Agile Development

Extreme Programming

- It is the most widely used approach for agile software development.

1. **XP Values:**

   Beck defines a set of five values:

- Communication
- Simplicity
- Feedback
- Courage
- Respect

2. The XP Process

Extreme Programming uses object oriented approach

Four framework activities:

- Planning:
  - It begins with listening
  - Creation of a set of stories
  - Value to the story
  - The XP team order the stories that will be in one of the three ways
    i. Implemented immediately
    ii. Stories with highest value implemented first
    iii. Riskiest stories will be implemented first
  - XP team computes project velocity
  - It can be used to schedule and estimate.

Design

- XP design follows KIS (Keep it simple) principle.
- Simple design X complex presentation
- CRC cards identify and organize the object oriented classes that are relevant to the current software increment.
- Spike solution-prototype
- Refactoring: it is a process of changing software system in such a way that it does not alter the external behavior of the code yet improves the internal structure.
- Design occurs both before and after code commences.

- Coding
  - After designing it undergoes unit testing.
  - Pair Programming.

- Testing
  - Unit testing
  - Regression testing
  - Integration testing
  - Validation testing
  - Acceptance testing

3. Industrial XP

Joshua Kerievsky defines this Industrial XP

It is imbued with XP's minimalist, customer-centric and test-driven spirit.

I. Readiness assessment:

- An appropriate development environment
- Proper set of stakeholders
- Continuous improvement
- The organizational will support the new values of an agile team
- Broader project community.

II. Project community

III. Project chartering

IV. Test-driven management

V. Retrospectives

VI. Continuous learning

4. The XP debate

- Requirement volatility
- Conflicting customer needs

- Requirements are expressed informally
- Lack of formal design

Other agile process models

- Adaptive software development(ASD)
- Scrum
- Dynamic systems development methods(DSDM)
- Crystal
- Feature Driven Development(FDD)
- Lean software development(LSD)
- Agile Modeling(AM)
- Agile unified process(AUP)

Scrum


24 hrs
Daily Scrum
Product Backlog
Stories
Create account
Book reservation
Print report
Login
Search
Sprint Backlog
Create Permit
Approve Permit
Cancel Permit
2 weeks
Sprint
Potentially Shippable
Product Increment

FDD


Develop An Overall Model
Build A Features List
Plan By Feature
Design By Feature
Build By Feature

THANK YOU

Quality of Design

Software Quality Guidelines and Attributes

- Implement all explicit requirements
- Accommodate all the implicit requirements
- Readable, understandable guide to code and test
- Complete picture of the software- addressing data , functional and behavioral domains.

- QUALITY GUIDELINES:
- ARCHITECTURE THAT

I. HAS BEEN CREATED USING RECOGNIZABLE ARCHITECTURAL STYLES OR PATTERNS
II. IS COMPOSED OF COMPONENTS THAT EXHIBIT GOOD DESIGN CHARACTERISTICS
III. CAN BE IMPLEMENTED IN AN EVOLUTIONARY FASHION THEREBY FACILITATING IMPLEMENTATION AND TESTING

- MODULES
- DISTINCT REPRESENTATIONS OF DATA, ARCHITECTURE, INTERFACES AND COMPONENTS.
- LEAD TO DATA STRUCTURES THAT ARE APPROPRIATE.
- INDEPENDENT FUNCTIONAL CHARACTERISTICS.
- LEAD TO INTERFACES-REDUCE THE COMPLEXITY OF CONNECTIONS.
- DERIVED USING A REPEATABLE METHOD
- REPRESENTED USING A NOTATION.

- QUALITY ATTRIBUTES
- FUNCTIONALITY
- USABILITY
- RELIABILITY
- PERFORMANCE
- SUPPORTABILITY

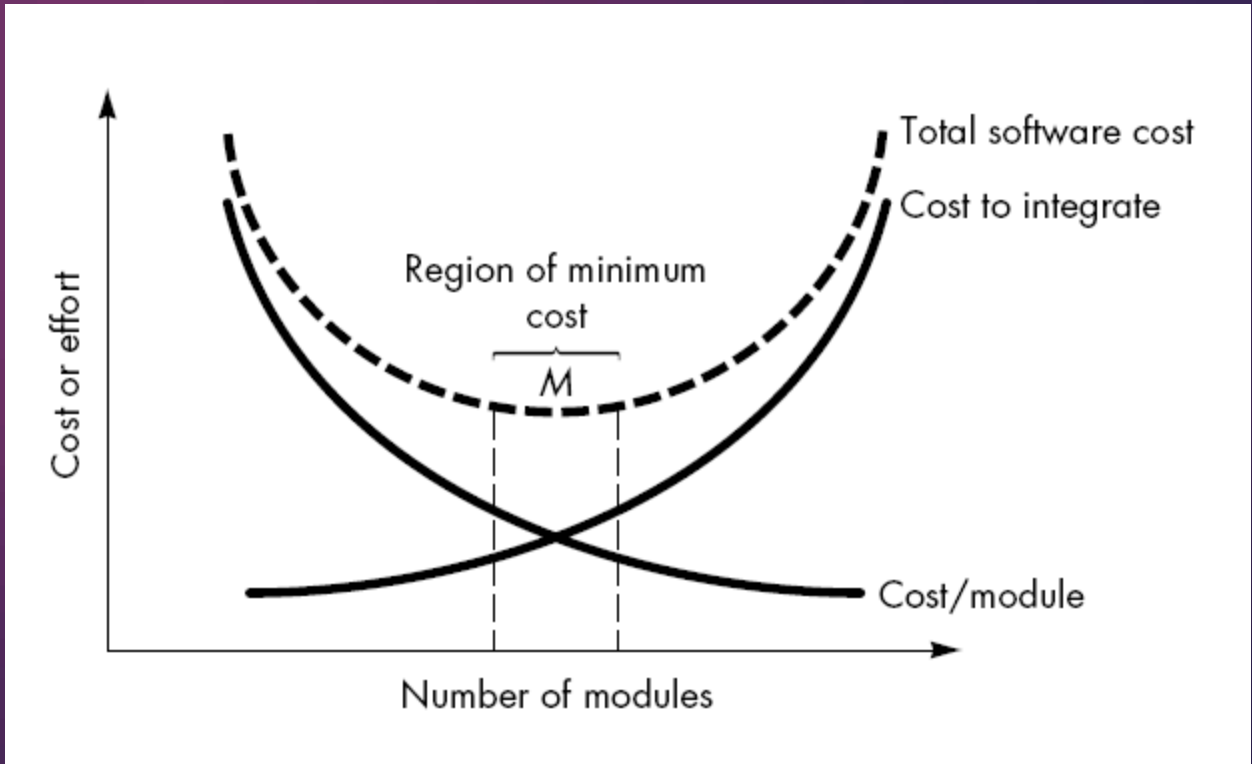
Design Concepts

- What criteria can be used to partition software into individual components?
- How is function or data structure detail separated from a conceptual representation of the software?
- What uniform criteria define the technical quality of a software design?

- Abstraction:
  - Low-level
  - High-level
  - Data

- Architecture
  - Structural properties
  - Extra-functional properties
  - Families of related systems

- PATTERNS
  - WHETHER THE PATTERN IS APPLICABLE TO THE CURRENT WORK
  - WHETHER THE PATTERN CAN BE REUSED
  - WHETHER THE PATTERN CAN BE SERVED AS GUIDE FOR DEVELOPING SIMILAR, BUT FUNCTIONALLY OR STRUCTURALLY DIFFERENT PATTERN
- SEPARATION OF CONCERNS
- MODULARITY



- Information hiding
- Functional independence: cohesion and coupling
- Refinement: elaboration
- Aspects
- Refactoring: reorganization technique

OO DESIGN CONCEPTS

- DESIGN CLASSES:
- USER INTERFACE CLASSES
- BUSINESS DOMAIN CLASSES
- PROCESS CLASSES
- PERSISTENT CLASSES
- SYSTEM CLASSES

- Characteristics:
  - Complete and sufficient
  - Primitiveness
  - High cohesion
  - Low coupling

