Subjct Name: Principles of Software Engineering

Subject Code: 66661

Teacher Name:- Md. Tofael Alam Siddiquee

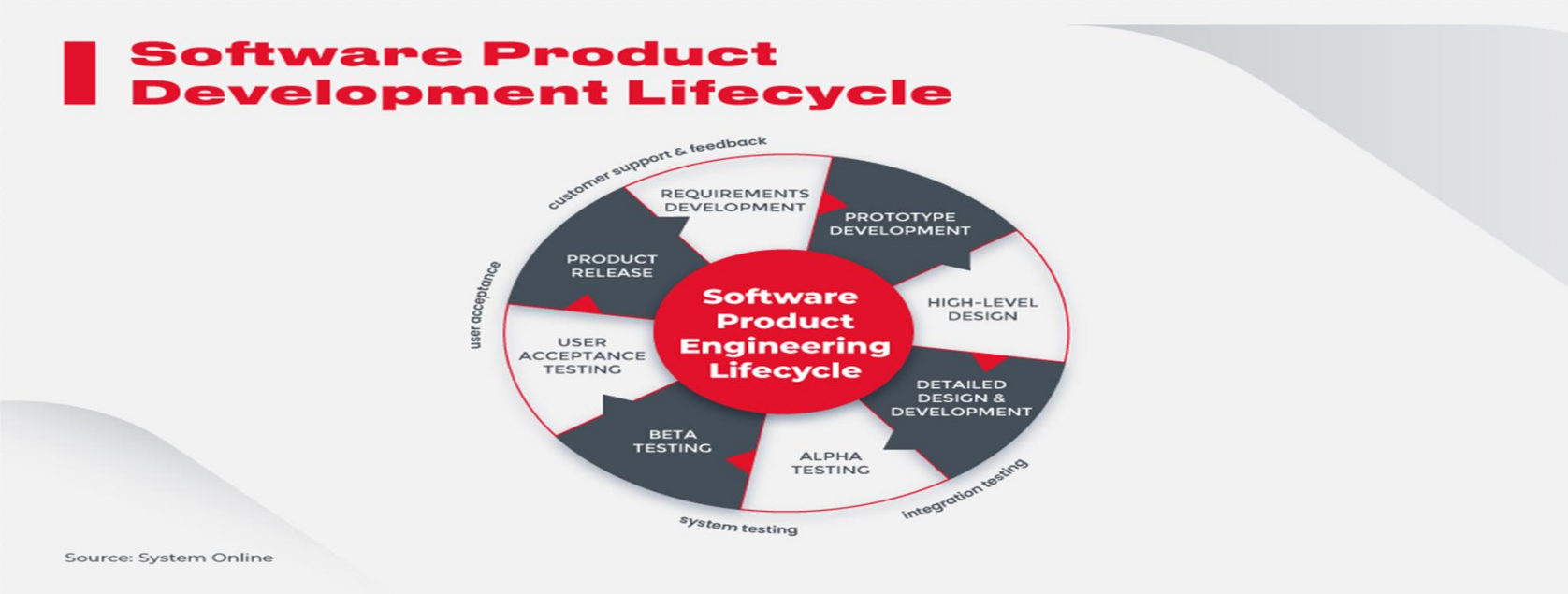
Designation : Instructor (Tech/Computer)

Department of Computer Science and Technology

Chapter : one

Topic :Concepts of Software Engineering

Software engineering:- Software engineering is the process of developing, testing and deploying computer applications to solve real-world problems by adhering to a set of engineering principles and best practices.



Evolution of software engineering : Software Evolution is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc. The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.

