`

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

SYLLABUS
DEPARTMENT OF CS, CA & IT

SUBJECT NAME: SOFTWARE ENGINEERING **SUBJECT CODE: 18CAU401**

SEMESTER: IV **CLASS: II BCA**

Scope

This course sharpens the theoretical, technical, and practical knowledge of software requirements, analysis, design, implementation, verification and validation, and documentation skills of the students

Objectives

- Apply their knowledge of mathematics, sciences, and computer science to the modeling, analysis, and measurement of software artifacts.
- Analyze, specify and document software requirements for a software system.
- Implement a given software design using sound development practices.
- Verify, validate, assess and assure the quality of software artifacts.
- Design, select and apply the most appropriate software engineering process for a given project, plan for a software project, identify its scope and risks, and estimate its cost and time.

Unit-I

Introduction: The Evolving Role of Software, Software Characteristics, Changing Nature of Software, Software Engineering as a Layered Technology, Software Process Framework, Framework and Umbrella Activities, Process Models, Capability Maturity Model Integration (CMMI).

Unit-II

Requirement Analysis; Initiating Requirement EngineeringProcess- Requirement Analysis and Modeling Techniques- FlowOriented Modeling- Need for SRS- Characteristics and Components of SRS- Software Project Management: Estimation in Project Planning Process, Project Scheduling.

Unit-III

Risk Management: Software Risks, Risk Identification Risk Projection and Risk Refinement, RMMM plan, **Quality Management-** Quality Concepts, Software Quality Assurance, Software Reviews, Metrics for Process and Projects

`

Unit-IV

Design Engineering-Design Concepts, Architectural Design Elements, Software Architecture, Data Design at the Architectural Level and Component Level, Mapping of Data Flow into Software Architecture, Modeling Component Level Design

Unit-V

Testing Strategies & Tactics: Software Testing Fundamentals, Strategic Approach to Software Testing, Test Strategies for Conventional Software, Validation Testing, System testing Black-Box Testing, White-Box Testing and their type, Basis Path Testing

Suggested Readings

1. Aggarwal K.K., Singh,Y., (2008). *Software Engineering*, (2nd ed.), New Age International Publishers.
2. Bell,D., (2005). *Software Engineering for Students*, (4th ed.), Addison-Wesley.
3. Jalote,P., (2008). *An Integrated Approach to Software Engineering* (2nd ed.), New Age International Publishers.
4. Mall,R.,(2004). *Fundamentals of Software Engineering*, (2nd ed.), Prentice-Hall of India.
5. Pressman, R.S.,(2009). *Software Engineering: A Practitioner's Approach,* (7th Edition), McGraw-Hill.
6. Sommerville, I.,(2006), *Software Engineering*, (8th ed.), Addison Wesley.

Websites

1. http://en.wikipedia.org/wiki/Software_engineering
2. http://www.onesmartclick.com/engineering/software-engineering.html
3. http://www.CSU.gatech.edu/classes/AY2000/cs3802_fall/

Question Paper Pattern:

CIA	**Max.Marks : 50**
Part A	Objective type questions : 20 x 1 = 20 Marks
Part B	Answer all the questions Either/Or : 3 x 2 = 6 Marks
Part C	Answer all the questions : 3 x 8 = 24 Marks

ESE	**Max.Marks : 60**
Part A	Objective type questions : 20 x 1 = 20 Marks
Part B	Answer all the questions Either/Or : 5 x 6 = 30 Marks
Part C	Answer all the questions : 1 x 10 = 10 Marks

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established under Section 3 of UGC Act 1956)
Pollachi Main Road, Eacharani Post, Coimbatore-641 021

DEPARTMENT OF COMPUTER APPLICATIONS
Semester – IV
SOFTWARE ENGINEERING(18CAU401)
LESSON PLAN
UNIT-I

S.NO	DURATION	TOPICS TO BE COVERED	SUPPORTED MATERIALS
1.	1	Introduction: The Evolving Role of Software,Software Characteristics	T1:34-40
2.	1	Changing The Nature of Software	W1
3.	1	Software engineering as a Layered Technology	T1:53,W1
4.	1	Software Process Framework	T1:54-58
5.	1	Framework & Umberlla Activities	W1
6.	1	Process Model	W1
7.	1	Capability Maturity Model Integration(CMMI)	T1:59-67
8.	1	RECAPITULATION OF IMPORTANT QUESTIONS	
TOTAL HOURS: 8 HOURS			

TEXTBOOK :

T1: Pressman, R.S.,(2009). *Software Engineering: A Practitioner's Approach,* (7th Edition), McGraw-Hill.

REFERENCE BOOKS:

R1: Aggarwal K.K., Singh,Y., (2008). *Software Engineering,* (2nd ed.), New Age International Publishers.

WEBSITE:

W1: http://en.wikipedia.org/wiki/Software_engineering

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established under Section 3 of UGC Act 1956)
Pollachi Main Road, Eacharani Post, Coimbatore-641 021

DEPARTMENT OF COMPUTER APPLICATIONS
Semester – IV
SOFTWARE ENGINEERING(18CAU401)
LESSON PLAN
UNIT-II

S.NO	DURATION	TOPICS TO BE COVERED	SUPPORTED MATERIALS
1.	1	Requirement Analysis:Intiating Requirement Engineering Process	T1:208
2.	1	Requirrement Analysis & Modeling Techniques	T1:211-215,W1
3.	1	FlowOriented Modelling	T1:226-235,W1
4.	1	Need for SRS	T1:254-258
5.	1	Characteristics & component of SRS	W1
6.	1	Software Project management :Estimation In Project Planning Process	T1:259-267,W1
7.	1	Project Scheduling	T1:705-712,W1
8.	1	RECAPITULATION OF IMPORTANT QUESTIONS	
TOTAL HOURS: 8 HOURS			

TEXTBOOK :

T1: Pressman, R.S.,(2009). *Software Engineering: A Practitioner's Approach*, (7th Edition), McGraw-Hill.

REFERENCE BOOKS:

R1: Aggarwal K.K., Singh,Y., (2008). *Software Engineering*, (2nd ed.), New Age International Publishers.

WEBSITE:

W1: http://en.wikipedia.org/wiki/Software_engineering

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established under Section 3 of UGC Act 1956)
Pollachi Main Road, Eacharani Post, Coimbatore-641 021

DEPARTMENT OF COMPUTER APPLICATIONS
Semester – IV
SOFTWARE ENGINEERING(18CAU401)
LESSON PLAN
UNIT-III

S.NO	DURATION	TOPICS TO BE COVERED	SUPPORTED MATERIALS
1.	1	Risk Management : Software Risk	T1:728,W1
2.	1	Risk Identification , Risk Projection & Risk Refinement	T1:729-737,W1
3.	1	RMMM Plan	T1:740,W1
4.	1	Quality Management:Quality Concept	T1:745-747,W1 R1:555-560
5.	1	Software Quality Assurance , Software Review	T1:748-754,W1
6.	1	Metrics For Process & Projects	W1 R1:97-100
7.	1	RECAPITULATION OF IMPORTANT QUESTIONS	
TOTAL HOURS: 7 HOURS			

TEXTBOOK :

T1: Pressman, R.S.,(2009). *Software Engineering: A Practitioner's Approach,* (7th Edition), McGraw-Hill.

REFERENCE BOOKS:

R1: Aggarwal K.K., Singh,Y., (2008). *Software Engineering,* (2nd ed.), New Age International Publishers.

WEBSITE:

W1: http://en.wikipedia.org/wiki/Software_engineering

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established under Section 3 of UGC Act 1956)
Pollachi Main Road, Eacharani Post, Coimbatore-641 021

DEPARTMENT OF COMPUTER APPLICATIONS
Semester – IV
SOFTWARE ENGINEERING(18CAU401)
LESSON PLAN
UNIT-IV

S.NO	DURATION	TOPICS TO BE COVERED	SUPPORTED MATERIALS
1.	1	Design Engineering : Design Concept	T1:261-264
2.	1	Architectural Design Elements	T1:275,276
3.	1	Software Architecture	T1:287-289
4.	1	Data Design At The Architectural Level & Component Level	T1:289,290, 303 R1:300-310
5.	1	Mapping Of DataFlow Into Software Architecture	T1:307-320 R1:340
6.	1	Modelling Component Level Design	T1:324-346
7.	1	RECAPITULATION OF IMPORTANT QUESTIONS	
TOTAL HOURS: 7 HOURS			

TEXTBOOK :

T1: Pressman, R.S.,(2009). *Software Engineering: A Practitioner's Approach,* (7th Edition), McGraw-Hill.

REFERENCE BOOKS:

R1: Aggarwal K.K., Singh,Y., (2008). *Software Engineering*, (2nd ed.), New Age International Publishers.

WEBSITE:

W1: http://en.wikipedia.org/wiki/Software_engineering

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established under Section 3 of UGC Act 1956)
Pollachi Main Road, Eacharani Post, Coimbatore-641 021

DEPARTMENT OF COMPUTER APPLICATIONS
Semester – IV
SOFTWARE ENGINEERING(18CAU401)
LESSON PLAN
UNIT-V

S.NO	DURATION	TOPICS TO BE COVERED	SUPPORTED MATERIALS
1.	1	Testing Strategies & Tactics: Software Testing Fundamentals	T1:386,421 W1
2.	1	Strategic approach To Software Testing	T1:387-392
3.	1	Test Strategies For Conventional Software	T1:394-403
4.	1	Validation Testing, System Testing, Black-Box Testing	T1:406-410,434-436 W1
5.	1	White-Box Testing & Their Types	T1:436-440
6.	1	Basis Path Testing	T1:425-431
7.	1	RECAPITULATION OF IMPORTANT QUESTIONS	
8.	1	DISCUSSION OF ESE QUESTION PAPER	
9.	1	DISCUSSION OF ESE QUESTION PAPER	
10.	1	DISCUSSION OF ESE QUESTION PAPER	
TOTAL HOURS: 10 HOURS			

TEXTBOOK :

T1: Pressman, R.S.,(2009). *Software Engineering: A Practitioner's Approach*, (7th Edition), McGraw-Hill.

REFERENCE BOOKS:

R1: Aggarwal K.K., Singh,Y., (2008). *Software Engineering*, (2nd ed.), New Age International Publishers.

WEBSITE:

W1: http://en.wikipedia.org/wiki/Software_engineering

UNIT 1
SYLLABUS

Introduction: The Evolving Role of Software, Software Characteristics, Changing Nature of Software, Software Engineering as a Layered Technology, Software Process Framework, Framework and Umbrella Activities, Process Models, Capability Maturity Model Integration (CMMI).

1.1 INTRODUCTION TO SOFTWARE ENGINEERING:

Software has become critical to advancement in almost all areas of human Endeavour. The art of programming only is no longer sufficient to construct large programs. There are serious problems in the cost, timeliness, maintenance and quality of many software products. Software engineering has the objective of solving these problems by producing good quality, maintainable software, on time, within budget. To achieve this objective, we have to focus in a disciplined manner on both the quality of the product and on the process used to develop the product.

Definition

At the first conference on software engineering in 1968, Fritz Bauer [FRIT68] defined software engineering as "The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines". Stephen Schacht [SCHA90] defined the same as "A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements". Both the definitions are popular and acceptable to majority. However, due to increase in cost of maintaining software, objective is now shifting to produce quality software that is maintainable, delivered on time, within budget, and also satisfies its requirements.

What is Software?

The product that software professionals build and then support over the long term. Software encompasses: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately store and manipulate information and (3) documentation that describes the operation and use of the programs.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- Customized products
  - Software that is commissioned by **a specific customer** to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Features of Software

- Its characteristics that make it different from other things human being build.