THANK YOU

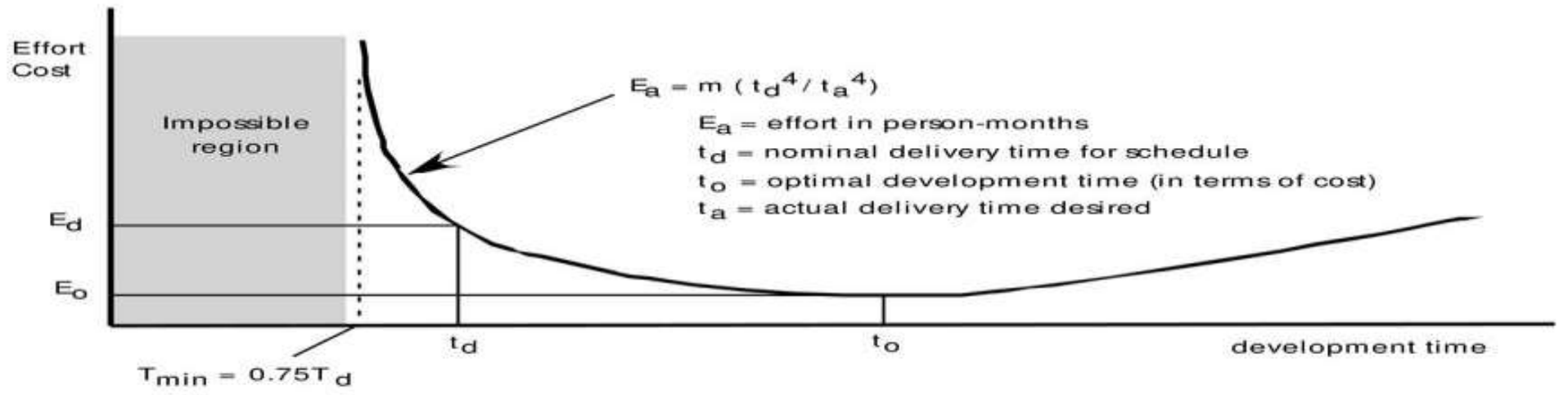
Effort
Risk
Configuration

Effort

- Software Project Scheduling is an action that distributes the estimated effort
- Early stages- Macroscopic schedule
- Two different perspectives of viewing the schedule
  - Set by Computer-based system.
  - Set by Software Engineering organization
- Effort validation:
  - Defined number of people
  - Ex: 3 engineers ⟶ three person-days and 7 concurrent tasks ⟶ 0.5 person-days

The Relationship between effort and delivery time

Relation between effort & delivery time: The PNR Curve



Effort distribution

40-20-40

- 40-front end
- 40-back end
- 20- coding

- Project planning-2 to 3
- Customer communication and requirement analysis- 10 to 25
- Software design- 20 to 25
- Code- 15 to 20
- Testing- 30 to 40

Risk Mitigation, Monitoring and Management

- Strategy-for dealing risk
- 3 issues: risk avoidance , risk monitoring and risk management and contingency plan
- Risk avoidance can be achieved by RISK MITIGATION.
- Meet the current staff to determine turnover.
- Mitigate those causes.
- Assume turnover will develop techniques to ensure continuity when people leave.
- Information is widely dispersed.
- Define work products and establish mechanisms.
- Conduct peer review.
- Assign a backup staff member.

RISK MONITORING

- Manager monitors the factors
- Project pressures
- Interpersonal relationships
- Potential problems with compensation and benefits
- Availability of jobs

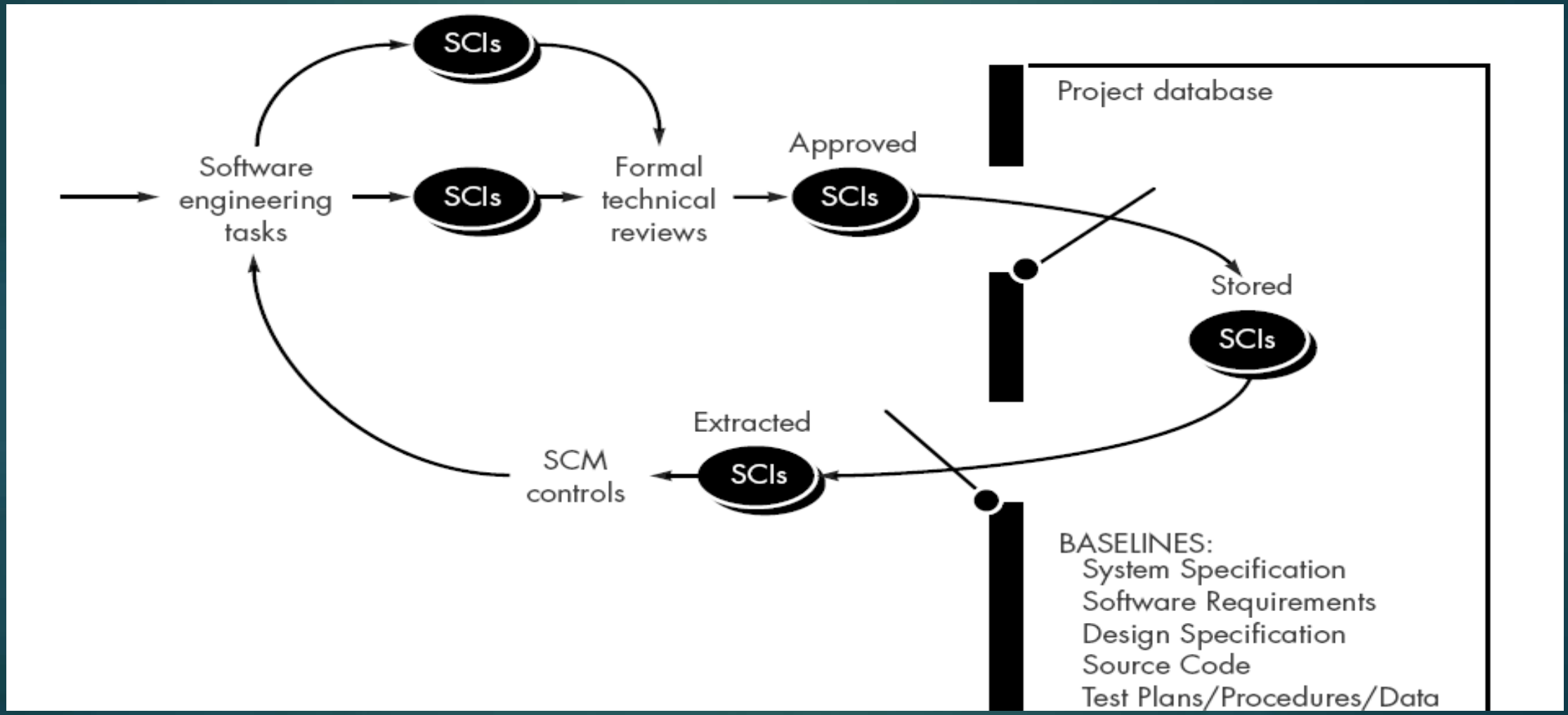
RISK MANAGEMENT AND CONTIGENCY

- Enabling new comers
- Knowledge transfer mode

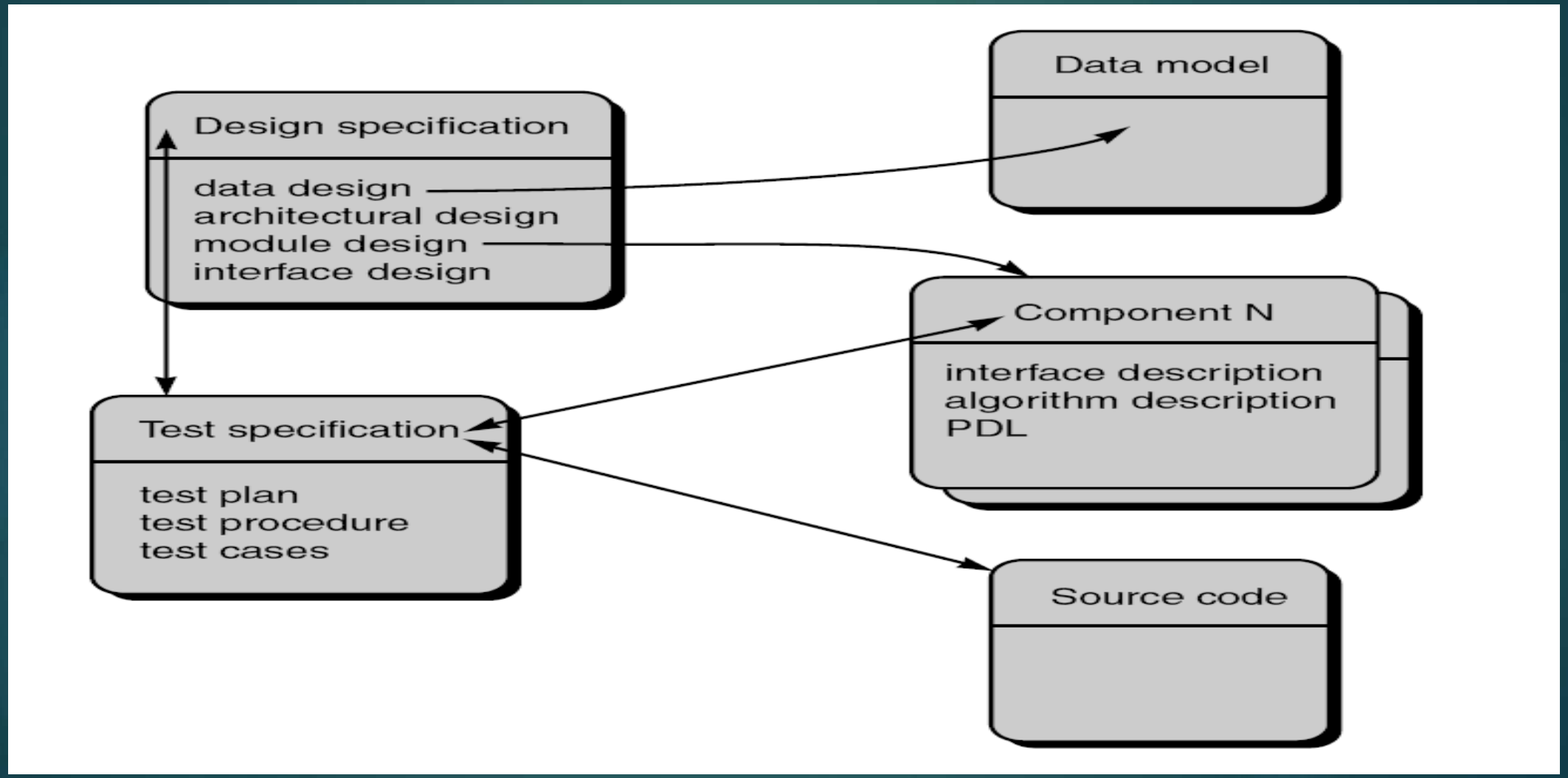
Software Configuration Management(SCM)