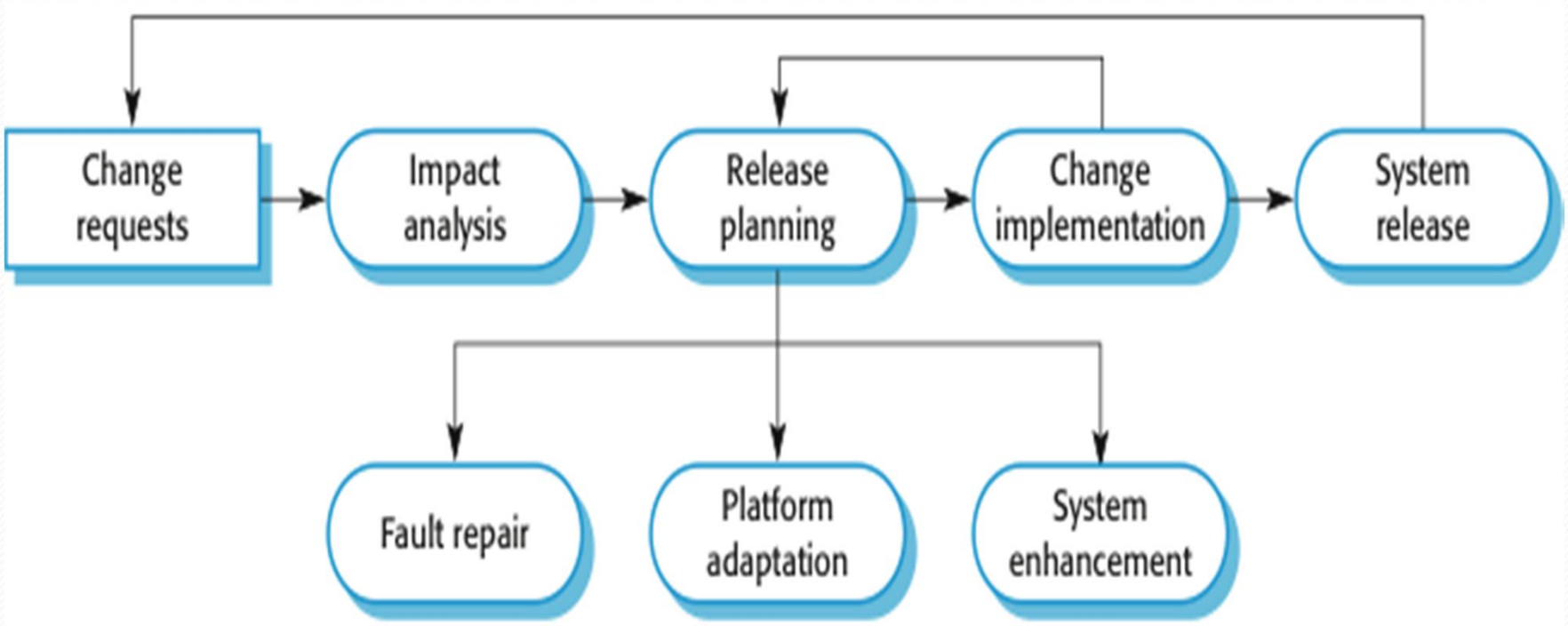


Fig: Evolution of software Engineering

Importance Evolution of software engineering :

:

Importance:

- ✓ A large amount of resources are spent in an organization's software.
- ✓ Defects are found in existing software, which require fixing.
- ✓ New features need to be added to the existing software to keep it up-to-date.
- ✓ It is important to be continually improving the performance and reliability of software.
- ✓ Security of the software must be upgraded in order to stay relevant and safe.

E-Type software evolution:

There are following E-Type software evolution are given below.

Continues changed:

Similarly to the E-Type software evolution, the real-world requirements must change in the program and the environment becomes less progressively used.

Conservation of familiarity:

The familiarity with the software evolution about how it was developing, why it was developing in that specific manner. The lifetime of over the system the approximately incremental change in each release.

Continues growth:

In this evolution system, the functionality of the system is continually increasing and maintain the user specification. In the continued growth the measure of implementing the changes grows become to the lifestyle changes of the business.

Organization stability:
The lifecycle of the programs, its rate of development is approximately constant.

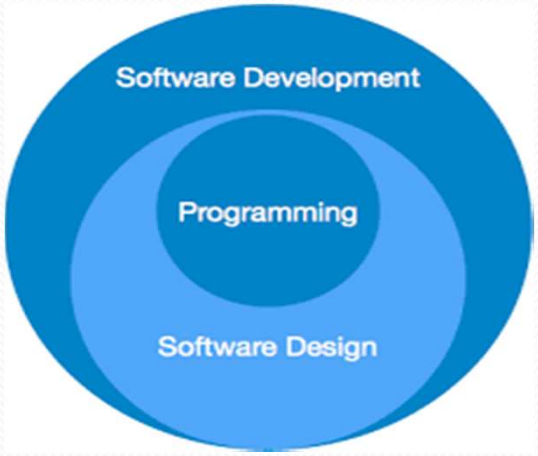
Reducing or Declining quality:
In this evolution, the quality of the system is declining unless and operational environment is changing.

Self-regulation:
In which the process of E-Type software evolution the distributing of self-regulation is the product and process are a measure to normal.

Feedback system:
In which the feedback process the multi-loop are constitute working and the multi-level feedback system must be treated such as successful to improve.

Software Paradigms:

Software Paradigms provides the first complete compilation of software paradigms commonly used to develop large software applications, with coverage ranging from discrete problems to full-scale applications. The book focuses on providing a structure for understanding a hierarchy of software development approaches, and showing the relationships between the different models. Coverage includes paradigms in design patterns, software components, software architectures, and frameworks. Chapters within each of these sections include design issues related to building and using the paradigm as well as numerous real world applications. A practical overview of the hierarchy of development paradigms, Software Paradigms is an excellent teaching tool for undergraduates and graduates, and a comprehensive and reliable reference for software engineers.



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.

- Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

- Cost- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted. Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

- Quality Management- Better process of software development provides better and quality software product

Characteristics of Good Software:

1. Functionality

The ability of software to perform and function in accordance with design specifications is referred to as its functionality. In simple terms, software systems should function properly; that is, they should execute all of the functions for which they were created. The functions are the things that the end user and the company expect as core system capabilities. All of these functions must be built into the system. Many software applications are created for ease of use. However, the ultimate goal of the software is to give all needed functionality to its users.

2. Efficiency

Efficiency refers to a software's capacity to use human and system resources as effectively and efficiently as possible, such as time, effort, CPU, memory, compute power, network bandwidth, files, databases, and so on. Efficiency is important for the success of a software project. In addition to addressing the goals for which software was created, it must also provide exceptional features designed to help users complete their duties more quickly. Software should make good use of storage space and execute commands in accordance with time constraints.

3. Flexibility

The ability of a software solution to adjust to potential or future changes in its requirements is referred to as software flexibility. When assessing software flexibility, consider how easy it is to add, modify, or remove features without interfering with the current operation. Keeping up with quickly changing markets, technologies, and client expectations is important. Change is unavoidable in software development; it might occur during the development process or as a result of future requirements. As a result, adaptability is highly prized.

4. Usability

The software's user-friendliness is defined by its ease of use. In other words, learning how to utilize the software should take less time and effort. Navigating the software is essential since it determines the path the user travels within the software. This is important for ensuring that visitors stay on your website and have a great experience, resulting in increased sales and brand loyalty.

5. Reliability

A software product's reliability reflects the possibility that it will work without failure for a defined period of time under certain conditions. It assesses a software's ability to sustain its level of performance (offer required functionality) under given conditions for a set amount of time. In general, software reliability is measured by the software's availability. The value should be greater than 99%.

6. Portability

Software portability is an important consideration that must not be overlooked. The capacity to use software in diverse contexts is referred to as portability. This is the ease with which software can be moved from one platform to another without (or with few) changes and achieve comparable results. As basic as it may sound, it refers to software's ability to run across a variety of hardware platforms with no (or few) alterations.

7. Maintainability

The ease with which you can repair, improve, and interpret software code is referred to as maintainability. Maintaining is related to being flexible in several aspects. Maintainability is concerned with the correction of errors and minor changes to software code, whereas flexibility is concerned with substantial functional additions. It also entails keeping the software's services and functions operational.

8. Integrity

Integrity is essential for showing your software's safety, security, and maintainability. Moreover, software that must comply with industry laws and coding standards necessitates strong code integrity. It can be challenging to achieve program integrity. The difficulty, however, may be readily overcome with the correct procedures to improve safety, security, and maintainability. All programs must integrate this feature in these days of rising security concerns.

Chapter : Two

Topic :Basics of Software Develpment Life Sycle (SDFLC)

Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a structured process that enables the production of high-quality, low-cost software, in the shortest possible production time. The goal of the SDLC is to produce superior software that meets and exceeds all customer expectations and demands. The SDLC defines and outlines a detailed plan with stages, or phases, that each encompass their own process and deliverables. Adherence to the SDLC enhances development speed and minimizes project risks and costs associated with alternative methods of production.

SDLC Activities:

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

Communication:

This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms.
He submits his request to the service providing organization in writing.

Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and
tries to bring out as much information as possible on their requirements.
The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given –

- ✓ studying the existing or obsolete system and software,
- ✓ conducting interviews of users and developers,
- ✓ referring to the database or
- ✓ collecting answers from the questionnaires.

Feasibility Study

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

Software Design

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

Coding

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

Integration

Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation

This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance

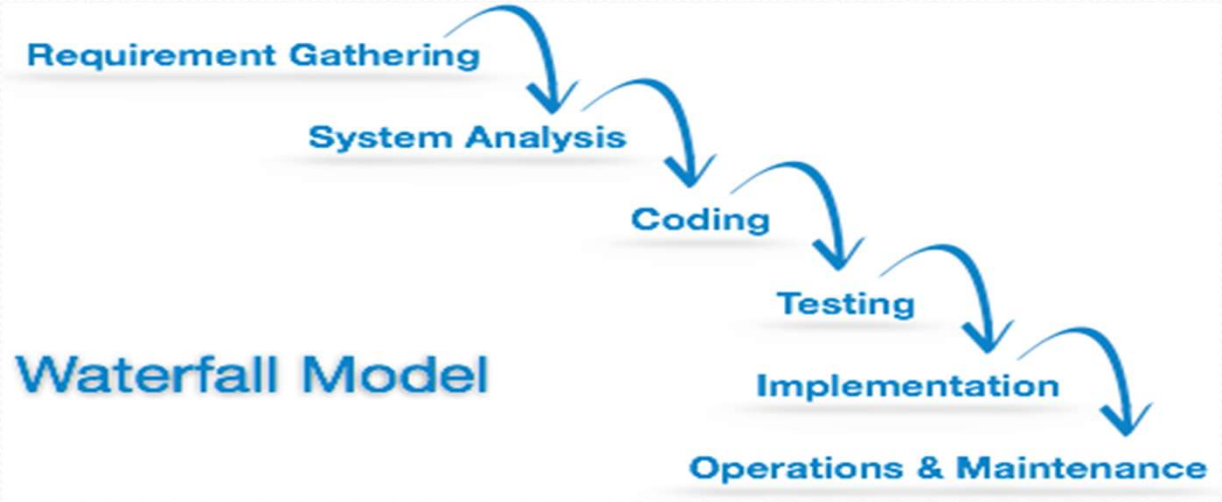
This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