Features of such logical system:

- Software is developed or engineered; it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs). Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface E.g. graphical user interface, window, pull-down menus in library etc.

1.1.1 The Evolving Role of Software

Software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is information transformer— producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). Software delivers the most important product of our time—information.

Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer- based systems. The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application.

1.1.2 Software, Software Myths

Software

Computer software, or just **software**, is a collection of computer programs and related data that provide the instructions for telling a computer

what to do and how to do it. In other words, software is a conceptual entity which is a set of computer programs, procedures, and associated documentation concerned with the operation of a data processing system. We can also say software refers to one or more computer programs and data held in the storage of the computer for some purposes.

In other words software is a set of **programs, procedures, algorithms** and its **documentation**. Program software performs the function of the program it implements, either by directly providing instructions to the computer hardware or by serving as input to another piece of software.

The term was coined to contrast to the old term hardware (meaning physical devices). In contrast to hardware, software is intangible, meaning it "cannot be touched". Software is also sometimes used in a more narrow sense, meaning application software only. Sometimes the term includes data that has not traditionally been associated with computers, such as film, tapes, and records.

Software Myths

The following are different myths about software:

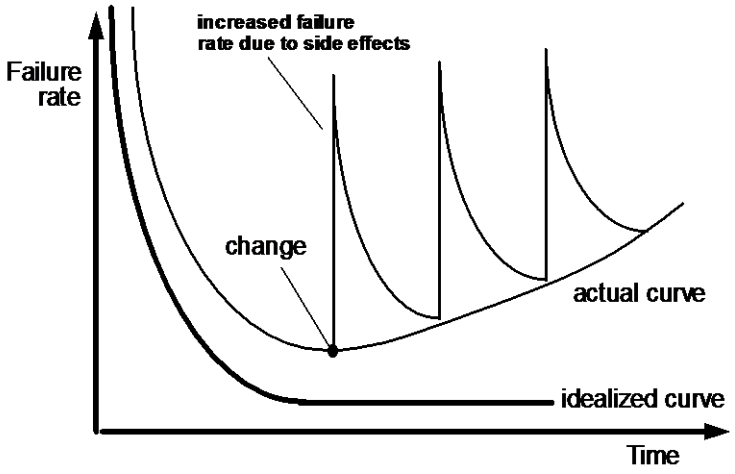
- If we get behind schedule, we can add more programmers and catch up.
- If we decide to outsource the software project to a third party, we can just relax and let that firm build it.
- Project requirement continuously changes, but changes can be easily accommodated because software is flexible.
- The only deliverable work product for a successful project is the working program.
- Software with more features is better software.
- Once we write the program and get it to work, our job is done.
- Until we get the program running, we have no way of assessing its quality.
- Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.
- A general statement of objectives is sufficient to begin writing programs; we can fill in the details later.

- A general statement of objectives is sufficient to begin writing programs; we can fill in the details later.
- We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Software Myths Examples

- Myth 1: Once we write the program and get it to work, our job is done.
- Reality: the sooner you begin writing code, the longer it will take you to get done. 60% to 80% of all efforts are spent after software is delivered to the customer for the first time.
- Myth 2: Until I get the program running, I have no way of assessing its quality.
- Reality: technical review are a quality filter that can be used to find certain classes of software defects from the inception of a project.
- Myth 3: software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.
- Reality: it is not about creating documents. It is about creating a quality product. Better quality leads to a reduced rework. Reduced work results in faster delivery times.
- Many people recognize the fallacy of the myths. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach.

Wear vs. Deterioration



Software Applications

1. System software: such as compilers, editors, file management utilities
2. Application software: stand-alone programs for specific needs.
3. Engineering/scientific software: Characterized by "number crunching"algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc
4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
5. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

1.2 A Generic View of Process

1.2.1 Software Engineering as a Layered Technology

Rests on organizational approach to quality, e.g. total quality management and such emphasize continuous process improvement (that is increasingly more effective approaches to software engineering)

Process layer provides a framework for effective use of software technologies. Forms a basis for management control and establishes context in which technical methods are applied, work products produced, milestones established, quality is ensured and change is managed. Methods provide technical "how to's". Methods include a broad range of tasks that include communication, requirements analysis, design modeling, programming, and testing. Software engineering methods rely on a set of basic principles that govern each area of the technology. Tools provide automated or semi automated support for the process and methods.

Process Framework

Identifies a small number of framework activities that are applicable to all software projects. In addition the framework encompasses umbrella activities that are applicable across the software process.

Generic Process Framework Activities

Each framework activity is populated by a set of software engineering actions. An action, e.g. design, is a collection of related tasks that produce a major software engineering work product.

Communication – lots of communication and collaboration with customer and other stakeholders.. Encompasses requirements gathering.

Planning – establishes plan for software engineering work that follows. Describes technical tasks, likely risks, required resources, works products and a work schedule

Modeling – encompasses creation of models that allow the developer and customer to better understand software requirements and the design that will achieve those requirements.

Construction – code generation and testing.

Deployment – software, partial or complete, is delivered to the customer who evaluates it and provides feedback.

Modeling Activity – composed of two software engineering actions

Analysis – composed of work tasks (e.g. requirement gathering, elaboration, specification and validation) that lead to creation of analysis model and/or requirements specification.

Design – encompasses work tasks such as data design, architectural design, interface design and component level design. It leads to creation of design model and/or a design specification. Different projects demand different task sets. Software team chooses task set based on problem and project characteristics.

A Layered Technology

Quality, Process, Methods, and Tools

Software engineering is a layered technology. Most engineering approaches (including software engineering) must rest on an organizational commitment to quality. The bedrock that supports software engineering is a

quality focus'layer.

-Quality: a product should meet its specification. This is problematical for software systems. There is a tension between customer quality requirements (efficiency, reliability, etc.), developer quality requirements (maintainability, reusability, etc.), users (usability, efficiency, etc.), and etc. But note:

- Some quality requirements are difficult to specify in an unambiguous way.
- Software specifications are usually incomplete and often inconsistent.

-Process: The foundation for software engineering is the ***process***'layer. Software engineering process is the glue that holds the technology together and enables rational and timely development of computer software. The work products are produced, milestones are established, quality is ensured, and changes are properly managed.

-Methods: Software engineering ***methods*** provides the technical how-to's for building software. Methods encompass a broad array of tasks that include the requirements analysis, design, program construction, testing, and support.

-Tools: Software engineering ***tools*** provide automated or semi-automated supports for the process and the methods. When the tools are integrated so that the information created by one tool can be used by another, a system for the support of software development, called ***computer-aided software engineering (CASE)***. CASE combine software, hardware, and software engineering database.

Fig 1.2.2. A Layered Technology

- Any engineering approach must rest on organizational commitment to quality which fosters a continuous process improvement culture.
- Process layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.
- Method provides technical how-to's for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.
- Tools provide automated or semi-automated support for the process and methods.

Five Activities of a Generic Process framework

- Communication: communicate with customer to understand objectives and gather requirements
- Planning: creates a "map" defines the work by describing the tasks, risks and resources, work products and work schedule.
- Modeling: Create a "sketch", what it looks like architecturally, how the constituent parts fit together and other characteristics.
- Construction: code generation and the testing.
- Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.
- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.

- For many software projects, these framework activities are applied iteratively as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

Umbrella Activities

Complement the five process framework activities and help team manage and control progress, quality, change, and risk.

- Software project tracking and control: assess progress against the plan and take actions to maintain the schedule.
- Risk management: assesses risks that may affect the outcome and quality.
- Software quality assurance: defines and conduct activities to ensure quality.
- Technical reviews: assesses work products to uncover and remove errors before going to the next activity.
- Measurement: define and collects process, project, and product measures to ensure stakeholder's needs are met.
- Software configuration management: manage the effects of change throughout the software process.
- Reusability management: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- Work product preparation and production: create work products such as models, documents, logs, forms and lists.

Objectives

- To introduce software process models
- To describe three generic process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution
- To explain the Rational Unified Process model
- To introduce CASE technology to support software process activities

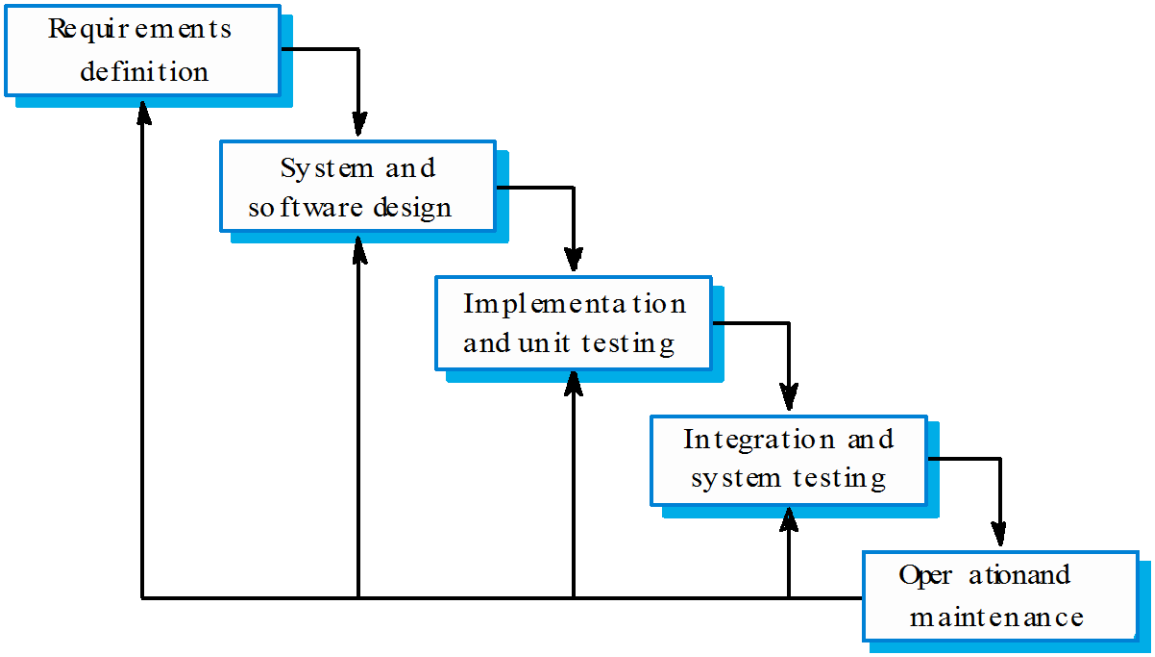
1.3 The Software Process Models

- 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

- The waterfall model
  - Separate and distinct phases of specification and development.
- Evolutionary development
  - Specification, development and validation are interleaved.
- Component-based software engineering
  - The system is assembled from existing components.
- There are many variants of these models e.g. formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.

1.3.1 Waterfall model



Waterfall Model

The classic software life cycle is often represented as a simple prescriptive waterfall software phase model, where software evolution proceeds through an orderly sequence of transitions from one phase to the next in order (Royce 1970). Such models resemble finite state machine descriptions of software evolution.

However, these models have been perhaps most useful in helping to structure, staff, and manage large software development projects in complex organizational settings, which was one of the primary purposes (Royce 1970, Boehm 1976).Alternatively, these classic models have been widely characterized as both poor descriptive and prescriptive models of how software development "in-the-small" or "in-the-large" can or should occur. Figure 1 provides a common view of the waterfall model for software development attributed to Royce (1970).