- The items that comprise all information produced as a part of software process are collectively called a SOFTWARE CONFIGURATION(SC).
- Work progress- SC ITEMS(SCI).
- Project manager
- Elements in SCM system
  - Component elements
  - Process elements
  - Construction elements
  - Human elements

Baselines

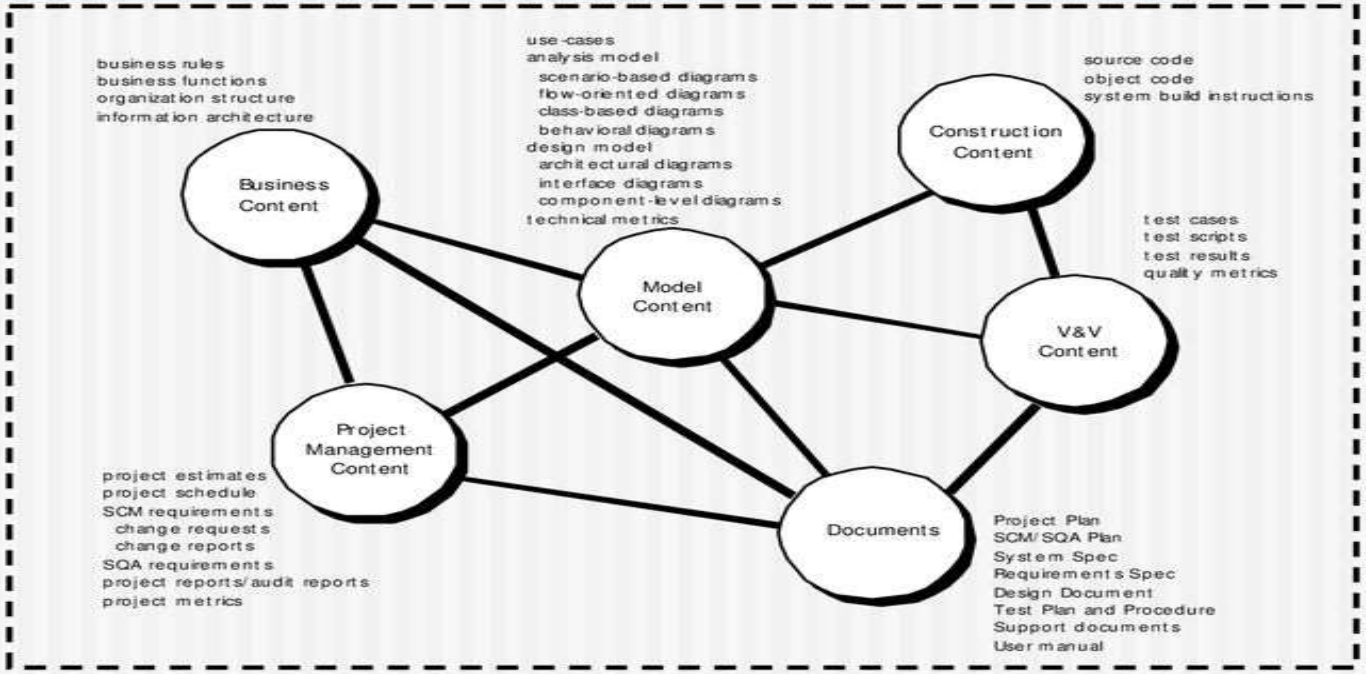

SCIs
Software engineering tasks
SCIs
Formal technical reviews
Approved
SCIs
Project database
Stored
SCIs
Extracted
SCIs
SCM controls
BASELINES:
System Specification
Software Requirements
Design Specification
Source Code
Test Plans/Procedures/Data

Software Configuration Items



SCM Repository

Repository Content

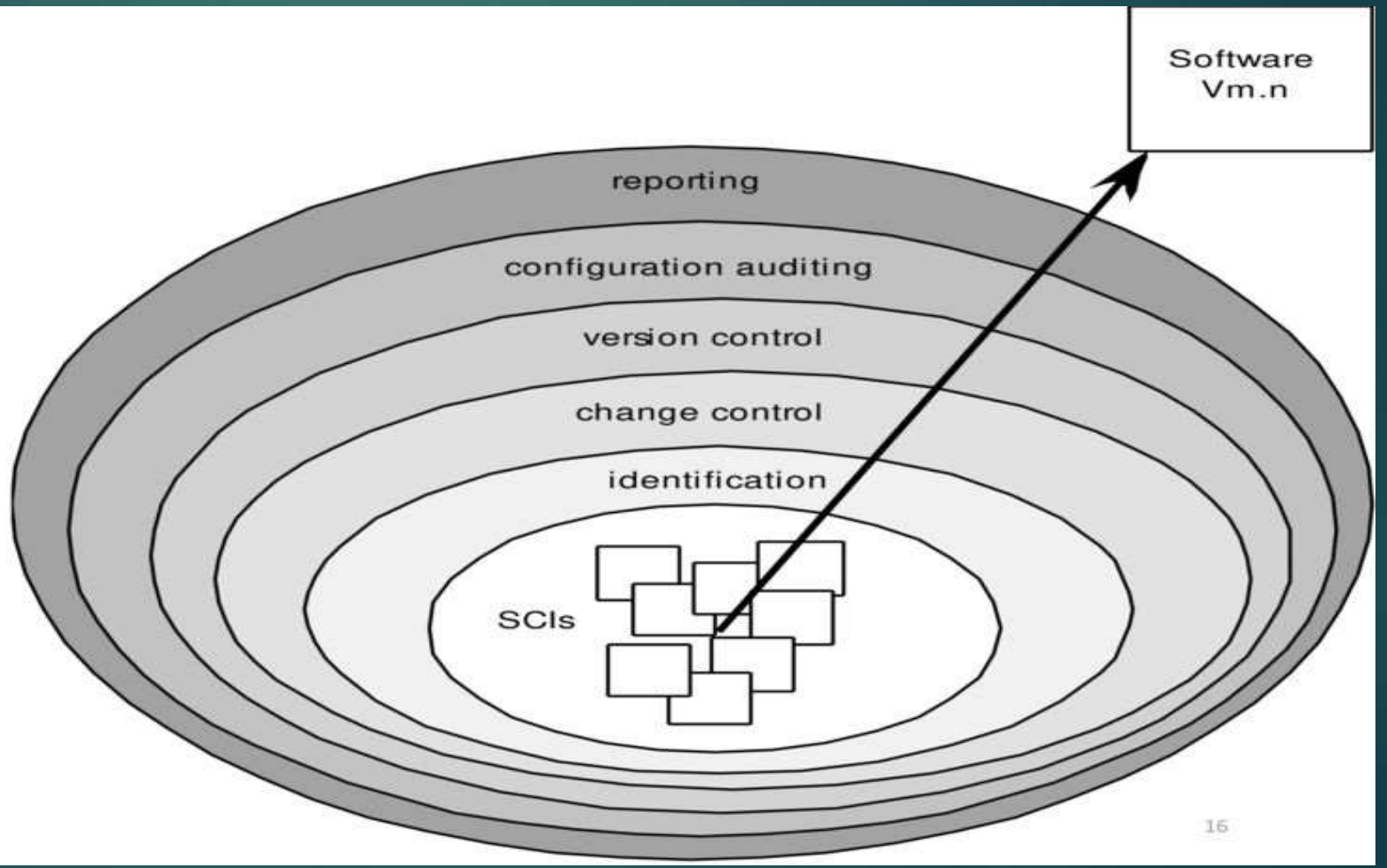


SCM Features

- Versioning
- Dependency tracking and change management
- Requirement tracing
- Configuration management
- Audit trails