Software Development Paradigm

The software development paradigm helps developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods and procedures, which are expressed clearly and defines software development life cycle. A few of software development paradigms or process models are defined as follows:

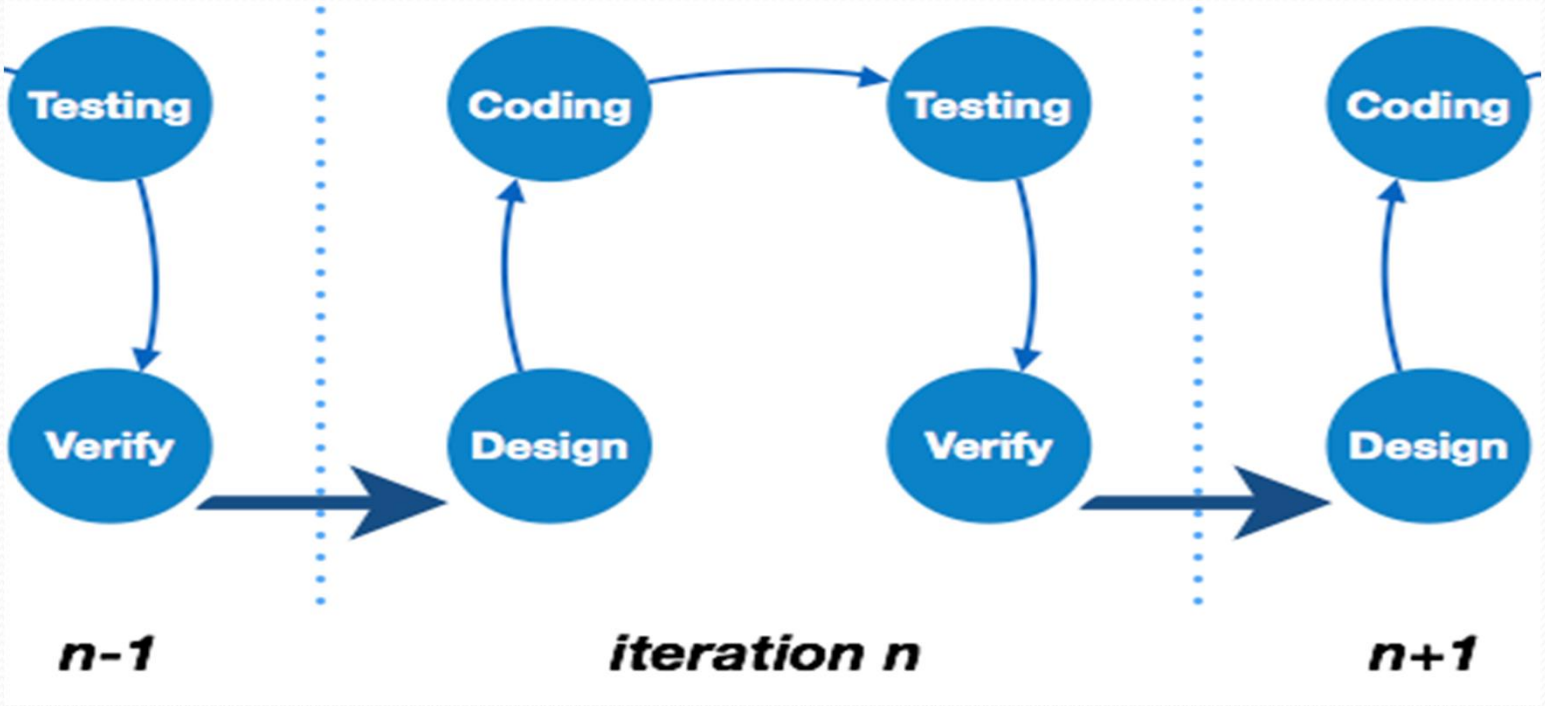
Waterfall Model:

Waterfall model is the simplest model of software development paradigm. It says the all the phases of SDLC will function one after another in linear manner. That is, when the first phase is finished then only the second phase will start and so on.