Waterfall model phases

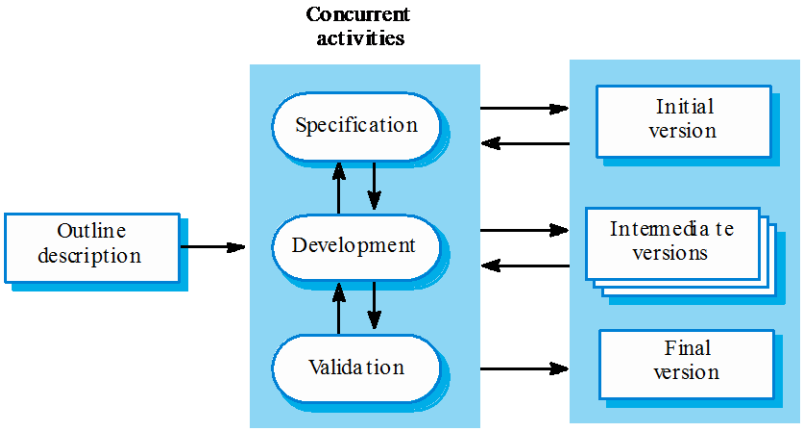
- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.

Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

1.3.2 Evolutionary development

- Exploratory development
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.
- Throw-away prototyping
  - Objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.



Evolutionary development

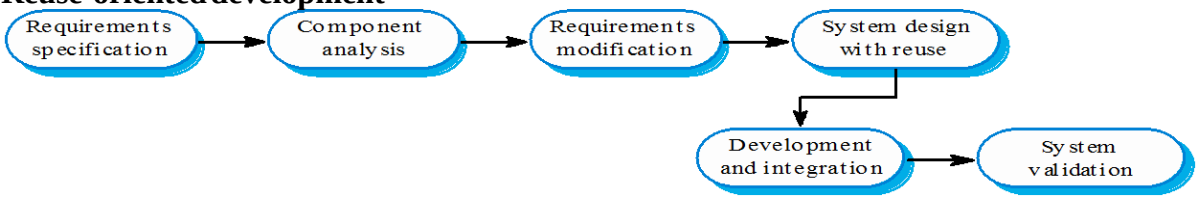
- Problems
  - Lack of process visibility;
  - Systems are often poorly structured;
  - Special skills (e.g. in languages for rapid prototyping) may be required.
- Applicability
  - For small or medium-size interactive systems;
  - For parts of large systems (e.g. the user interface);
  - For short-lifetime systems.

Component-based software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.

- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- This approach is becoming increasingly used as component standards have emerged.

Reuse-oriented development



1.3.3 Incremental Process Model

Incremental Model : The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software.

In incremental model, the first increment is often a core product. Here the basic requirements are addressed, but many supplementary features remain undelivered. The core product is used by the customer to develop a plan for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

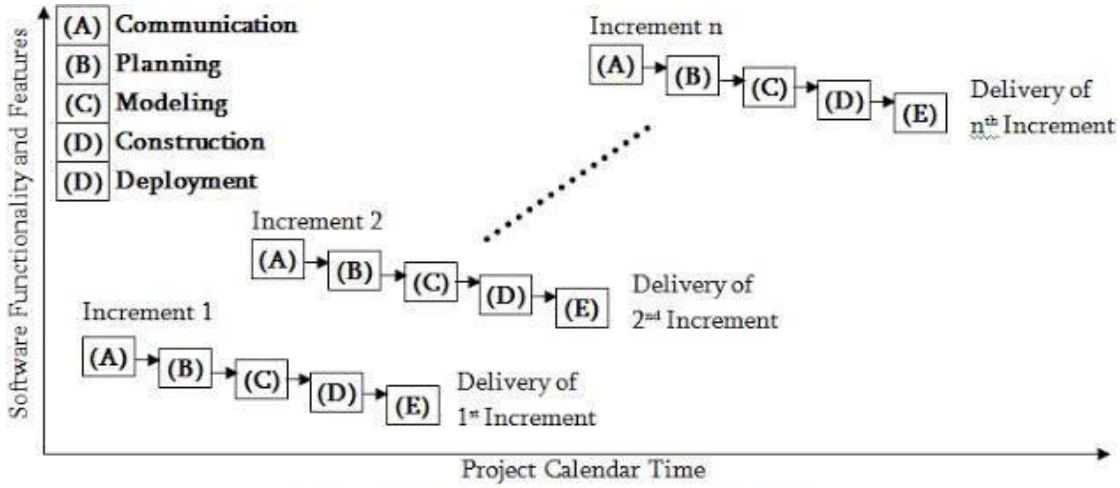


Figure : Flowchart of Incremental Model

Incremental development is particularly useful when staffing is unavailable for a complete implementation. If the core product is well received with fewer staffs then additional staff, if required), can be added to implement the next increment. Increments can be planned to manage technical risks. For example, a major system might require new hardware that is under development. Then it might be possible to plan early increments in a way that avoids the use of this hardware.

1.3.4 Evolutionary Software Process Model

Model Types

Software Products can be perceived as evolving over a time period.

However, neither the Linear Sequential Model nor the Prototype Model applies this aspect to software production. The Linear Sequential Model was designed for straight-line development. The Prototype Model was designed to assist the customer in understanding requirements and is designed to produce a visualization of the final system.

But the Evolutionary Models take the concept of "evolution" into the engineering paradigm. Therefore Evolutionary Models are **iterative.** They are built in a manner that enables software engineers to develop increasingly more complex versions of the software.

1.3.4.1 Prototyping

Prototyping Model: Often, a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach.

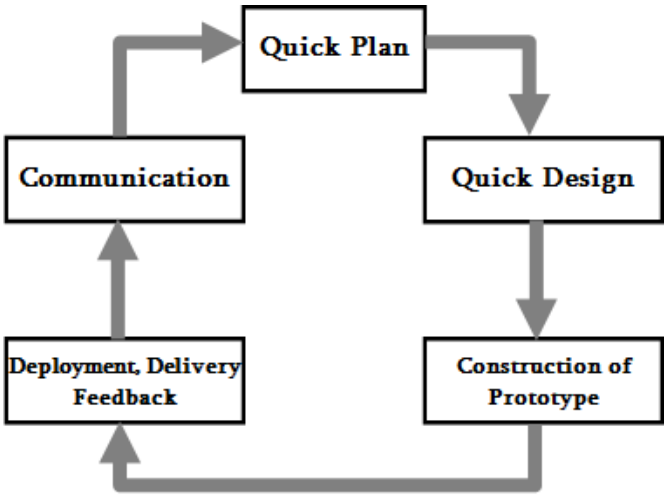


Figure : Flowchart of Prototyping Model

(i) Communication: Firstly, developer and customer meet and define the overall objectives, requirements, and outline areas where further definition is mandatory.

(ii) Quick Plan: Based on the requirements and others of the communication part a quick plane is made to design the software.

(iii) Modeling Quick Design: Based on the quick plane, 'A Quick Design' occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user, such as input approaches and output formats.

(iv) Construction of Prototype: The quick design leads to the construction of a prototype.

(v) Deployment, delivery and feedback: The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. All these steps are repeated to tune the prototype to satisfy user's need. At the same time enable the developer to better understand what needs to be done.

Problems with Prototype Model:

(i) In the rush to get the software working the overall software quality or long-term maintainability will not get consideration. So software, in that way, gets the need of excessive modification to ensure quality.

(ii) The developer may choose inappropriate operating system, algorithms, language in the rush to make the prototype working.

Prototyping is an effective paradigm for software engineering. It necessary to build the prototype to define requirements and then to engineer the software with a eye toward quality.

1.3.4.2 The Spiral Model

Spiral Model: The spiral model is an evolutionary software process model that combines the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions. Given figure is of a spiral model that contains five task regions.

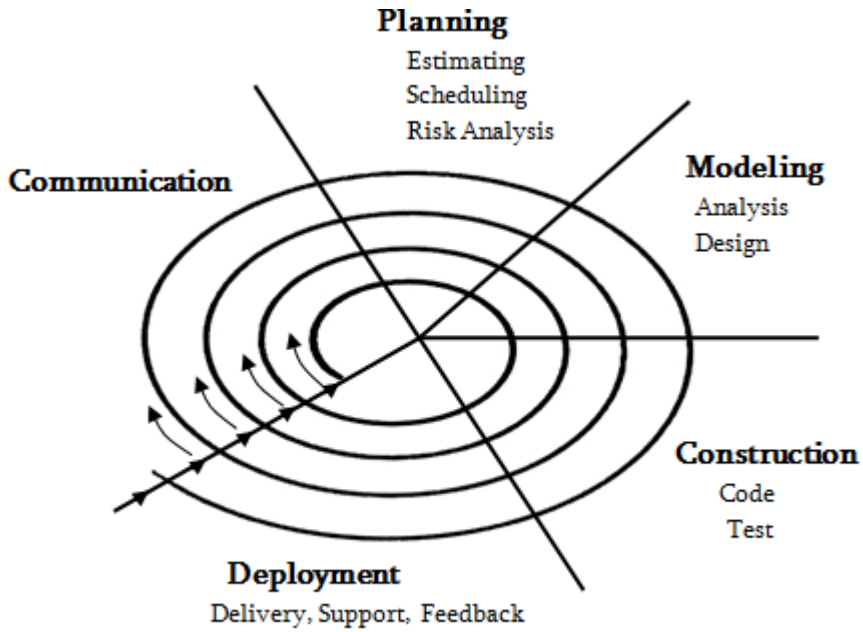


Figure : Spiral Model

Model that contains five task regions.

(i) Customer communication — Tasks required to establish effective communication between developer and customer.

(ii) Planning — Tasks required to define resources, timelines, and other project related information.

(iii) Modeling — Tasks required in building one or more representations of the application.

(iv) Construction and release — Tasks required to construct, test, install.

(v) Deployment — Tasks required to deliver the software, getting feedbacks etc.

Software engineering team moves around the spiral in a clockwise direction, beginning at the center. The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from customer evaluation.

In addition, the project manager adjusts the planned number of iterations required to complete the software.

The spiral model is a realistic approach to the development of large- scale systems and software. The spiral model enables the developer to apply the prototyping approach at any stage in the evolution of the product. The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic. It demands considerable risk assessment expertise and relies on this expertise for success.

Process activities

- Software specification
- Software design and implementation
- Software validation
- Software evolution

1.4 Specialized process model

Specialized process models take on many of the characteristics of one or more of the traditional models presented in the preceding sections. However, these models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.

1.4.1 Component-Based Development

- Consists of the following process steps
  - Available component-based products are researched and evaluated for the application domain in question
  - Component integration issues are considered
  - A software architecture is designed to accommodate the components
  - Components are integrated into the architecture
  - Comprehensive testing is conducted to ensure proper functionality
- Relies on a robust component library
- Capitalizes on software reuse, which leads to documented savings in project cost and time

1.4.2 The Formal Methods Model

- Encompasses a set of activities that leads to formal mathematical specification of computer software
- Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- Offers the promise of defect-free software
- Used often when building safety-critical systems
- Development of formal methods is currently quite time-consuming and expensive
- Because few software developers have the necessary background to apply formal methods, extensive training is required
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers

1.4.3 Aspect-Oriented Software Development

Regardless of the software process that is chosen, the builders of complex software invariably implement a set of localized features, functions, and information content. These localized software characteristics are modeled as components (e.g., objectoriented classes) and then constructed within the context of a system architecture. As modern computer-based systems become more sophisticated (and complex), certain *concerns*—customer required properties or areas of technical interest—span the entire architecture. Some concerns are high-level properties of a system (e.g., security, fault tolerance). Other concerns affect functions (e.g., the application of business rules), while others are systemic (e.g., task synchronization or memory management).

When concerns cut across multiple system functions, features, and information, they are often referred to as *crosscutting concerns. Aspectual requirements* define those crosscutting concerns that have an impact across the software architecture.