SCM Process

Layers of SCM Process



- Identification Of objects in the software configuration

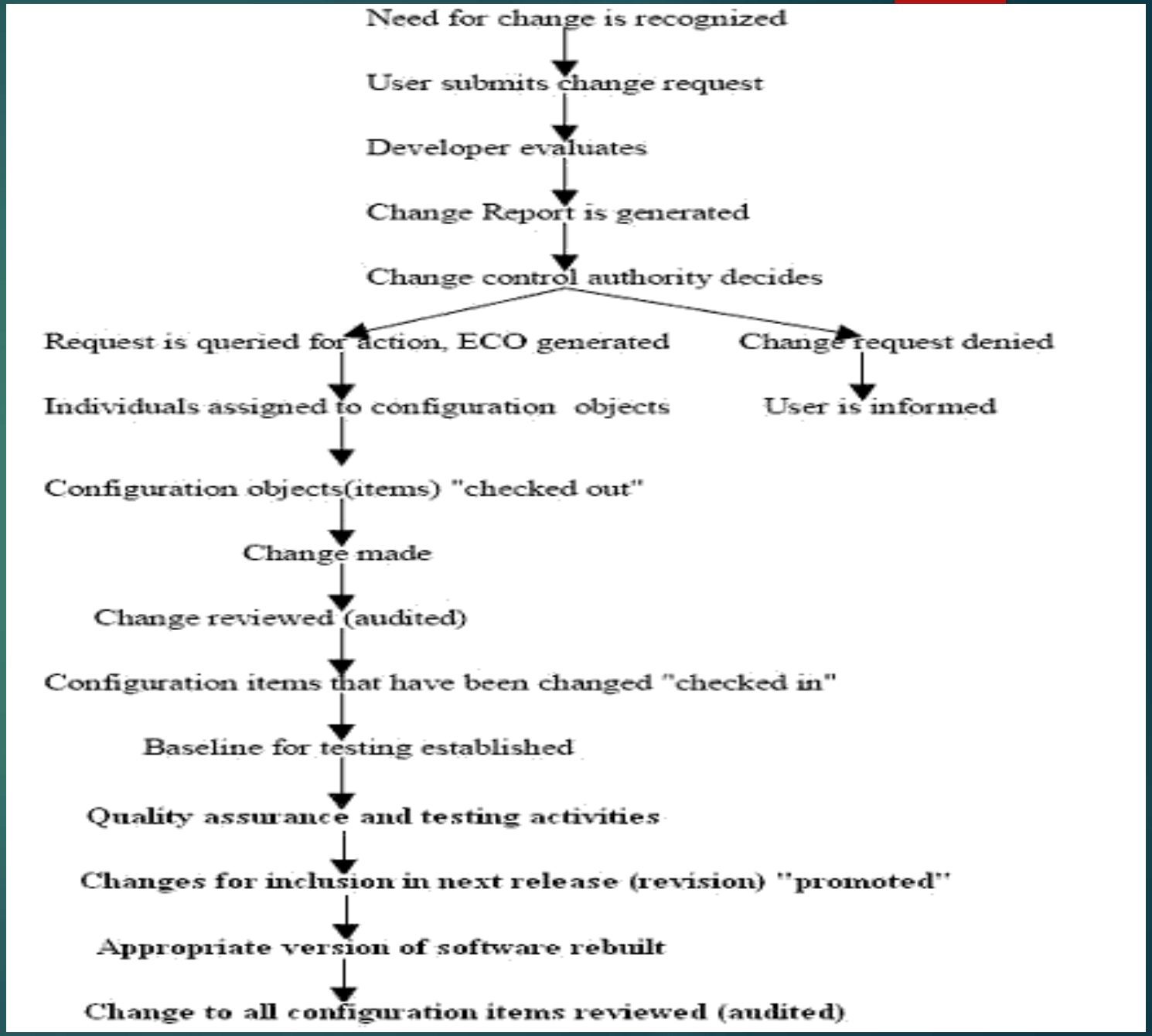
  - Base objects
  - Aggregate objects

- Version Control

  - A project database
  - Version management capability
  - Make a facility

  - System modelling approach
  - Template
  - Construction rules
  - Verification rules

Change control



Configuration Audit

- Has the change specified in the ECO been made?
- Has a technical review been conducted?
- Has the software process been followed and SE standards been properly applied?
- Has the change been “highlighted” in SCI?
- Have SCM procedures been followed?
- Have all related SCIs been properly updated?

Status report

- What happened?
- Who did it?
- When did it happened?
- What else will be affected?

THANK YOU

SOFTWARE QUALITY

What is quality?

- Quality……you know what it is , yet you don't know what it is. It is self-contradictory.
- David Garvin of the Harvard Business school – "quality is a complex and multifaceted concept" that can describe five different points of view:

➢ Transcendental view

➢ User view

➢ Manufacturer's view

➢ Product view

➢ Value-based view

- Quality of design
- Quality of conformance
- Robert Glass argues that a more intuitive relationship is in order:

User satisfaction = compliant product + good quality + delivery within budget and schedule

Software Quality

It mainly serves to emphasize three main points

- An Effective software process: analyze problem and design a solid solution
- A useful product: reliable and error-free
- Adding value: By adding value for both the producer and the user of the software product

Garvin's Quality Dimensions

- Performance quality
- Feature Quality
- Reliability
- Conformance
- Durability
- Serviceability
- Aesthetics
- Perception

McCall's Quality Factor

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability
- Maintainability
- Flexibility
- Testability
- Portability
- Reusability
- Interoperability

ISO 9126 Quality Factors

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Software Quality Dilemma

- "Good enough" Software:

➢ Software with known bugs

➢ Versions

➢ Good enough-software with high quality && Specialized functions with known bugs

➢ Major companies

➢ Small company

- The cost of Quality
  - Time
  - Money

  - The cost of quality
  - Prevention costs
  - Appraisal costs
  - Failure costs
  - Internal failure costs
  - External failure costs

- Risks
- Negligence and liability
- Quality and security
- The impact of Management Actions
  - Estimation decisions
  - Scheduling decisions
  - Risk-oriented decisions

THANK YOU

Emergence of Software Engineering as a Discipline

- Software Engineering is a “Systematic approach to the analysis, design, assessment, implementation, test, maintenance and re-engineering of software.

- As a Profession:-thinking, communicating, defining, designing, building, testing and maintaining software systems.

- One word definition of Software Engineering is MODELING
- Two word definition of Software Engineering is MODELING and OPTIMIZATION.
- Modeling is a Conversion activity.
- Optimization deals with finding the most economical conversion possible.

SE Discipline is the result of advancement in the field of technology

- Early Computer Programming
- High Level Language Programming
- Control flow base design
- Data flow oriented design
- Object Oriented design

➨ Early Computer Programming:

- Computers were slow and expensive.
- Programs were small

➨ High Level Language Programming:

- Computers became smaller, faster and cheaper(Semiconductors)
- Assembly languages to high level languages
- COBOL and FORTRAN

➨ Control flow based Design:

- Flowcharting technique
- More GOTO constructs makes the algorithm messy to understand and debug.
- Structured flowcharts-Structured constructs decision
- Structured Programming language

➨ Data flow oriented design:

- Computers became powerful and faster(VLSI)
- Flow of date through business functions is represented by DFD

➨ Object Oriented Design:

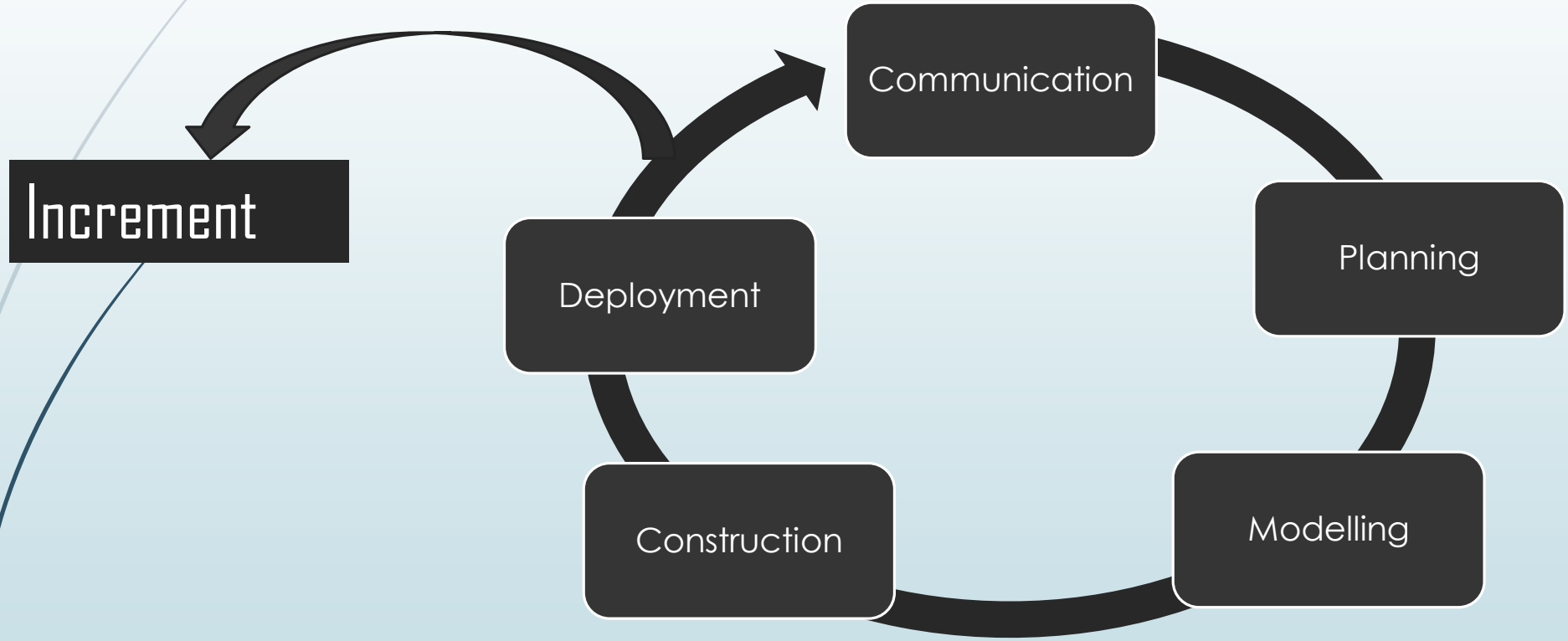
- This has revolutionized the process of Software development
- It includes some new and powerful features such as inheritance, polymorphism, abstraction and encapsulation.
- Reusability of code.

THANK YOU

THE UNIFIED PROCESS

- In early 1990s James Rambaugh, Grady Booch and Ivar Jacobson began working on unified-process.
- It recognizes the importance of customer communication and streamlined methods for describing the customer view of a system.
- It suggests a process flow that is iterative, incremental, providing the evolutionary feel that is essential in modern software development.

Phases of the Traditional Process Model:



Phases of the Unified Process:

- Inception
  - Communication
  - Planning
- Elaboration
  - Communication
  - Modelling
- Construction
- Transition
  - Construction
  - Deployment
- Production

THANK YOU

WHITE BOX TESTING

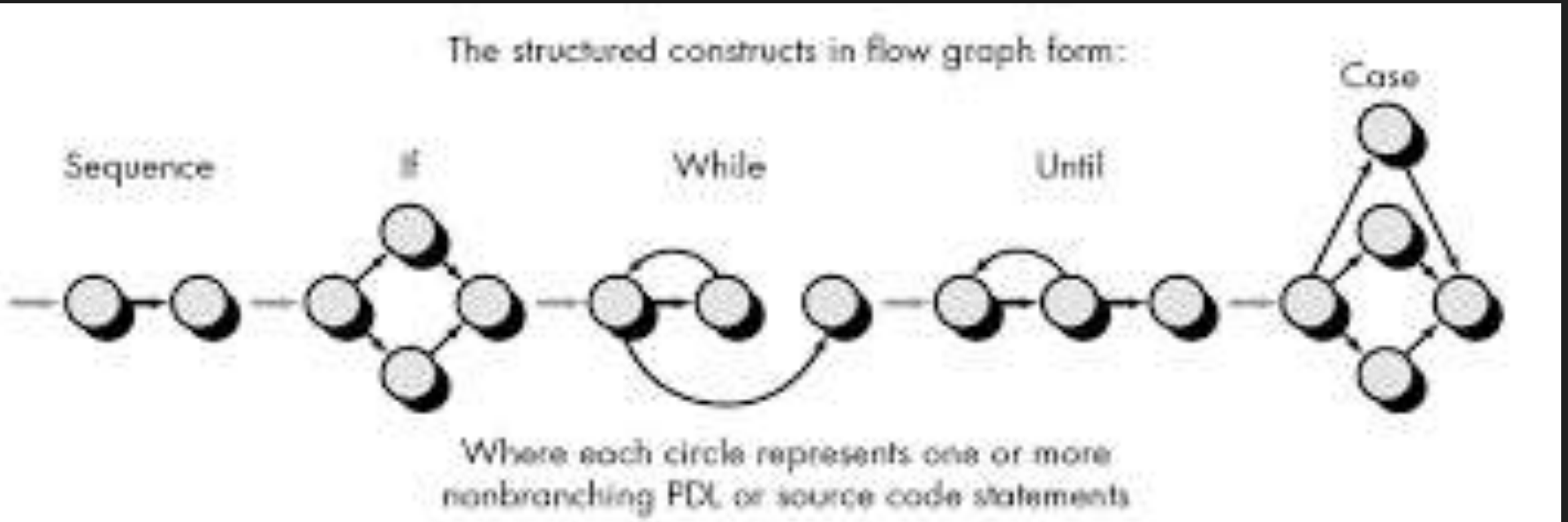
- Glass box testing
- Test-case design philosophy---Component-level design

- Can drive test cases that

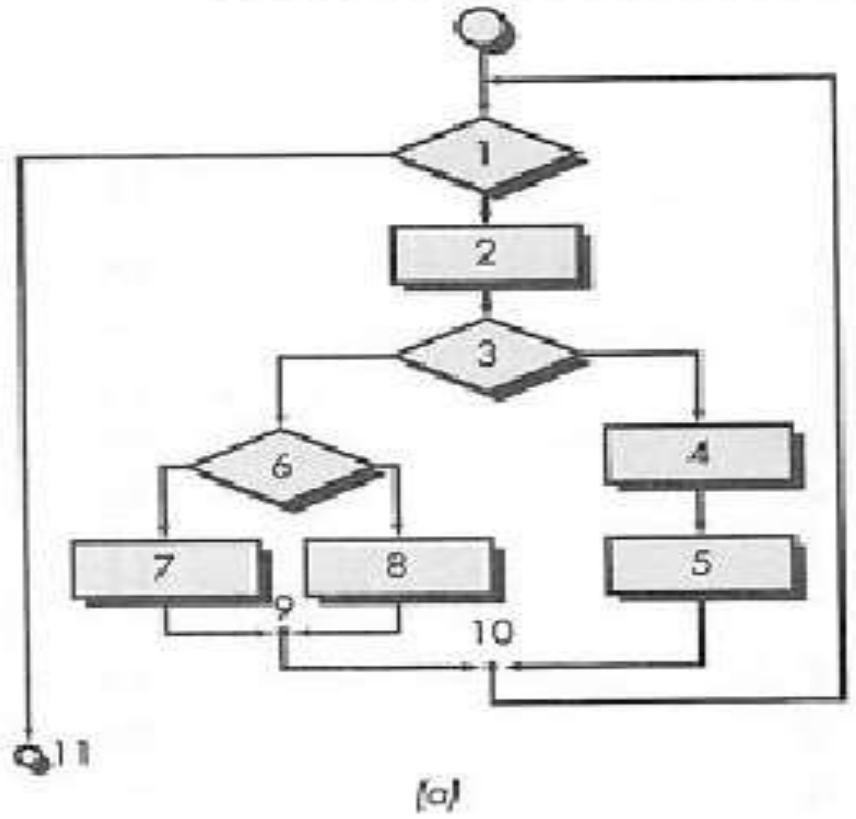
1. Guarantee that all the independent paths within a module have been exercised at lease once
2. Exercise all logical decisions on their true and false slides
3. Execute all loops at their boundaries & within their operational bounds
4. Exercise internal data structures to ensure their validity

- BASIC PATH TESTING is a white box testing technique first proposed by Tom McCabe