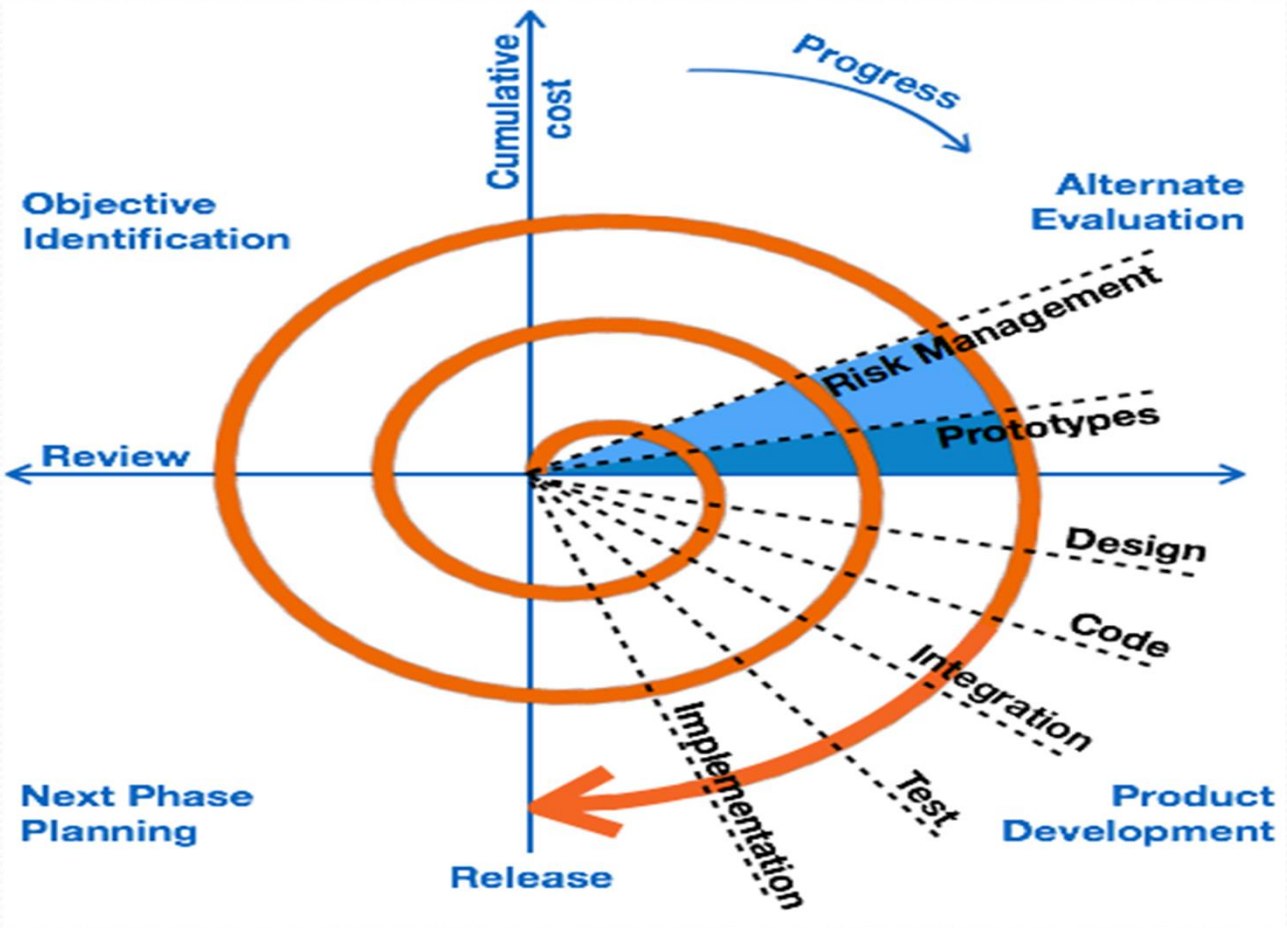
Iterative Model:

This model leads the software development process in iterations. It projects the process of development in cyclic manner repeating every step after every cycle of SDLC process.



Spiral Model:

Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combine it with cyclic process (iterative model).



Agile Development Model:

The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.