Aspect-oriented software development (AOSD), often referred to as *aspect- oriented programming* (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*—"mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern"

POSSIBLE QUESTIONS

1. Define software.
2. define software engineering.
3. write about the characteristics of software.
4. write a note on nature of software.
5. what is meant by layered technology in software engineering? 6.what are the process in software?
7. write a note about the software models.
8. what are the process activities in software?
9. why software is important?
10. what are the features of software?

SOFWARE ENGINEERING (17CAU401)

Class: II BCA

Semester : IV

Unit - I

S.No.	Question	Choice 1	Choice 2	Choice 3	Choice 4
1	Software takes on a _____ role.	single	dual	triple	tetra
2	Software is a _____.	virtual	system	modifier	framework
3	Instructions that when executed provide desired function and performance is called	software	hardware	firmware	humanware
4	High quality of software is achieved through _____.	testing	good design	construction	manufacture
5	Software doesn't _____.	tearout	wearout	degrade	deteriorate
6	Software is not susceptible to _____.	hardware	defects	environmental melodies	deterioration
7	Software will undergo _____.	database	testing	enhancement	manufacture
8	_____ refers to the meaning and form of incoming and outgoing information.	content	software	hardware	data
9	_____ refers to the predictability of the order and timing of information.	system software	network software	information determinacy	database
10	_____ is not a system software.	MS Office	compiler	editor	file management utility
11	Collection of programs written to service other programs are called _____.	system software	business software	embedded software	d. pc software
12	Which one is not coming under software myths	Management myths	customer myths	product myths	practitioners myths
13	_____ is a PC Software.	MS word	LISP	CAD	C
14	Software that monitors, analyses, controls real world events is called _____.	Business software	real time software	web based software	d. embedded software
15	The bedrock that supports software engineering is a_____	tools	methods	process models	a quality focus
16	A complete software process by identifying a small number of _____	framework activities	umbrella activities	framework process	software process
17	The process framework encompassess a set of _____	framework activities	umbrella activities	framework process	software process
18	software engineering action is_____	design	chronic	decision	crisis
19	Which one is effect the outcome of the project?	Risk management	Measurement	technical reviews	Reusability
20	Continuing indefinitely is called _____.	crisis	decision	affliction	chronic
21	Component based development uses _____.	functions	subroutines	procedures	objects

22	UML stands for _____.	Universal Modelling Language	User Modified Language	Unified Modelling Language	User Model Language
23	A model which uses formal mathematical specification is called _____.	4 GT model	Unified method model	formal methods model	component based development
24	A variation of formal methods model is called _____.	component based development	4 GT model	unified method model	cleanroom software engineering
25	The development of formal methods is _____.	less time consuming	quite time consuming	does not consume time	very less time consuming
26	The first step to develop software is _____.	analysis	design	requirements gathering	coding
27	The waterfall model sometimes called as	classic model	classic life cycle model	life cycle model	cycle model
28	Software engineering activities include _____	decision	affliction	hardware	maintenance
29	all process model prescribes a _____.	circular	elliptical	spiral	workflow
30	Component based development incorporates the characteristics of the _____ model	circular	elliptical	spiral	hierarchical
31	Prototype is a _____.	software	hardware	computer	model
32	For small applications it is possible to move from requirement gathering step to_____.	analysis	implementation	design	modeling
33	Software process model includes _____.	software	analysis	hardware	computer
50	Software project management begins with a set of activities that are collectively called _____	project planning	software scope	software estimation	decomposition
51	_____.	project planning	software scope	estimation	decomposition
52	The ease with which software can be transferred from one computer to another. This quality attribute is called _____.	portability	reliability	efficiency	accuracy
53	The ability of a program to perform a required function under stated condition for a stated period of time. This quality attribute is called _____.	portability	reliability	efficiency	accuracy
54	The event to which software performs its intended function. This quality attribute is called _____.	portability	reliability	efficiency	accuracy
55	A qualitative assessments of freedom from errors. This quality attribute is called _____.	portability	reliability	efficiency	accuracy
56	The extent to which software can continue to operate correctly. This quality attribute is called _____.	robustness	correctness	efficiency	reliability
57	The extent to which the software is free from design and coding defects ie fault free. This quality attribute is called _____.	robustness	correctness	efficiency	reliability

58	System shall reside in 50KB of memory is an example of _____.	quantified requirement	qualified requirement	functional requirement	performance requirement
59	Accuracy shall be sufficient to support mission is an example of _____.	quantified requirement	qualified requirement	functional requirement	performance requirement
60	System shall make efficient use of memory is an example of _____.	quantified requirement	qualified requirement	functional requirement	performance requirement

Answer
dual
modifier
software
good design
wearout
environmental
melodies
enhancement
content
information
determinacy
MS Office
system software
product myths
MS word
real time
software
a quality focus
framework
activities
umbrella
activities
design
Risk
management
chronic
objects

Unified
Modelling
Language
formal methods
model
cleanroom
software
engineering
quite time
consuming
requirements
gathering
classic life cycle
model
maintenance
workflow

spiral
model

modeling

analysis

project planning

decomposition

portability

reliability

efficiency

accuracy

robustness

correctness

quantified
requirement
qualified
requirement
qualified
requirement

Unit - 2

1	The _____ as a bridge between the systm decription and the design model	design	analysis model
2	The role of the software engineer in the requirement analysis is called____	designer	analyst
3	analysis modeling often begins with _____	data modeling	function modeling
4	A _____ can be an external entity	function object	structural object
5	_____ defines the properities of a data object	relationship	cardinality
6	Data objects are connected to one another in different ways is called _____	modality	relationships
7	_____ is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object	modality	relationships
8	_____ defines the maximum number of objects that can participate in a relationship	modality	relationships
9	_____ provide an indication of whether or not a particular data object must participate in the relationship	Modality	relationships
10	The _____ diagram takes an an input-process-output view of sysytem	use-case diagram	activity diagram
11	The level 0 DFD is called as _____ diagram	contract level diagram	context level diagram
12	The _____ describes the behavior of the system but not the inner working of the processes	PSPEC	ASPEC
13	The _____ is used to describe all flow model processes the appear in the final level of refinement	CSPEC	ASPEC
14	The _____ model indicates how software will respond to external events	data	behavior
15	The _____ represents a sequence of activities that involves actor and the system	csase tool	activity
16	The _____ diagram indicates how events cause transitions from object to object	sequence diagram	activity
17	Which one depict the software requirements from the user's point of view.	behavioral based	flow based model
18	Which model depicts how input is transformed into output as data objects move through a system	behavioral based	flow based model
19			
20			

Unit 3

1	_____is a iterativeprocess through which requirements are translated into a blueprint for constructing the software	requirements gathering	coding
2	Who developeda set of software quality attributefor the software design	Barry Boehm	R.Pattis
3	Which quality attribute measure the response time, throughput and effeciency of the sysytem	Functionality	Usability

4	The quality attribute, Usability is assessed by considering the overall _____ of the system	consistency	Functionality
5	A _____ refers to a sequence of instructionsthat have a specific and limited functions	procedural abstraction	data abstraction
6	_____ represent architecture as an organized collection of programs components	process models	structural models
7	_____ models address the behavioral aspects of the program architecture	process models	structural models
8	Software is divided into separately named and addressable components is called _____	process	behavior
9	the _____ is a process of changing a software by which doesnot alter the external behavior of the code	refinement	cohesion
10	_____ is an indication of the relative functional strength of the module	refinement	cohesion
11	_____ is an indication of the relativeinterdependency among modules	cohesion	patterns
12	Refinement is a top-down design strategy which is actually a process of _____	eloboration	abstraction
13	A _____ is a named collection of data that describes a data object.	procedural abstraction	data abstraction
14	_____ implies a program control mechanism without specifying internal detail.	procedural abstraction	data abstraction
15	software architecture consider ____ levels of the design pyramid	3	2
16	Which action translates data objects into data structures	data design	component design
17	In data centered arcjitecture _____ resides at the centre of the architecture which is accessed frequently by other components	client software	data store
18	_____ represents the structure of data and program components that are required to build a computer-based syste,	architectural design	data design
19	KDD stand for	Knowledge Discovery of data manipulatio	Knowledge Discovery in database
20	the _____ classes defines all abstraction that are necessary for human computer interaction	primitive class	user interface
21	The _____ classes implement lower level business abstraction required to manage the business domain class	primitive class	user interface
22	_____ suggest that a method should send or receive messages from friend class	Law of cohesion	Law of meter
23	_____ is achieve by developing modules with single minded function and aversion of excessive interaction	refinement	refactoring
24	_____ suggest that the information contained in one module is inaccesible to othe modules	refinement	refactoring
25	Refinement is a process of _____	abstraction	eloboration
26	_____ is a process of breaking up of complex problem into a manageable piecies	refinement	refactoring
27	_____ is evaluated by measuring the frequency and severity of failure.	Usability	performance

28	iIn transform flow the information must entered and exit in _____ form	external world	internal world
29	Information flow is characterized by an single data item is called	context flow	transaction flow
30	A _____ diagram is mapped into a program structure using transform or transaction mapping	data flow	use case
31	_____ language provides a semantic and syntax for describing a software architecture	architectural description	architectural design
32	Design begins with the _____ model.	data	requirements
33	_____ focus on the design of the business or technical process that the system must accommodate.	framework models	dynamic models
34	_____ can be used to represent the functional hierarchy of a system.	framework models	dynamic models
35	_____ represent architecture as an organized collection of program components.	dynamic models	functional models
36	_____ increases the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.	framework models	dynamic models
37	_____ address the behavioural aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.	framework models	dynamic models
38	_____ is the place where quality is fostered in software engineering	model	data
39	_____ provides us with representations of software that can be assessed for quality.	design	specification
40	_____ describes a program should not have any bugs that inhibit its function	firmness	commodity
41	A program should be suitable for the purposes for which it is intended is called	firmness	commodity
42	The experience of using the program should be a pleasurable one is called	firmness	commodity
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			

planning	construction	analysis model
programmer	tester	analyst
behavior modling	structure modeling	
Data object	flow object	Data object
Data attributes	modality	Data attributes
cardinality	Data attributes	relationships
cardinality	Data attributes	Cardinality
Data attributes	cardinality	Cardinality
Data attributes	cardinality	Modality
data flow diagram	ERD	data flow diagram
text level diagram	zero level diagram	context level diagram
LSPEC	CSPEC	CSPEC
LSPEC	PSPEC	PSPEC
function	structural	behavior
use-case	swimlane	use-case
use-case	swimlane	sequence diagram
class based	scenario based model	scenario based model
class based	scenario based model	flow based model
software design	deployment	software design
M.C.Escher	Hewlett-Packard	Hewlett-Packard
Performance	Supportability	Performance

Supportability	Performance	consistency
behavior abstraction	structural abstraction	procedural abstraction
dynamic models	framework models	structural models
dynamic models	framework models	dynamic models
modules	data	Modules
patterns	refactoring	refactoring
patterns	refactoring	cohesion
coupling	functional dependency	coupling
refactoring	information hiding	eloboration
control abstraction	behavior abstraction	data abstraction
control abstraction	behavior abstraction	control abstraction

1	4	2
behavior design	functional design	data design
filter	pipes	data store
software design	behavioural design	architectural design
Knowing of database discovery process	Knowing discovery of database business	Knowledge Discovery in database
classes	domain	user interface
process classes	System class	process classes
Low of completeness	law of primitiveness	Law of meter
functional independence	information hiding	functional independence
functional independence	information hiding	information hiding
architecture	modularity	eloboration
modularity	arichiteture	modularity
supportability	reliability	reliability

top down	bottom up	external world
transform flow	contract flow	transaction flow
state diagram	activity diagram	data flow
architectural pattern	architecturaldefinition	architectural description
specification	code	requirements
process models	functional models	process models
process models	functional models	functional models
framework models	structural models	structural models
process models	functional models	framework models
process models	functional models	dynamic models
design	specification	design
data	prototype	design
delight	roman	firmness
delight	roman	commodity
delight	roman	delight

UNIT 2
SYLLABUS

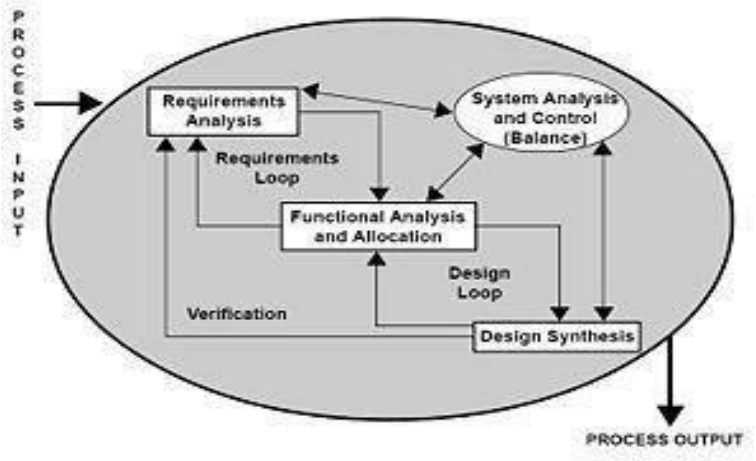
Requirement Analysis; Initiating Requirement EngineeringProcess- Requirement Analysis and Modeling Techniques- FlowOriented Modeling- Need for SRS- Characteristics and Components of SRS- Software Project Management: Estimation in Project Planning Process, Project Scheduling.