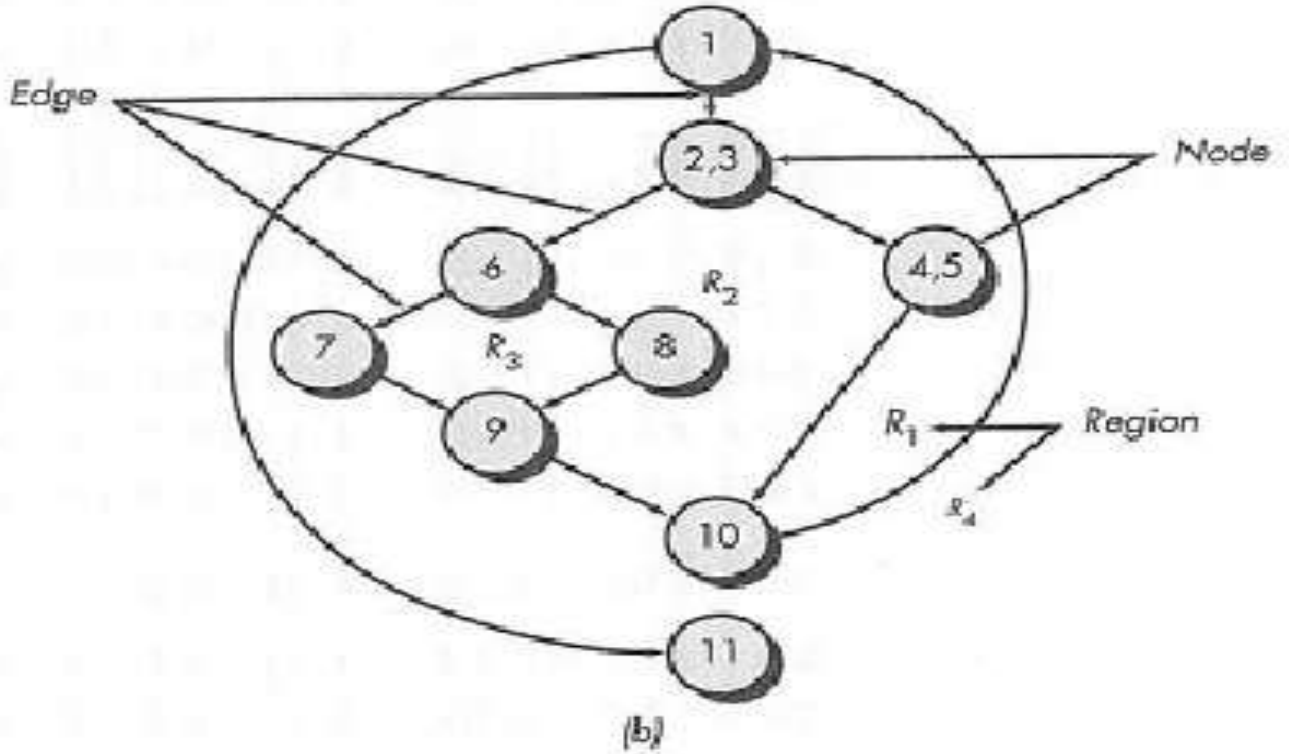
Flow graph Notation



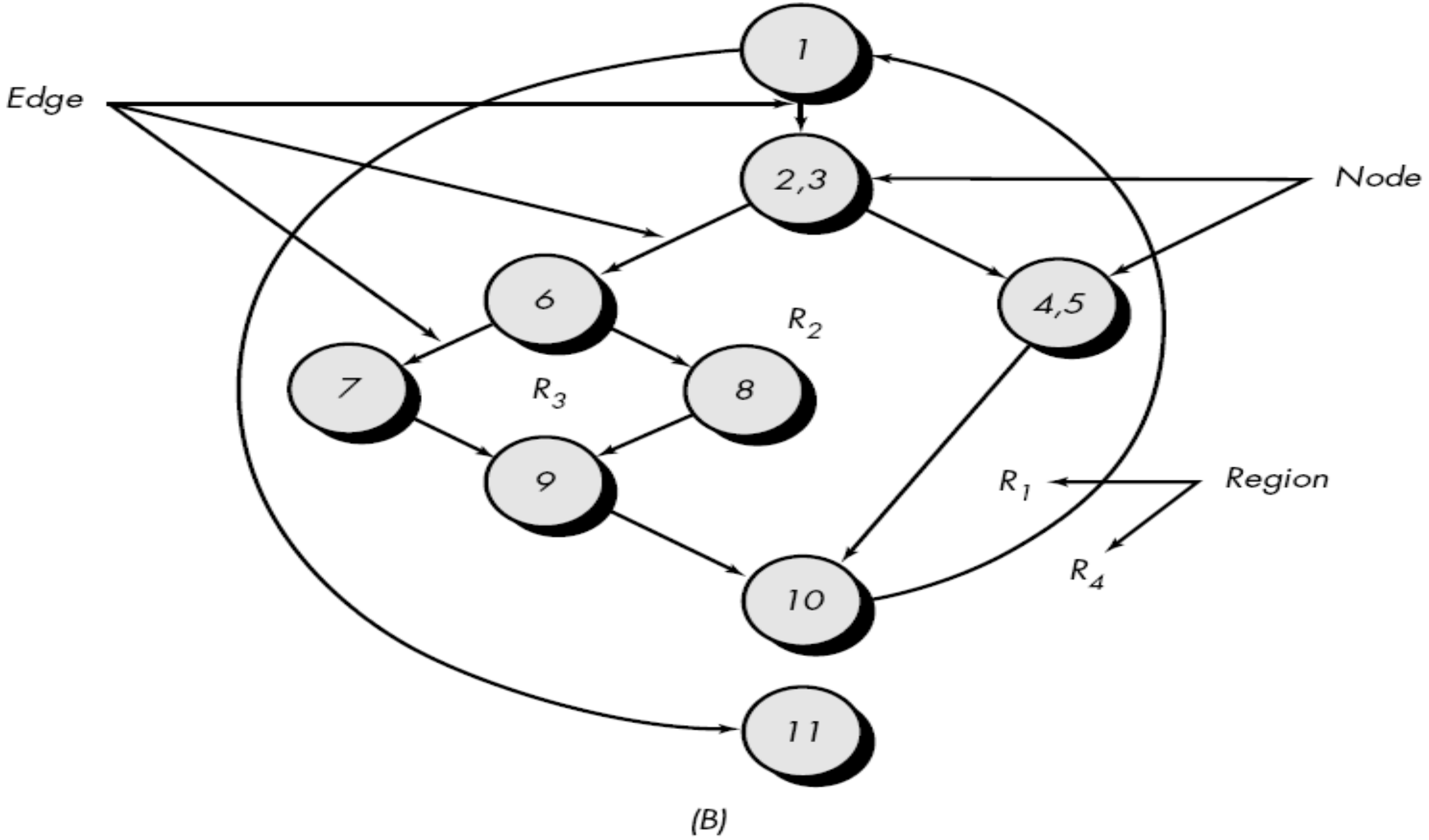
Flow graph notation



(a)

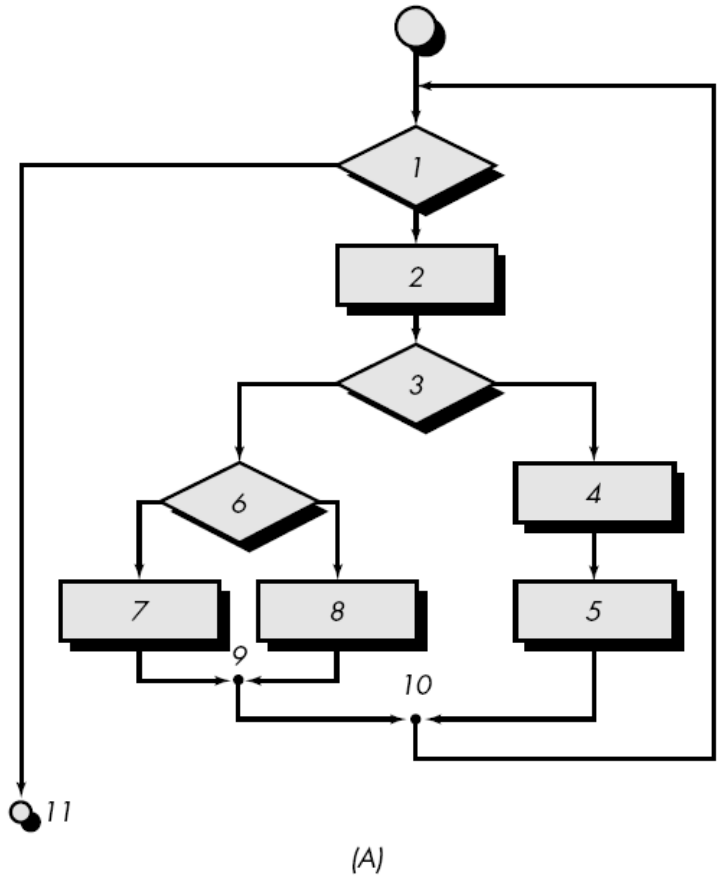


(b)



(B)

Independent Program Paths



Path 1: 1-11
Path 2: 1-2-3-4-5-10-1-11
Path 3: 1-2-3-6-8-9-10-1-11
Path 4: 1-2-3-6-7-9-10-1-11

Paths 1 through 4 constitute a basis set for the flow graph

If you can design tests to force execution of these paths, every statement in the program will have been guaranteed to be executed at least once and every condition will have been executed on its true and false sides

- Cyclomatic complexity: Software metric that provides a quantitative measure of the logical complexity of a program.
- Cyclomatic complexity defines number of independent paths and provide an upper bound for the number of tests.