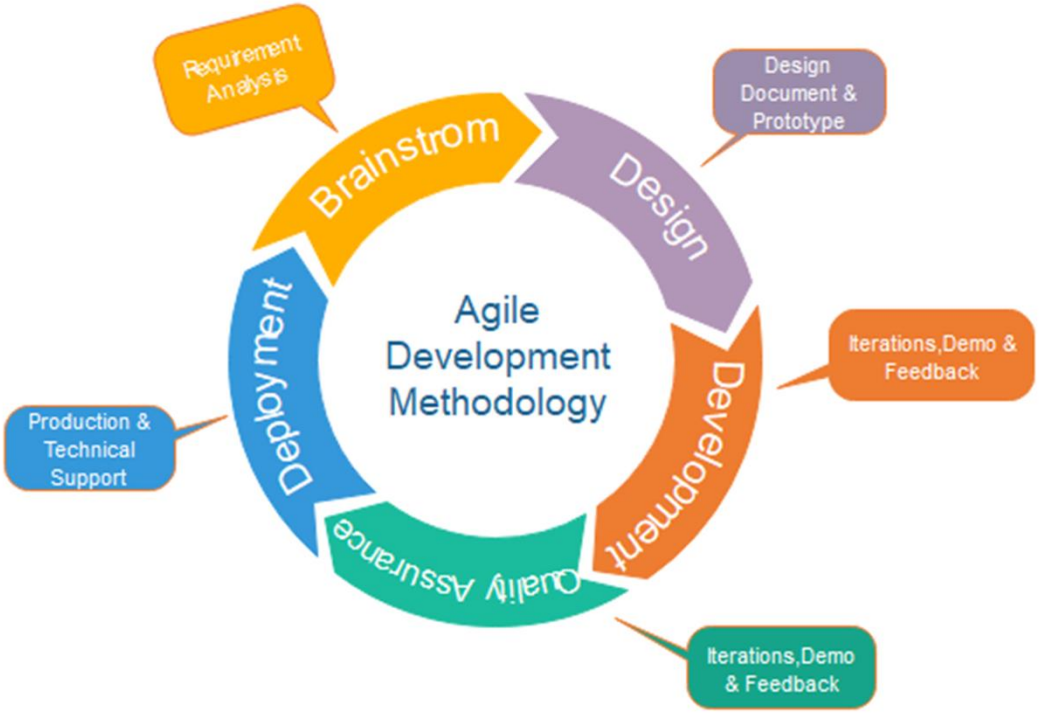


Fig. Agile Model

Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering:

In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements:

When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Chapter : Three

Topic :Software Project Management

Software Project Management:

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of software project management:

Software is said to be an intangible product. Software development is a kind of all new stream in world business and there's very little experience in building software products. Most software products are tailor made to fit client's requirements. The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.

The image above shows triple constraints for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled.
There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factor can severely impact the other two. Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Roles of a Software Project Manager:

A software project manager is a person who undertakes the responsibility of executing the software project. Software project manager is thoroughly aware of all the phases of SDLC that the software would go through. Project manager may never directly involve in producing the end product but he controls and manages the activities involved in production.

Let us see few responsibilities that a project manager shoulders -

Managing People

I. Act as project leader
II. Liaison with stakeholders
III. Managing human resources
IV. Setting up reporting hierarchy etc.

Managing Project

I. Defining and setting up project scope
II. Managing project management activities
III. Monitoring progress and performance
IV. Risk analysis at every phase
V. Take necessary step to avoid or come out of problems
VI. Act as project spokesperson

Software Management Activities:

Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

- ✓ **Project Planning**
- ✓ **Scope Management**
- ✓ **Project Estimation**

Project Planning

Software project planning is task, which is performed before the production of software actually starts. It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production.

Scope Management:

It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done. This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.

During Project Scope management, it is necessary to –

- Define the scope
- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation:

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation :**Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.
- **Effort estimation**The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.
- **Time estimation**Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software.

❖ **Cost estimation**This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider –

- Size of software
- Software quality
- Hardware
- Additional software or tools, licenses etc.
- Skilled personnel with task-specific skills
- Travel involved
- Communication
- Training and support

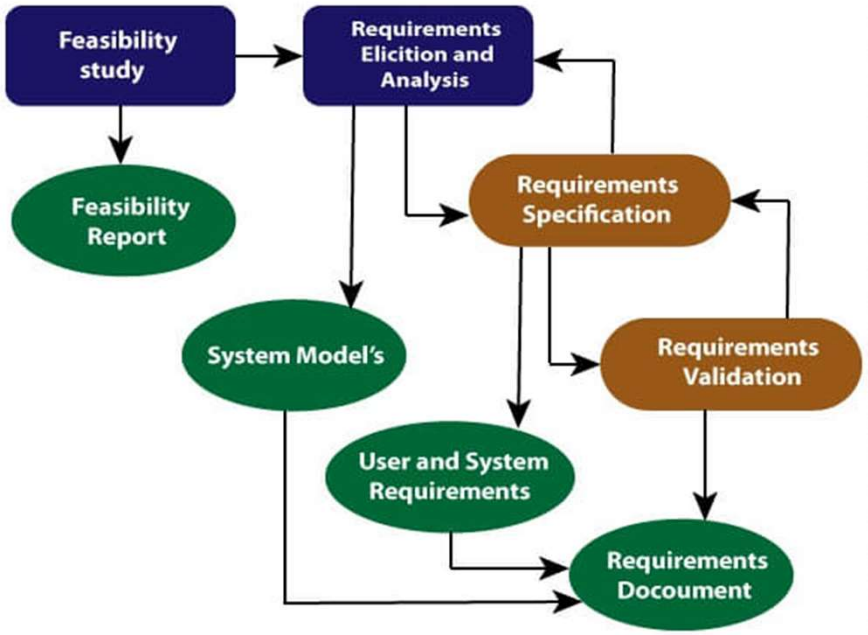
Chapter : Four

Topic : Software Requirement Engineering

Requirement Engineering Process: Requirements engineering is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system.