BUILDING THE ANALYSIS MODEL:

2.1 Requirements Analysis:

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users. It is an early stage in the more general activity of requirements engineering which encompasses all activities concerned with eliciting, analyzing, documenting, validating and managing software or system requirements.

Requirements analysis is critical to the success of a systems or software project. The Requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.



Requirements Engineering Tasks:

Types of Requirements

Customer Requirements

Statements of fact and assumptions that define the expectations of the system in terms of mission objectives, environment, constraints, and measures of effectiveness and suitability (MOE/MOS). The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer. Operational requirements will define the basic need and, at a minimum, answer the questions posed in the following listing

- *Operational distribution or deployment*: Where will the system be used?
- *Mission profile or scenario*: How will the system accomplish its mission objective?

- *Performance and related parameters*: What are the critical system parameters to accomplish the mission?
- *Utilization environments*: How are the various system components to be used?
- *Effectiveness requirements*: How effective or efficient must the system be in performing its mission?
- *Operational life cycle*: How long will the system be in use by the user?
- *Environment*: What environments will the system be expected to operate in an effective manner?

Architectural Requirements

Architectural requirements explain what has to be done by identifying the necessary system architecture of a system.

Structural Requirements

Structural requirements explain what has to be done by identifying the necessary structure of a system.

Behavioral Requirements

Behavioral requirements explain what has to be done by identifying the necessary behavior of a system.

Functional Requirements

Functional requirements explain what has to be done by identifying the necessary task, action or activity that must be accomplished. Functional requirements analysis will be used as the toplevel functions for functional analysis.

Non-functional Requirements

Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviors.

Performance Requirements

The extent to which a mission or function must be executed; generally measured in terms of quantity, quality, coverage, timeliness or readiness. During requirements analysis, performance (how well does it have to be done) requirements will be interactively developed across all identified functions based on system life cycle factors; and characterized in terms of the degree of certainty in their estimate, the degree of criticality to system success, and their relationship to other requirements.

Design Requirements

The "build to," "code to," and "buy to" requirements for products and "how to execute" requirements for processes expressed in technical data packages and technical manuals.

Derived Requirements

Requirements that are implied or transformed from higher-level requirement. For example, a requirement for long range or high speed may result in a design requirement for low weight.

Allocated Requirements

A requirement that is established by dividing or otherwise allocating a high-level requirement into multiple lower-level requirements. Example: A 100-pound item that consists of two subsystems might result in weight requirements of 70 pounds and 30 pounds for the two lower-level items.

Building the Analysis Model:

Data Flow Diagram (DFD)

Represents how data objects are transformed as they move through the system
Input-Process-Output (I-P-O) view of software.

Flow model

computer
based
system
input output
Every computer-based system is an

Process

A data transformer (changes input to output)
Examples: compute taxes, determine area,format report, display graph Data must always be processed in some way to achieve system function

Flow

Data flows through a system, beginning as input and be transformed into output.

Data Store

Data is often stored for later use.
look-up
sensor
data
sensor #
report required
sensor #, type,
location, age
sensor data
sensor number
type,
location, age

Guideline for DFD

• All icons must be labeled with meaningful names
• The DFD evolves through a number of levels of detail
• Always begin with a context level diagram (also called level 0)
• Always show external entities at level 0
• Always label data flow arrows
• do not represent procedural logic

Constructing DFD-1

• review the data model to isolate data objects and use a grammatical parse to determine "operations"
• determine external entities (producers and consumers of data)

Constructing DFD-2

• write a narrative describing the transform
• parse to determine next level transforms
• "balance" the flow to maintain data flow continuity
• develop a level 1 DFD

Requirements are prone to issues of ambiguity, incompleteness, and inconsistency. Techniques such as rigorous inspection have been shown to help deal with these issues. Ambiguities, incompleteness, and inconsistencies that can be resolved in the requirements phase typically cost orders of magnitude less to correct than when these same issues are found in later stages of product development. Requirements analysis strives to address these issues.

There is an engineering trade off to consider between requirements which are too vague, and those which are so detailed that they

1. take a long time to produce - sometimes to the point of being obsolete once completed
2. limit the implementation options available
3. Are costly to produce.

At a technical level, software engineering begins with a series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software to be built. The analysis model, actually a set of models, is the first technical representation of a system. Over the years many methods have been proposed for analysis modeling.

However, two now dominate. The first, structured analysis is a classical modeling method. The other approach, object oriented analysis.

Structured analysis is a model building activity. Applying the operational analysis principles we create and partition data, functional, and behavioral models that depict the essence of what must built.

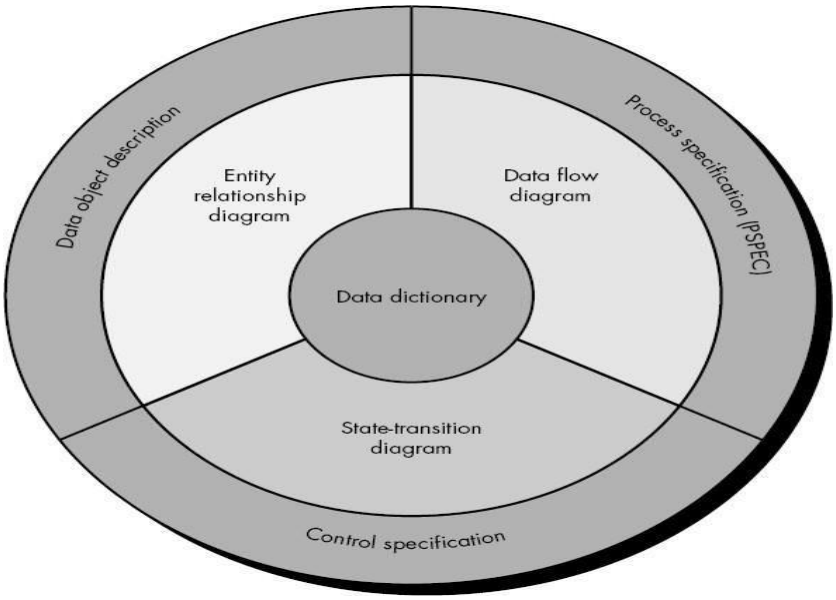
The Elements of the Analysis Model

The analysis model must achieve three primary objectives:

1. To describe what the customer requires.
2. To establish a basis for the creation of a software design.

3. To define a set of requirements that can be validated once the software is built.

To accomplish these objectives, the analysis model derived during structured analysis takes the form illustrated in



The structure of the analysis model

- At the core of the model lies the data dictionary—a repository that contains descriptions of all data objects consumed or produced by the software.

- Three different diagrams surround the core.

- The entity relation diagram (ERD) depicts relationships between data objects. The ERD is the notation that is used to conduct the data modeling activity.

The attributes of each data object noted in the ERD can be described using a data object description.

- The data flow diagram (DFD) serves two purposes:

1. To provide an indication of how data are transformed as they move through the system.

2. To depict the functions (and subfunctions) that transform the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modeling of function.
A description of each function presented in the DFD is contained in a process specification (PSPEC).

- The state transition diagram (STD) indicates how the system behaves as a consequence of external events. To accomplish this, the STD represents the various modes of behavior (called states) of the system and the manner in which transitions are made from state to state. The STD serves as the basis for behavioral modeling.

Additional information about the control aspects of the software is contained in the control specification (CSPEC).
The analysis model encompasses each of the diagrams, specifications, descriptions, and the dictionary noted in Figure 1.

2.2 Analysis Modeling Approaches:

- **Structured analysis**
  - Considers data and the processes that transform the data as separate entities
  - Data is modeled in terms of only attributes and relationships (but no operations)
  - Processes are modeled to show the 1) input data, 2) the transformation that occurs on that data, and 3) the resulting output data
- **Object-oriented analysis** –Focuses on the definition of classes and the manner in which they collaborate with one another to fulfill customer requirements

– Focuses on the definition of classes and the manner in which they collaborate with one another to fulfill customer requirements

2.3 Data Modeling Concepts:

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Therefore, the process of data modeling involves professional data modelers working closely with business stakeholders, as well as potential users of the information system. There are three different types of data models produced while progressing from requirements to the actual database to be used for the information system.

The data requirements are initially recorded as a conceptual data model which is essentially a set of technology independent specifications about the data and is used to discuss initial requirements with the business stakeholders.

The conceptual model is then translated into a logical data model, which documents structures of the data that can be implemented in databases. Implementation of one conceptual data model may require multiple logical data models. The last step in data modeling is transforming the logical data model to a physical data model that organizes the data into tables, and accounts for access, performance and storage details. Data modeling defines not just data elements, but their structures and relationships between them.

Data modeling techniques and methodologies are used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The use of data modeling standards is strongly recommended for all projects requiring a standard means of defining and analyzing data within an organization, e.g., using data modeling:

- to manage data as a resource;
- For the integration of information systems.
- For designing databases/data warehouses.
- Data modeling may be performed during various types of projects and in multiple phases of projects. Data models are progressive; there is no such thing as the final data model for a business or application. Instead a data model should be considered a living document that will change in response to a changing business.

The data models should ideally be stored in a repository so that they can be retrieved, expanded, and edited over time. Whitten (2004) determined two types of data modeling Strategic data modeling: This is part of the creation of an information systems strategy, which defines an overall vision and architecture for information systems is defined. Information engineering is a methodology that embraces this approach.

Data modeling during systems analysis: In systems analysis logical data models are created as part of the development of new databases.

Data modeling is also used as a technique for detailing business requirements for specific databases. It is sometimes called *database modeling* because a data model is eventually implemented in a database.

Data models

Data models provide a structure for data used within information systems by providing specific definition and format. If a data model is used consistently across systems then compatibility of data can be achieved. If the same data structures are used to store and access data then different applications can share data seamlessly.

The results of this are indicated in the diagram. However, systems and interfaces often cost more than they should, to build, operate, and maintain. They may also constrain the business rather than support it. This may occur when the quality of the data models implemented in systems and interfaces is poor.