- No.of ways to compute CC:

- The no.of regions of the flow graph
- CC V(G) for a flow graph G is defined as:
  V(G) = E-N+2
- V(G) = P+1

- The flow graph has four regions.
- V(G) = 11 edges-9 nodes+2=4
- V(G)= 3 predicate nodes+1=4

Deriving testcases

- Using the design or code as foundation, draw a corresponding flow graph
- Determine the cyclomatic complexity of the resultant flow graph
- Determine a basis set of linearly independent paths
- Prepare test cases that will force execution of each path in the basis

Control structure testing

- Condition testing:
- It exercises that logical conditions contained in a program module

- A simple condition is a Boolean variable or a relational expression
- E1<relational-operator> E2

- A compound condition is composed of two or more simple conditions, Boolean operators and parenthesis

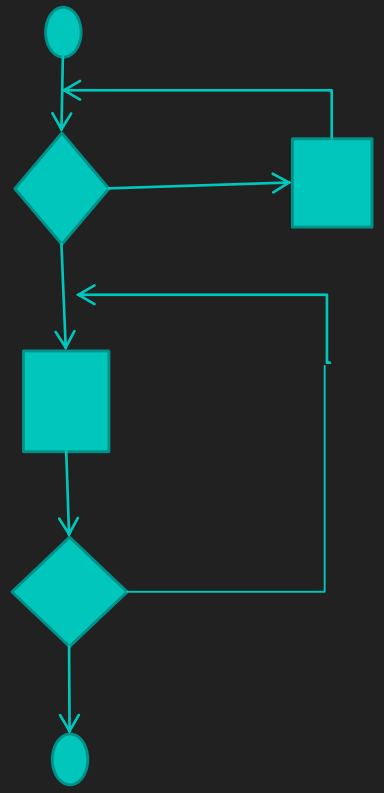
- Types of errors in condition include Boolean operator errors, Boolean variable errors, Boolean parenthesis errors, relational operator errors and arithmetic expression errors.

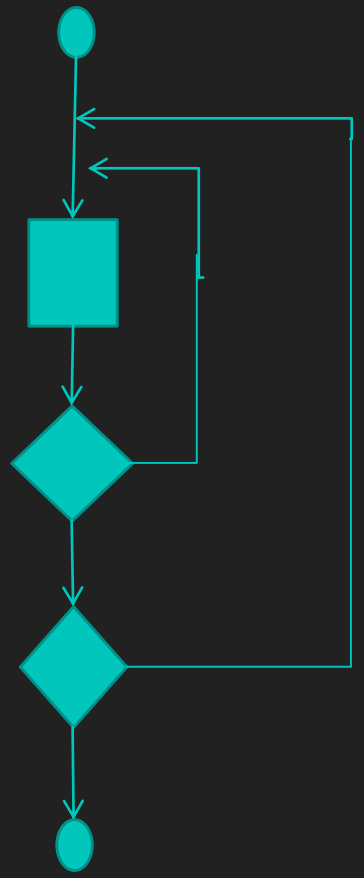
Data flow testing

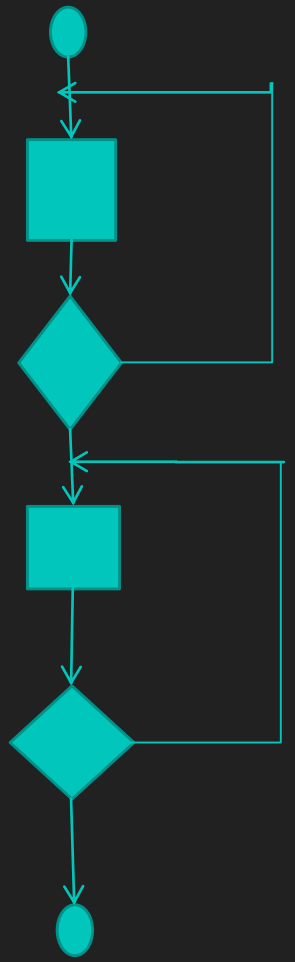
- DEF(S) = {X | statement S contains a definition of X}
- USE9S) = { X | statement S contains a use of X}

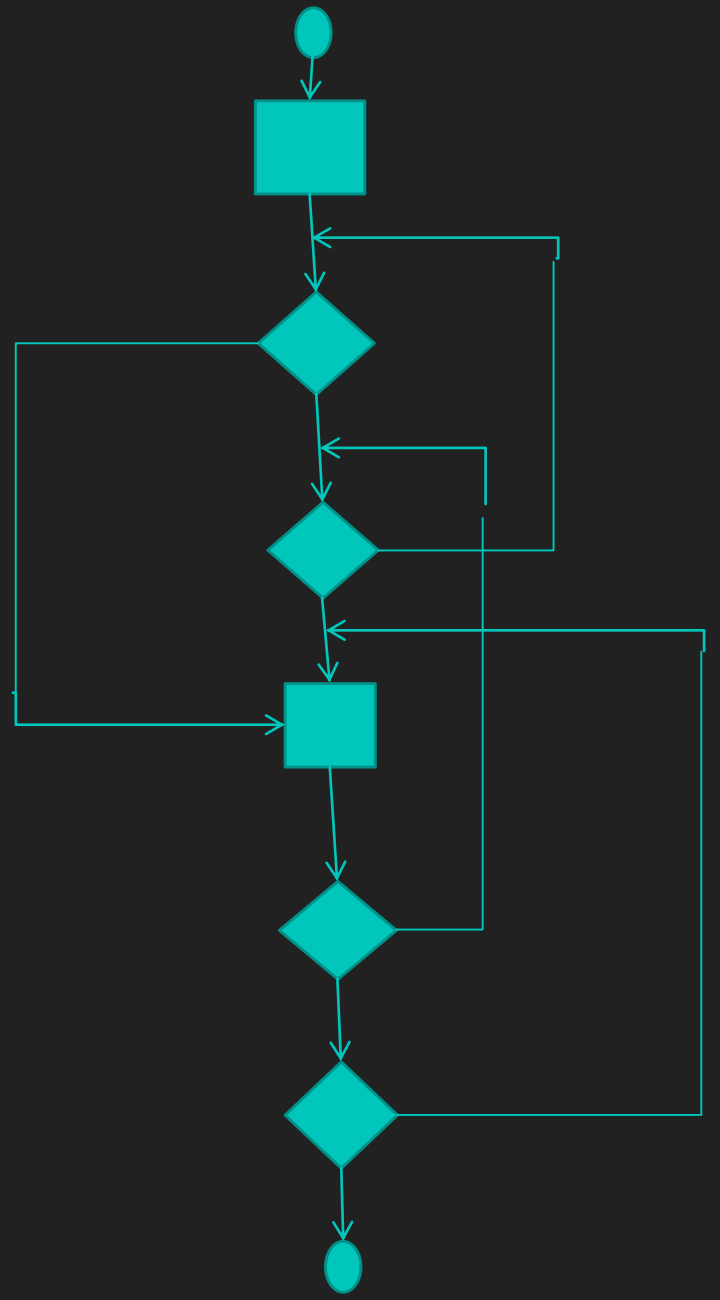
Loop Testing

- Simple loops
- Nested loops
- Concatenated loops
- Unstructured loops









Simple loop

- Skip the loop entirely
- Only one pass through the loop
- Two passes through the loop
- m passes through the loop where m<n
- n-1 , n , n+1 passes through the loop

Nested loop

- Start at the innermost loop. Set all other loops to minimum values
- Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter values. Add other tests for out-of-range or excluded values.
- Work outward, conducting tests for next loop, but keeping all other outer loops at minimum values and other nested loops to typical values
- Continue until all loops have been tested

Concatenated loops and Unstructured loops

- Concatenated loops can be tested using the approach defined for simple loops, if each of the loop is independent of the other.
- If the loops are not independent then the approach applied to nested loops is recommended.