It is a four-step process, which includes –

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management



Requirement Engineering Process

1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions –

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Chapter : Five

Topic : Software Design Basics, Analysis and Design Tools

Software Design Levels:

Software design yields three levels of results

Architectural Design - The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

High-level Design- The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.

Detailed Design- Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

HIPO Diagram:

HIPO stands for Hierarchical Input Process Output. HIPO diagram assesses the system and facilitates documentation. It organizes the software system's modules into a hierarchy. HIPO diagrams can be used to get a high-level picture of the system's functions.

Structured English:

Structured English makes computer codes intelligible to non-programmers by using English words to describe each step in the process of running a program. The structures designed to tell the computer what to do in a logical, step-by-step way. Non-programmers, however, are unable to read the actual symbols and directions in the code, so people have created programming languages that use straightforward English words to express what happens as the real code is running.

Data Dictionary:

A data dictionary contains metadata i.e., data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary, it is only handled by the database administrators.

Chapter : Six

Topic : Software Design Strategies

Structured Design:

Structured Design is a systematic methodology to determine design specification of software. The basic principles, tools and techniques of structured methodology are discussed in this chapter. It covers the four components of software design, namely, architectural design, detail design, data design and interface design. This chapter describes the following concepts, tools and techniques of structured design:

- *Coupling and cohesion*
- *Structure chart*
- *Transaction analysis and transform analysis*
- *Program flowchart*
- *Structured flowchart*
- *HIPO documentation*

Function Oriented Design:

Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.

2. Object Oriented Design:

Object oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that are to be manipulated by the program. Thus, it is orthogonal to function -oriented design. Object-oriented design begins with an examination of the real world “things”. These things are characteristics individually in terms of their attributes and behavior.

Chapter : Seven

Topic : User Interface Design

User interface:

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

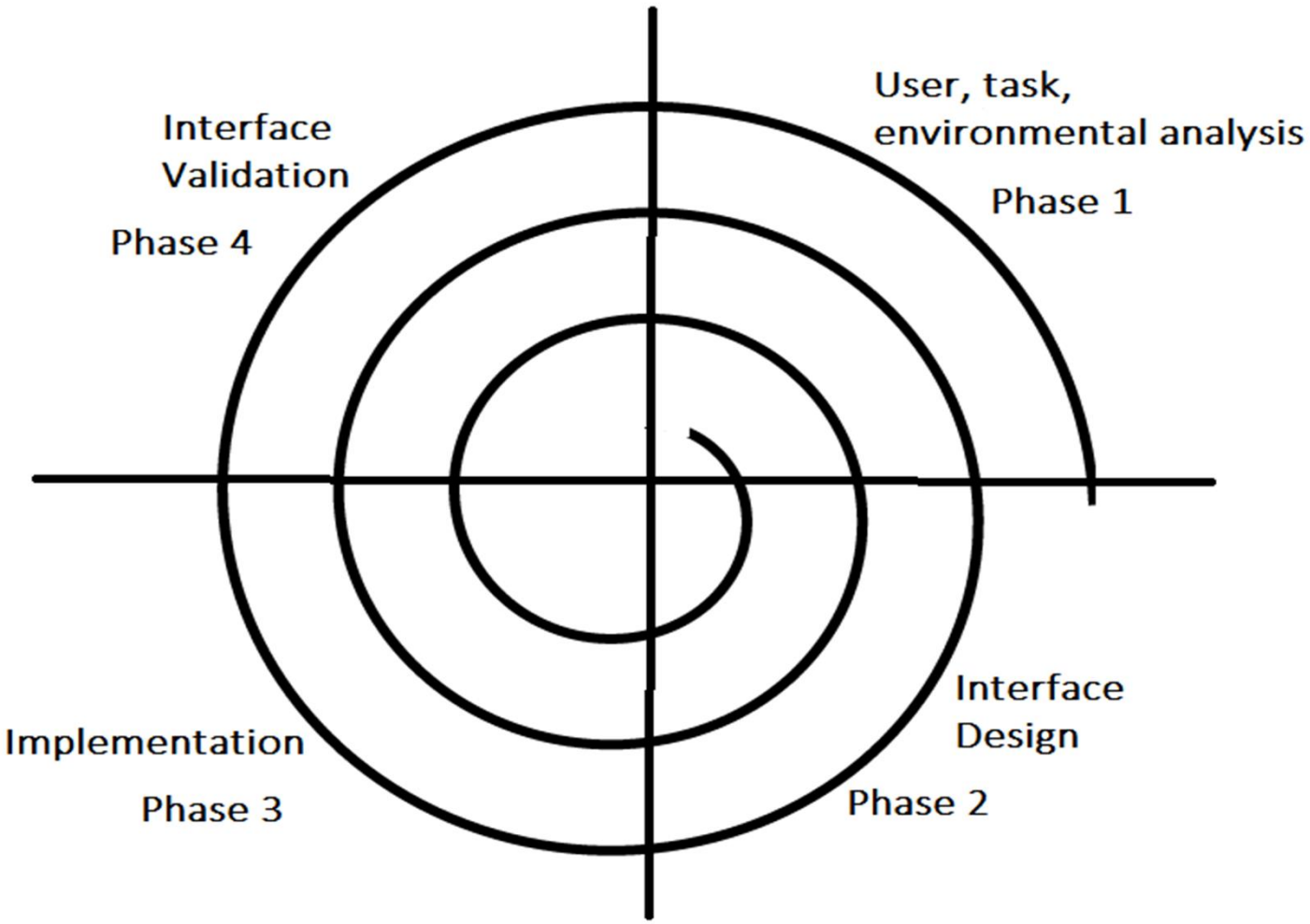
- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interface screens