- Business rules, specific to how things are done in a particular place, are often fixed in the structure of a data model. This means that small changes in the way business is conducted lead to large changes in computer systems and interfaces. So, business rules need to be implemented in a flexible way that does not result in complicated dependencies, rather the data model should be flexible enough so that changes in the business can be implemented within the data model in a relatively quick and efficient way.
- Entity types are often not identified, or are identified incorrectly. This can lead to replication of data, data structure and functionality, together with the attendant costs of that duplication in development and maintenance. Therefore, data definitions should be made as explicit and easy to understand as possible to minimize misinterpretation and duplication.
- Data models for different systems are arbitrarily different. The result of this is that complex interfaces are required between systems that share data. These interfaces can account for between 25-70% of the cost of current systems. Required interfaces should be considered inherently while designing a data model, as a data model on its own would not be usable without interfaces within different systems.
- Data cannot be shared electronically with customers and suppliers, because the structure and meaning of data has not been standardized. To obtain optimal value from an implemented data model, it is very important to define standards that will ensure that data models will both meet business needs and be consistent.

2.3.1 Data Objects, Data Attributes

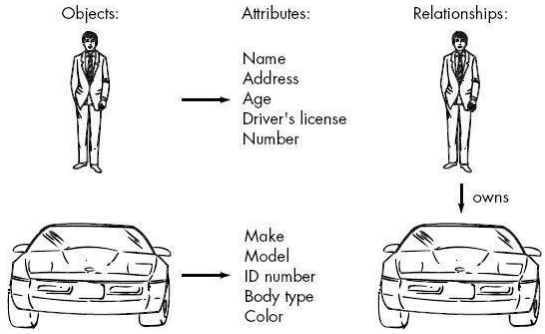
Data Objects

A *data object* is a representation of almost any composite information that must be understood by software. By *composite information,* we mean something that has a number of different properties or attributes. Therefore, width (a single value) would not be a valid data object, but dimensions (incorporating height, width, and depth) could be defined as an object.

A data object can be an external entity (e.g., anything that produces or consumes information), a thing (e.g., a report or a display), and an occurrence (e.g., a telephone call)

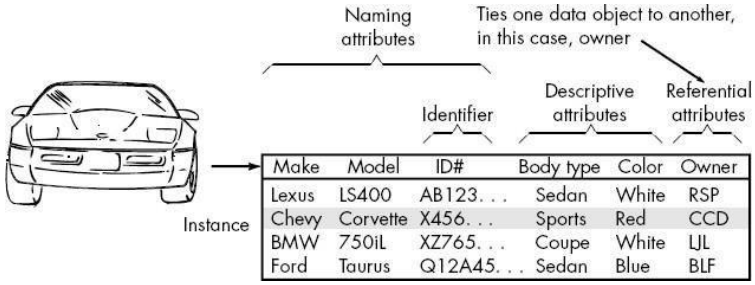
EX:

A person or a car (Figure 2) can be viewed as a data object in the sense that either can be defined in terms of a set of attributes. The data object description incorporates the data object and all of its attributes. Or event (e.g., an alarm), a role (e.g., salesperson), an organizational unit (e.g., accounting department), a place (e.g., a warehouse), or a structure (e.g., a file).



Data objects (represented in bold) are related to one another. For example, **person** can *own* **car**, where the relationship *own* connotes a specific "connection" between **person** and **car**. The relationships are always defined by the context of the problem that is being analyzed.

A data object encapsulates data only—there is no reference within a data object to operations that act on the data (This distinction separates the data object from the *class* or *object* defined as part of the object-oriented paradigm). Therefore, the data object can be represented as a table as shown in Figure 3. The headings in the table reflect attributes of the object. In this case, a car is defined in terms of make, model, ID number, body type color and owner. The body of the table represents specific instances of the data object. **For example, a Chevy Corvette is an instance of the data object** car.



Make	Model	ID#	Body type	Color	Owner
Lexus	LS400	AB123. . .	Sedan	White	RSP
Chevy	Corvette	X456. . .	Sports	Red	CCD
BMW	750iL	XZ765. . .	Coupe	White	LJL
Ford	Taurus	Q12A45. . .	Sedan	Blue	BLF

Data Attributes

Attributes define the properties of a data object and take on one of three different characteristics. They can be used to (1) name an instance of the data object, (2) describe the instance, or (3) make reference to another instance in another table.

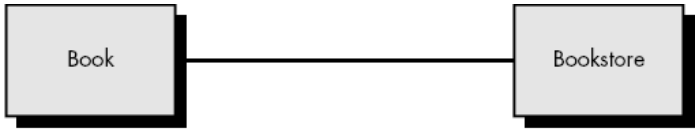
In addition, one or more of the attributes must be defined as an *identifier*—that is, the identifier attribute becomes a "key" when we want to find an instance of the data object. In some cases, values for the identifier(s) are unique, although this is not a requirement. Referring to the data object car, a reasonable identifier might be the **ID** number.

The set of attributes that is appropriate for a given data object is determined through an understanding of the problem context.

Data objects are connected to one another in different ways. Consider two data objects, **book** and **bookstore**. These objects can be represented using the simple notation illustrated in Figure 4a. A connection is established between **book** and **bookstore** because the two objects are related. But what are the relationships? To determine the

answer, we must understand the role of books and bookstores within the context of the software to be built. We can define a set of object/relationship pairs that define the relevant relationships. For example,

- A bookstore orders books.
- A bookstore displays books.
- A bookstore stocks books.
- A bookstore sells books.
- A bookstore returns books.



(a) A basic connection between objects

(b) Relationships between objects

2.3.2 Relationships Cardinality and Modality:

The elements of data modeling—data objects, attributes, and relationships—provide the basis for understanding the information domain of a problem. However, additional information related to these basic elements must also be understood.

We have defined a set of objects and represented the object/relationship pairs that bind them. But a simple pair that states: **object X** *relates* to **object Y** does not provide enough information for software engineering purposes. We must understand how many occurrences of **object X** are related to how many occurrences of **object Y.** This leads to a data modeling concept called *cardinality.*

Cardinality. The data model must be capable of representing the number of occurrences objects in a given relationship. Tillmann defines the *cardinality* of an object/relationship pair in the following manner:

Cardinality is the specification of the number of occurrences of one [object] that can be related to the number of occurrences of another [object].

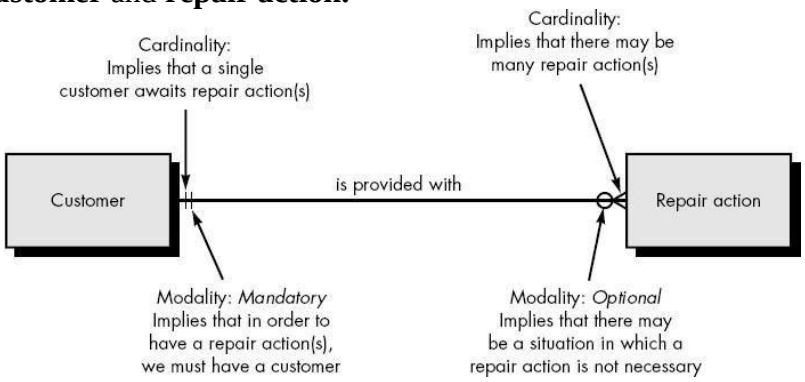
Cardinality is usually expressed as simply 'one' or 'many.'

Taking into consideration all combinations of 'one' and 'many,' two [objects] can be related as

- One-to-one (l:l)—An occurrence of [object] 'A' can relate to one and only one occurrence of [object] 'B,' and an occurrence of 'B' can relate to only one occurrence of 'A.'

• One-to-many (l:N)—One occurrence of [object] 'A' can relate to one or many occurrences of [object] 'B,' but an occurrence of 'B' can relate to only one occurrence of 'A.'
• Many-to-many (M:N)—An occurrence of [object] 'A' can relate to one or more occurrences of 'B,' while an occurrence of 'B' can relate to one or more occurrences of 'A.'
Cardinality defines "the maximum number of objects that can participate in a relationship". It does not, however, provide an indication of whether or not a particular data object must participate in the relationship. To specify this information, the data model adds modality to the object/relationship pair.

Modality. The *modality* of a relationship is 0 if there is no explicit need for the relationship to occur or the relationship is optional. The modality is 1 if an occurrence of the relationship is mandatory. To illustrate, consider software that is used by a local telephone company to process requests for field service. A customer indicates that there is a problem. If the problem is diagnosed as relatively simple, a single repair action occurs. However, if the problem is complex, multiple repair actions may be required. Figure 5 illustrates the relationship, cardinality, and modality between the data objects **customer** and **repair action.**



Cardinality and modality

Referring to the figure, a one to many cardinality relationships is established. That is, a single customer can be provided with zero or many repair actions. The symbols on the relationship connection closest to the data object rectangles indicate cardinality.

The vertical bar indicates one and the three-pronged fork indicates many.

Modality is indicated by the symbols that are further away from the data object rectangles. The second vertical bar on the left indicates that there must be a customer for a repair action to occur. The circle on the right indicates that there may be no repair action required for the type of problem reported by the customer.

2.4 Flow Oriented Modeling:

Flow models focus on the flow of data objects as they are transformed by processing functions. Derived from structured analysis, flow models use the data flow diagram, a modeling notation that depicts how input is transformed into output as data objects move through the system. Each software function that transforms data is

described by a process specification or narrative. In addition to data flow, this modeling element also depicts control flow.

Data flow oriented modeling is the most widely used analysis notation. Flow oriented modeling focuses on structured analysis and design, follows a top to down methodology and uses a graphical technique depicting information flows and the transformations that are applied as data moves from input to output.

The modeling tools that are used to build a data flow oriented model include context diagrams, data flow diagrams, entity relationship diagram, control flow diagram, state transition diagram, data dictionary, process specification and control specification.

Steps to create a data flow model

- Diagram 0: develop a context diagram.
- Decompose the Process into high level processes.
- In parallel to this, develop data flow diagrams, entity relationship diagrams and state transition diagrams.
- Define data stores which include normalization.
- Develop data dictionary.
- Finalize data flow diagrams, entity relationship diagram and state transition diagrams.
- Develop process specifications which include PDL, decision tables or trees.
- Perform transformational analysis which includes developing structure charts.

Information flow continuity must be maintained as each data flow diagram level is refined. This means that input and output at one level must be the same as input and output at a refined level.

2.4 1 Creating Data Flow Model:

Data Flow Diagram

Data Flow Diagram

A **Data Flow Diagram** (DFD) is a significant modeling technique for analyzing and constructing information processes. DFD literally means an illustration that explains the course or movement of information in a process. DFD illustrates this flow of information in a process based on the inputs and outputs. A DFD can be referred to as a Process Model.

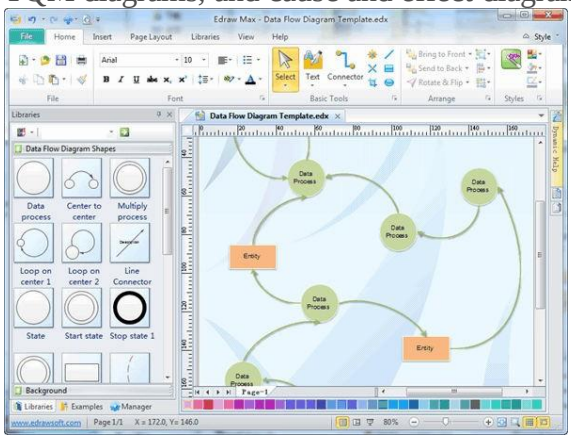
Additionally, a **DFD** can be utilized to visualize data processing or a structured design. A DFD illustrates technical or business processes with the help of the external data stored, the data flowing from a process to another, and the results.

A designer usually draws a context-level DFD showing the relationship between the entities inside and outside of a system as one single step. This basic DFD can be then disintegrated to a lower level diagram demonstrating smaller steps

exhibiting details of the system that is being modelled. Numerous levels may be required to explain a complicated system.

