Information Technology in order to operate their business e.g. Telecommunication or Banking company. For the other types of business, IT plays the big part of company including the applying of workflow instead of using the paper request form, using the application control instead of manual control which is more reliable or implementing the ERP application to facilitate the organization by using only 1 application. According to these, the importance of IT Audit is constantly increased. One of the most important roles of the IT Audit is to audit over the critical system in order to support the Financial audit or to support the specific regulations announced e.g. SOX.

Audit personnel

Qualifications

The CISM and CAP credentials are the two newest security auditing credentials, offered by the ISACA and (ISC) respectively. Strictly speaking, only the CISA or GSNA title would sufficiently demonstrate competences regarding both information technology and audit aspects with the CISA being more audit focused and the GSNA being more information technology focused.

Outside of the US, various credentials exist. For example, the Netherlands has the **RE** credential (as granted by the NOREA [Dutch site] IT-auditors' association), which among others requires a post-graduate IT-audit education from an accredited university, subscription to a Code of Ethics, and adherence to continuous education requirement

- *Auditing computer resources*
- *Audit techniques*
- *Audit applications*

THIS IS A SAMPLE.
TO GET COMPLETE NOTES,
CALL|TEXT|WHATSAPP 0728 776 317

OR

Email: info@masomomsingi.co.ke

FOLLOW US ON SOCIAL MEDIA @ [Masomo Msingi](#) for updates

www.masomomsingi.co.ke