There are two types of User Interface:

1. Command Line Interface: Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.

2. Graphical User Interface: Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

User Interface Design Process:



Shneiderman's Eight Golden Rules of Interface Design:

1. Strive for Consistency. ...
2. Enable Frequent Users to Use Shortcuts. ...
3. Offer Informative Feedback. ...
4. Design Dialog to Yield Closure. ...
5. Offer Simple Error Handling. ...
6. Permit Easy Reversal of Actions. ...
7. Support Internal Locus of Control. ...
8. Reduce Short-Term Memory Load.

Chapter : Eight

Topic : Software Design Complexity

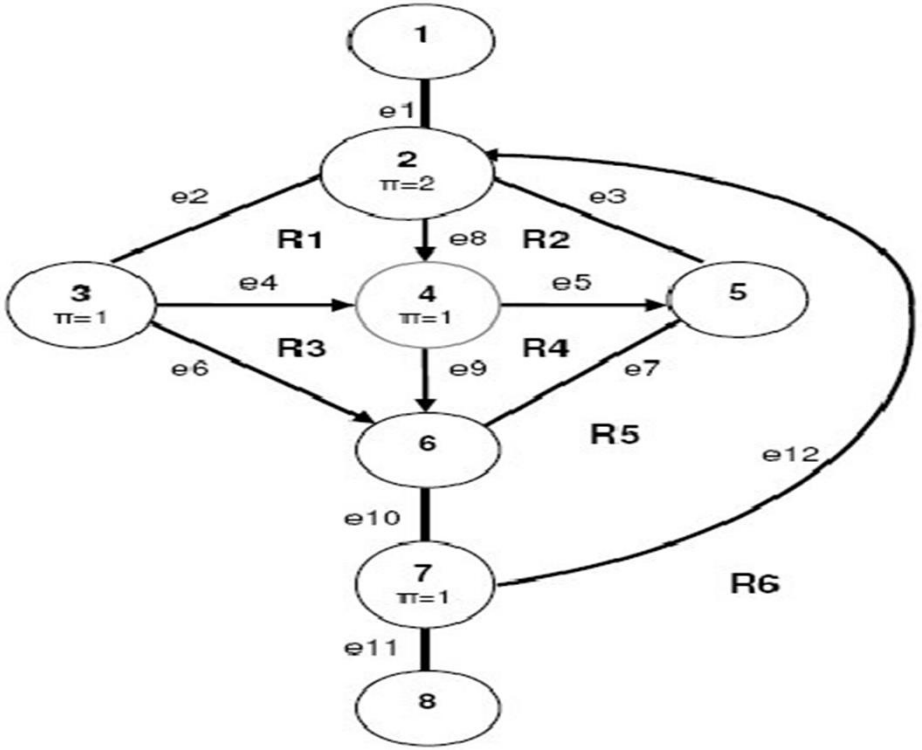
Cyclomatic Complexity:

Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Edges and Nodes Method:
v = e – n + 2
e = 12, n = 8
v = 12 – 8 + 2 = 6

Predicate Method:
v = Σπ + 1
Σπ = 5, sum of predicates
v = 5 + 1 = 6

Region (Topological) Method:
v = ΣR, sum of regions R
ΣR = 6
v = 6



Any Question ???

Thank you