Data Flow Diagram Software

Data Flow Diagram software is also called as DFD software. With Edraw Max, the designers can rapidly create structured analysis diagram, information flow diagram, process-oriented diagram, data-oriented diagram and data process diagrams as well as data flowcharts, business process diagrams, work flow diagrams, value stream maps, TQM diagrams, and cause and effect diagrams.



Data flow diagram templates, symbols and samples are also provided with this dfd tools; these ready-to-use dfd templates and symbols will enable rapid designing of important complicated DFDs and process models.

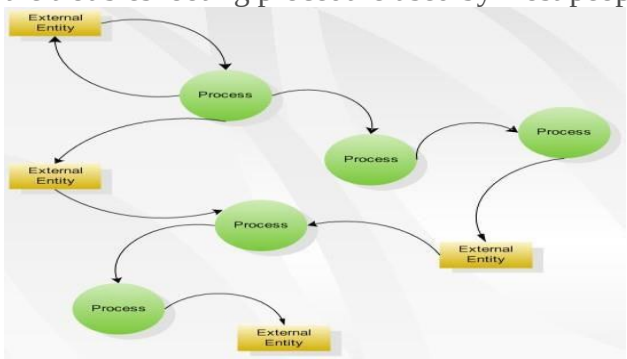
Data Flow Diagram Template

Use the data flow diagram template to describe data processes. You can use this diagram to assist in data analysis or show the flow of information for a process. The Data Flow Diagram Shapes template includes shapes for entities, states, and data processes. In general, you'll use this template to diagram the actions within a data flow, rather than the static state of a database.

Examples of Data Flow Diagrams

The following example demonstrates how to draw a data flow diagram.

Before it was eventually replaced, a copy machine suffered frequent paper jams and became a notorious troublemaker. Often, a problem could be cleared by simply opening and closing the access panel. Someone observed the situation and flowcharted the troubleshooting procedure used by most people.



When to use Data Flow Diagram

The DFD is an excellent communication tool for analysts to model processes and functional requirements. One of the primary tools of the structured analysis efforts of the 1970's it was developed and enhanced by the likes of Yourdon, McMenamin, Palmer, Gane and Sarson. It is still considered one of the best modeling techniques for eliciting and representing the processing requirements of a system.

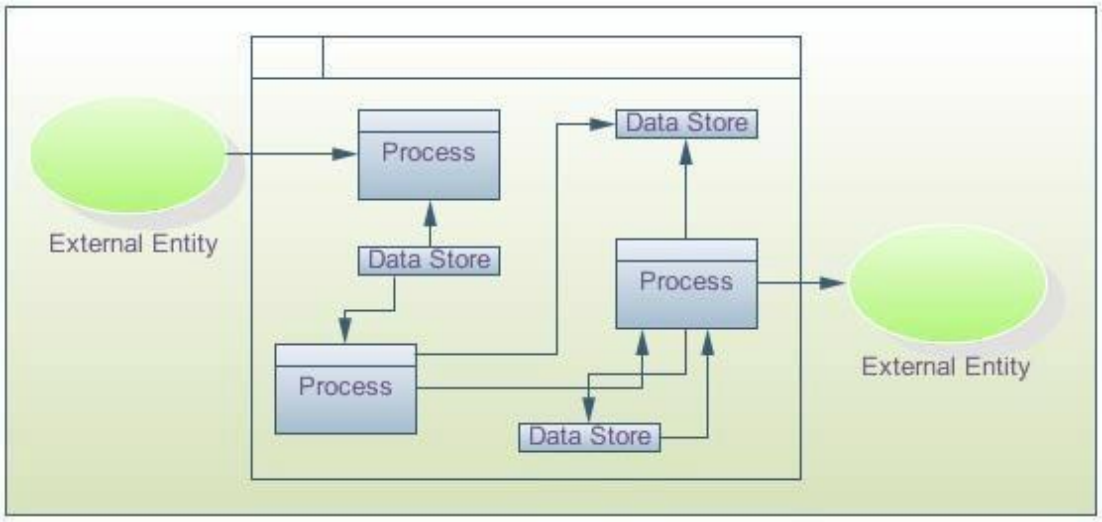
Used effectively, it is a useful and easy to understand modeling tool. It has broad application and usability across most software development projects. It is easily integrated with data modeling, workflow modeling tools, and textual specs. Together with these, it provides analysts and developers with solid models and specs. Alone, however, it has limited usability. It is simple and easy to understand by users and can be easily extended and refined with further specification into a physical version for the design and development teams.

Principle for Creating Data Flow Diagrams

Therefore, the principle for creating a DFD is that one system may be disintegrated into subsystems, which in turn can be disintegrated into subsystems at a much lower level, and so on and so forth. Every subsystem in a DFD represents a process. In this process or activity the input data is processed. Processes cannot be decomposed after reaching a certain lower level. Each process in a DFD characterises an entire system. In a DFD system, data is introduced into the system from the external environment. Once entered the data flows between processes. And then the processed data is produced as an output or a result.

Create a Data Flow Diagram

Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out to precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.



The result is a series of diagrams that represent the business activities in a way that is clear and easy to communicate. A business model comprises one or more data flow diagrams (also known as business process diagrams). Initially a context diagram is drawn, which is a simple representation of the entire system under investigation. This is followed by a level 1 diagram; which provides an overview of the major

functional areas of the business. Don't worry about the symbols at this stage, these are explained shortly. Using the context diagram together with additional information from the area of interest, the level 1 diagram can then be drawn.

The level 1 diagram identifies the major business processes at a high level and any of these processes can then be analyzed further - giving rise to a corresponding level 2 business process diagram. This process of more detailed analysis can then continue - through level 3, 4 and so on. However, most investigations will stop at level 2 and it is very unusual to go beyond a level 3 diagram.

Identifying the existing business processes, using a technique like data flow diagrams, is an essential precursor to business process re-engineering, migration to new technology, or refinement of an existing business process. However, the level of detail required will depend on the type of change being considered.

The process model is typically used in structured analysis and design methods. Also called a data flow diagram (DFD), it shows the flow of information through a system. Each process transforms inputs into outputs.

2.4 2 Creating a Control Flow Model

A **control flow diagram** (**CFD**) is a diagram to describe the control flow of a business process, process or program.

A control flow diagram can consist of a subdivision to show sequential steps, with if-then-else conditions, repetition, and/or case conditions. Suitably annotated geometrical figures are used to represent operations, data, or equipment, and arrows are used to indicate the sequential flow from one to another. There are several types of control flow diagrams, for example:

- Change control flow diagram, used in project management
- Configuration decision control flow diagram, used in configuration management
- Process control flow diagram, used in process management
- Quality control flow diagram, used in quality control.

In software and systems development control flow diagrams can be used in control flow analysis, data flow analysis, algorithm analysis, and simulation. Control and data flow analysis are most applicable for real time and data driven systems. These flow analyses transform logic and data requirements text into graphic flows which are easier to analyze than the text. PERT, state transition and transaction diagrams are examples of control flow diagrams.

Types of Control Flow Diagrams

Process Control Flow Diagram

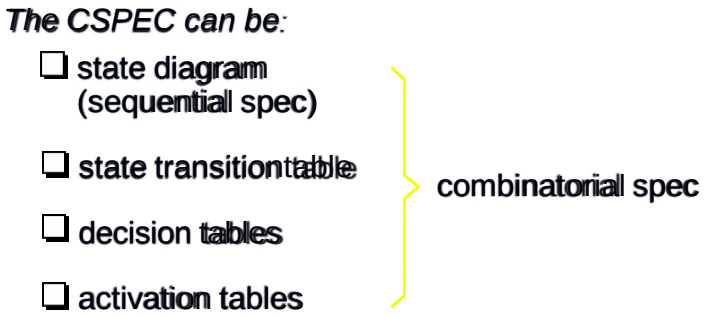
A flow diagram can be developed for the process control system for each critical activity. Process control is normally a closed cycle in which a sensor provides information to a process control software application through a communications system. The application determines if the sensor information is within the predetermined (or calculated) data parameters and constraints. The results of this comparison are fed to an actuator, which controls the critical component. This feedback may control the component electronically or may indicate the need for a manual action.

This closed-cycle process has many checks and balances to ensure that it stays safe. The investigation of how the process control can be subverted is likely to be extensive because all or part of the process control may be oral instructions to an individual monitoring the process. It may be fully computer controlled and automated, or

it may be a hybrid in which only the sensor is automated and the action requires manual intervention.

2.4.3 The Control Specification

Control Specification (CSPEC)

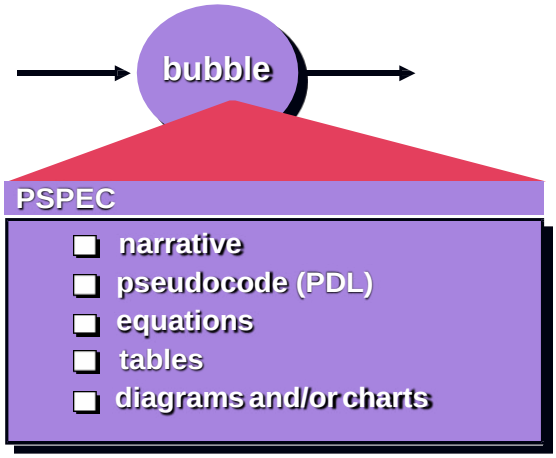


Guidelines for Building a CSPEC

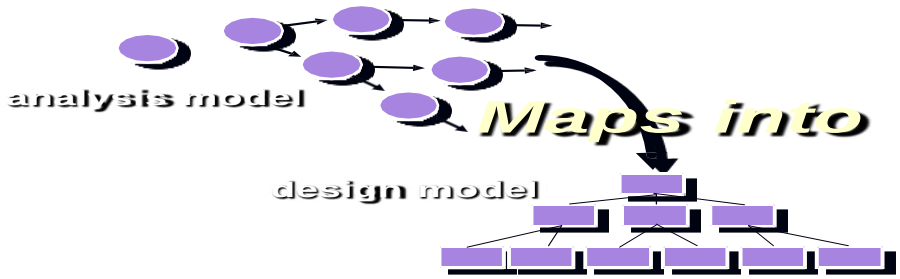
- List all sensors that are "read" by the software.
- List all interrupt conditions.
- List all "switches" that are actuated by the operator.
- List all data conditions.
- Recalling the noun-verb parse that was applied to the software statement of scope, review all "control items" as possible CSPEC inputs/outputs.
- Describe the behavior of a system by identifying its states; identify how each state is reach and defines the transitions between states.
- Focus on possible omissions ... a very common error in specifying control, e.g., ask: "Is there any other way I can get to this state or exit from it?"

2.4.4 The Process Specification

The *Process Specification* (PSPEC) is used to describe all flow model processes that appear at the final level of refinement. It is a "mini" specification for each transform at the lowest refined of a DFD.