- Whenever possible this class of loops should be redesigned to reflect the use of the structured programming constructs.

Black-box Testing

- Black box testing also called as Behavioral testing

- Enables to derive sets of input conditions that will fully exercise all functional requirements

- It is an alternative to white-box testing

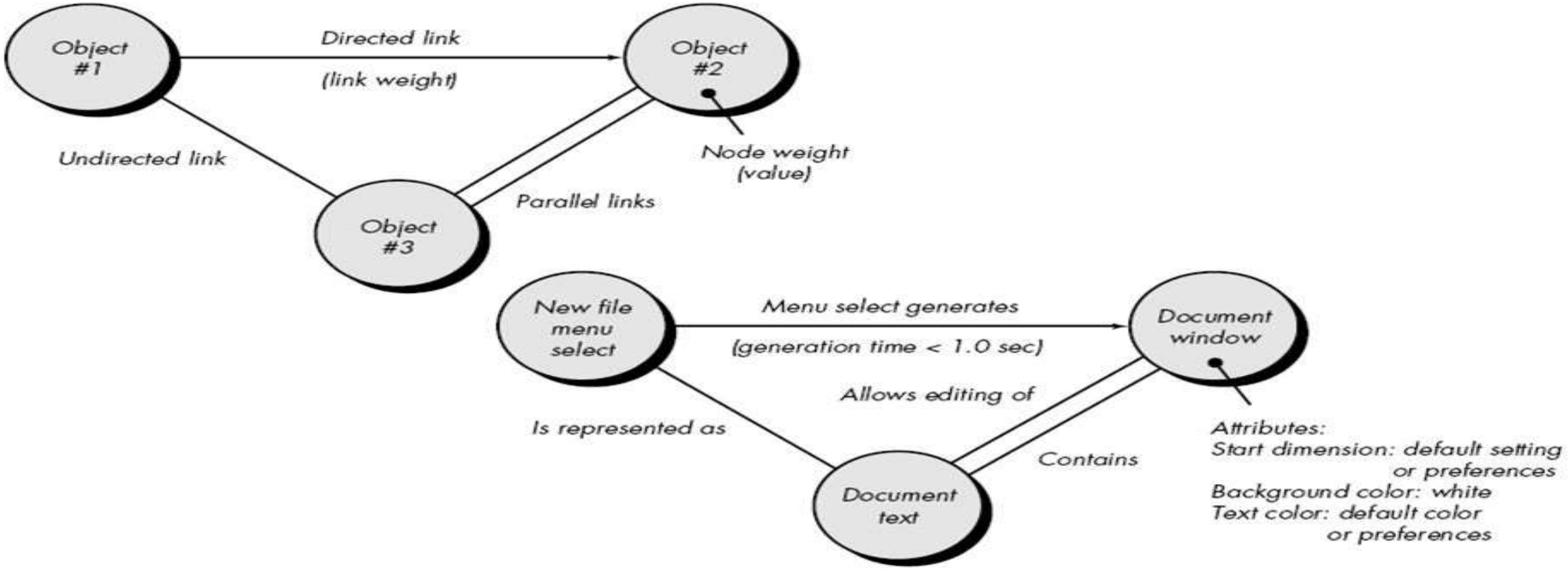
- BB testing attempts to find out errors in following categories:
  - Incorrect or missing functions
  - Interface errors
  - Errors in data structures or external database
  - Behavior or performance errors
  - Initialization and termination errors

- Performed during the stages of testing

- Tests are designed to answer the following questions
  - How is functional validity tested?
  - How are system behavior and performance tested?
  - What classes of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - How are the boundaries of data class isolated?
  - What data rates and data volume can the system tolerate?
  - What effect will specific combinations of data have on system operation

Graph-based testing methods

Graph-Based Testing



- Transaction flow modeling
- Finite state modeling
- Data flow modeling
- Timing modeling

Equivalence partitioning

- Divides the input domain of the program into classes of data from which test cases can be derived

- Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition

- Guidelines:

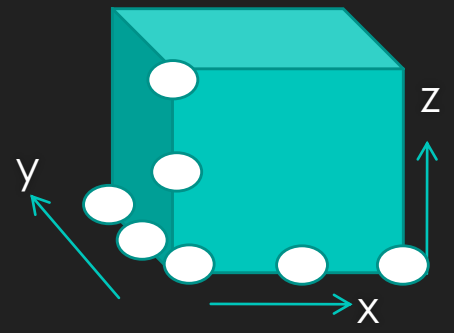
- If an input condition specifies a range , one valid and two invalid equivalence classes are defined
- If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
- If an input condition specifies a member of set, one valid and one invalid equivalence classes are defined.
- If an input condition is Boolean , one valid and one invalid classes are defined.

Boundary value analysis

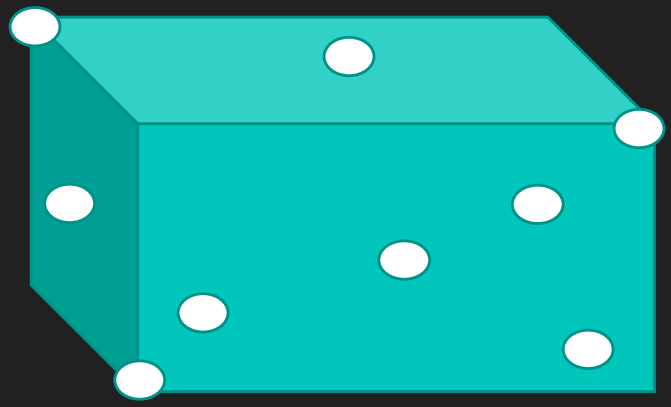
- As errors occurring at the boundaries are greater than at the center BVA is developed
- It is complementary to Equivalence partitioning
- Guidelines:
  - If an input condition specifies a range bounded by values a and b, test cases should be designed with value a and b and just above and just below a and b.
  - Specifies no.of values → exercise the minimum and maximum numbers
  - Apply guidelines 1 and 2 to output conditions
  - If internal program data structure have prescribed boundaries be certain to design a test case to exercise the data structure at its boundaries

Orthogonal array testing

- Orthogonal array testing can be applied to problems in which the input domain is relatively small.
- Exhaustively test the input domain
- Particularly in finding region faults



"One input at a time"



L9 orthogonal array

- "Balanced property"- "test cases are dispersed uniformly throughout the test domain"

Test case	Test parameters			
	P1	P2	P3	P4
	1	1	1	1
	2	1	1	1
	3	1	1	1
	1	2	1	1
	1	3	1	1
	1	1	2	1
	1	1	3	1
	1	1	1	2
	1	1	1	3

Test case	Test parameters			
	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

- **Detect and isolate all single mode faults**: A single mode fault is a consistent problem with any level of any single parameter

- **Detect all double mode faults**: A consistent problem when specific levels of two parameters occur together

- **Multimode faults**

THANK YOU

SYSTEM TESTING

- A classic system testing problem is “finger pointing”.
- Design error handling paths
- Conduct series of tests
- Record the result of test-evidence
- Participate in planning and design of the system

Recovery testing

- Recovery from faults and resume processing
- Fault tolerant
- RT is a system test that forces software to fail in a variety of ways and verifies the recovery
- Automatic- reinitialization, checkpoint mechanisms, data recovery and restart are evaluated for correctness
- Human intervention-MTTR

Security testing

- Improper or Illegal penetration
- ST- verify that protection mechanisms built into a system will, in fact, protect it from improper penetration
- The tester plays role of hacker
- Good security testing will ultimately penetrate a system.
- Penetration cost is more than value of the information

Stress Testing

- Stress testing- demands resources in abnormal quantity, frequency and volume
- Special tests may be designed that generate ten interrupts per second, when 1 or 2 is the average rate
- Input data rates may be increased by magnitude
- Require maximum memory
- Thrashing in virtual operating system
- Excessive hunting for disk-resident data
- Sensitivity testing-small range of data

Performance testing

- PT- the run time performance of software within the context of integrated system
- Often coupled with stress testing and usually require both hardware and software instrumentation
- Resource utilization
- Execution intervals , log events and sample machine states on a regular basis

Deployment testing

- Deployment testing-Configuration testing

- Examines all installation procedures and specialized installation software and documentation

Non Functional requirements testing

- Primary purpose- test the reading speed of the system as per non-functional requirements.
- Functional testing checks the correctness of internal functions while Non-Functional testing checks the ability to work in an external environment.
- Non-functional testing gives detailed knowledge of product behavior and used technologies


Performance
Efficiency
Loading Capacity
Non-Functional Testing Parameters
Reliability
Security Testing
Accountability
Portability

THANK YOU