DFDs: A Look Ahead



Control Flow Diagrams

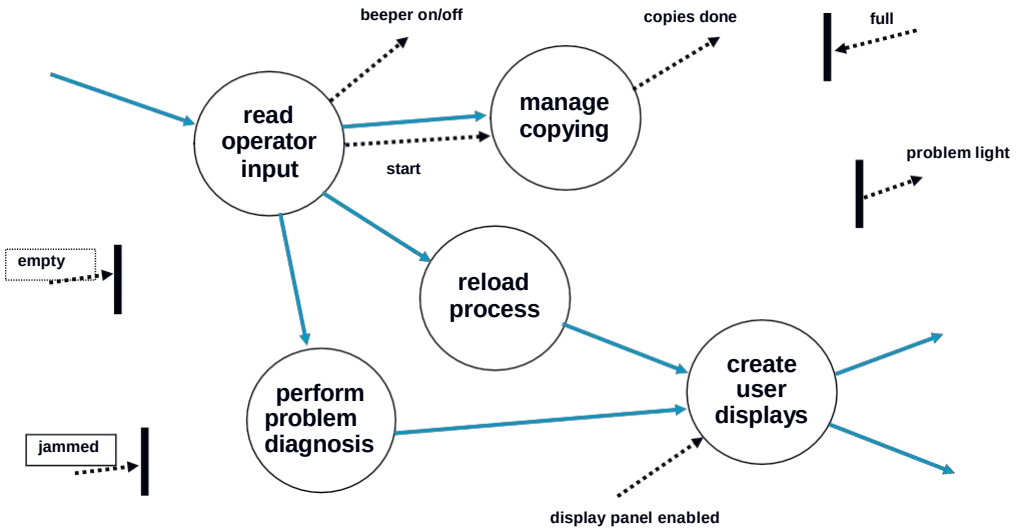
The diagram represents "events" and the processes that manage these events.
An "event" is a Boolean condition that can be ascertained by:

- Listing all sensors that are "read" by the software.
- Listing all interrupt conditions.
- Listing all "switches" that are actuated by an operator.
- Listing all data conditions.
- Recalling the noun/verb parse that was applied to the processing narrative, review all "control items" as possible CSPEC inputs/outputs.

The Control Model

- The control flow diagram is "superimposed" on the DFD and shows events that control the processes noted in the DFD.
- Control flows—events and control items—are noted by dashed arrows.
- A vertical bar implies an input to or output from a control spec (CSPEC) — a separate specification that describes how control is handled.
- A dashed arrow entering a vertical bar is an input to the CSPEC
- A dashed arrow leaving a process implies a data condition.
- A dashed arrow entering a process implies a control input read directly by the process.
- Control flows do not physically activate/deactivate the processes—this is done via the CSPEC.

Control Flow Diagram



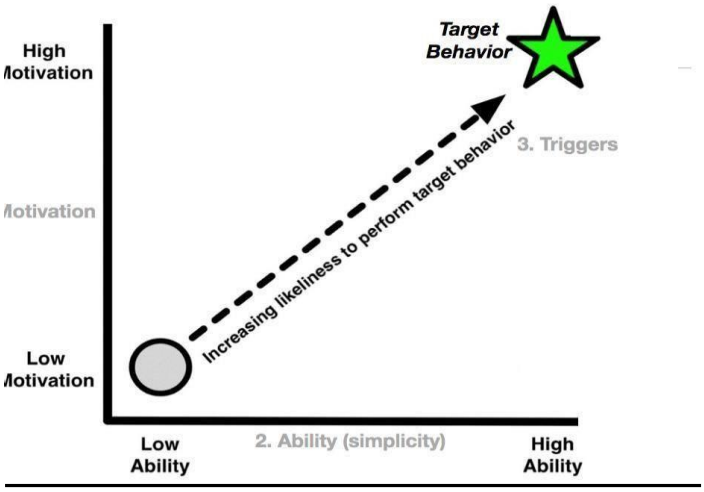
2.5 Creating a Behavioral Model:

While other analysis modeling elements provides a static view of the software, behavioral modeling depicts the dynamic behavior. The behavioral model uses input from scenario based, flow oriented and class based elements to represent the states of analysis classes and the system as a whole. To accomplish this, states are identified, the events that cause a class to make a transition from one state to another are defined, and the actions that occur as transition is accomplished are also identified.The behavioral model is an indication showing how software responds to external event. Steps to be followed are:

- All use cases are evaluated.
- Events are identified and their relations to classes are identified.
- An event occurs whenever the system and an user exchange information. An event is not the information that is exchanged but a fact that information has been exchanged**.**
- A sequence is created for use-case.
- A state diagram is built.

There are two different characterizations of states in behavioral modeling:

State of class as system performs its function and the state of the system as seen from outside. The system has states that represent specific externally observable behavior whereas a class has states that represent its behavior as the system performs its functions.



POSSIBLE QUESTIONS

1.What is meant by requirement analysis?
2.what are the types of requirement analysis?
3.what is meant by architectural requirement?
4.what is meant by functional requirement?
5.what is meant by non-functional requirement?
6.differntiate functional & non-functional requirement.
7.write a note on data flow diagram in software.

Unit II

_____ is a process of discovery, refinement, modeling, and _____ is the systematic use of proven principles, techniques, languages, and tools.	software engineering software engineering	software software analysis	software analysis software design	software design requirements engineering	software requirements engineering
Requirement engineering is conducted in a _____.	sporadic way	random way	haphazard way	systematic use of proven approaches	systematic use of proven approaches
Software requirements analysis work products must be reviewed for _____.	modeling	completeness	information processing	functional requirement	completeness
. _____ bridges the gap between system level requirement engineering and software design.	system engineering	modeling	requirements analysis	software engineering	software engineering
A _____ can be an external entity	function object	structural object	Data object	flow object	Data object
The _____ as a bridge between the systm decription and the design model	design	**analysis model**	planning	construction	analysis model
Software applications can be collectively called as _____.	data gathering	information gathering	data processing	information processing	data processing
_____ represents the individual data and control objects that constitute some larger collection of information transformed by the software.	information content	data content	data model	information model	information content
_____ represents the manner in which data and control change as each moves through a system.	information content	information flow	information structure	data structure	information flow
_____ represents the internal organization of various data and control items.	information content	information flow	information structure	data structure	information structure
Entity is a _____.	data	information	model	physical thing	physical thing
The first operational analysis principle requires an examination of the information domain and the creation of a _____.	data model	information model	data structure	information structure	data model
To transform software into information, the system performs _____.	input	processing	output	input, processing and output	input, processing and output
. To transform software into information, the system must perform _____ generic functions.	2	3	4	5	3
There are _____ types of models.	5	4	3	2	2
The horizontal partitioning of SafeHome function has _____ major functions on the first level of hierarchy.	2	3	4	5	3
The vertical partitioning of SafeHome function has _____ major functions on the first level of hierarchy.	2	3	4	5	3
A model of the software to be built is called _____.	data model	prototype	information model	software model	prototype

The _____ of software requirements presents the real world manifestation of processing functions and information structures	implementation view	essential view	partitioning view	evolutionary view	implementation view
. The essential view of the SafeHome function _____ does not concern itself with the physical form of the data that is used.	identify event	read sensor status	activate sensor	deactivate sensor	read sensor status
A prototype is the _____.	data model	information model	software model	evolution model	information model
Data objects are represented by _____	labeled arrows	bubbles	entity	label	labeled arrows
Transformations are represented by ____	labeled arrows	bubbles	entity	label	bubbles
_____ enables the software engineer to generate executable code quickly, they are ideal for rapid prototyping.	2 GT	3 GT	4 GT	5 GT	4 GT
The _____ provides a detailed description of the problem that the software must solve.	information description	software scope	function description	software description	information description
_____ is probably the most important and, ironically, the most often neglected section of software requirements specification.	behavioural description	processing narrative	overall structure	validation criteria	validation criteria
The software requirements specification includes _____.	bibliography	appendix	Bibliography and appendix	review	Bibliography and appendix
The _____ section of the specification examines the operation of the software as a consequence of external events and internally generated control characteristics.	behavioural description	representation format	specification principles	prototyping environment	behavioural description
The software requirements specification is developed as a consequence of _____.	review	analysis	prototyping	functional description	analysis
The role of the software engineer in the requirement analysis is called____	designer	**analyst**	programmer	tester	analyst
_____ is the first technical step in the software process.	requirements analysis	requirements specification	information description	information domain	requirements analysis
The close ended approach of the prototyping paradigm is called _____.	evolutionary prototyping	simply prototyping	open ended prototyping	throwaway prototyping	throwaway prototyping
analysis modeling often begins with _____	data modeling	function modeling	behavior modling	structure modeling	data modeling
A _____ can be an external entity	function object	structural object	Data object	flow object	Data object
_____ defines the properities of a data object	relationship	cardinality	Data attributes	modality	Data attributes
The description of each function required to solve the problem is presented in the _____.	functional description	behavioural description	data description	program description	functional description
One of the following is identified by eliciting information from the customer.	data description	program description	behavioural requirements	functional model	behavioural requirements
Software requirements analysis work products must be reviewed for _____.	system status	completeness	principles	paycheck	completeness

The overall role of software in a larger system is identified during the _____.	system engineering	software planning	software estimation	documentation	system engineering
The analyst finds that problems with the current manual system include _____.	inability to obtain the status of a component rapidly	record the origin of and the reason for every requirement	rank requirement	work to eliminate ambiguity	inability to obtain the status of a component rapidly
_____ is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object	modality	relationships	cardinality	Data attributes	Cardinality
_____ defines the maximum number of objects that can participate in a relationship	modality	relationships	Data attributes	cardinality	Cardinality
_____ provide an indication of whether or not a particular data object must participate in the relationship	Modality	relationships	Data attributes	cardinality	Modality
The _____ diagram takes an an input-process-output view of sysytem	use-case diagram	activity diagram	data flow diagram	ERD	data flow diagram
The level 0 DFD is called as _____ diagram	contract level diagram	context level diagram	text level diagram	zero level diagram	context level diagram
The _____ describes the behavior of the system but not the inner working of the processes	PSPEC	ASPEC	LSPEC	CSPEC	CSPEC
The _____ is used to describe all flow model processes the appear in the final level of refinement	CSPEC	ASPEC	LSPEC	PSPEC	PSPEC
All software applications collectively called _____.	packages	programs	software	data processing	data processing
The _____ model indicates how software will respond to external events	data	behavior	function	structural	behavior
The _____ represents a sequence of activities that involves actor and the system	csase tool	activity	use-case	swimlane	use-case
The _____ aids the analyst in understanding the information, function and behaviour of a system, thereby making the requirements analysis task easier and more systematic.	prototype	software	model	interface	model
_____ defines the properities of a data object	relationship	cardinality	Data attributes	modality	Data attributes
Data objects are connected to one another in different ways is called _____	modality	relationships	cardinality	Data attributes	relationships
the _____ is one method for representing the behavior of a system by depicting its state and evevts	state diagram	use case diagram	ER diagram	DFD	state diagram
When an sensor event is recognized, the _____ invokes an audible alarm attached to the system.	model	software	delay	prototype	software
The requirements for SafeHome software may be analysed by partitioning the _____ domains of the product	prototype	software	behavioural	interface	behavioural
The _____ diagram indicates how events cause transitions from object to object	sequence diagram	activity	use-case	swimlane	sequence diagram
Which one depict the software requirements from the user's point of view.	behavioral based	flow based model	class based	scenario based model	scenario based model
Which model depicts how input is transformed into output as data objects move through a system	behavioral based	flow based model	class based	scenario based model	flow based model

Unit - 2

1	The _____ as a bridge between the systm decription and the design model	design	analysis model
2	The role of the software engineer in the requirement analysis is called____	designer	analyst
3	analysis modeling often begins with _____	data modeling	function modeling
4	A _____ can be an external entity	function object	structural object
5	_____ defines the properities of a data object	relationship	cardinality
6	Data objects are connected to one another in different ways is called _____	modality	relationships
7	_____ is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object	modality	relationships
8	_____ defines the maximum number of objects that can participate in a relationship	modality	relationships
9	_____ provide an indication of whether or not a particular data object must participate in the relationship	Modality	relationships
10	The _____ diagram takes an an input-process-output view of sysytem	use-case diagram	activity diagram
11	The level 0 DFD is called as _____ diagram	contract level diagram	context level diagram
12	The _____ describes the behavior of the system but not the inner working of the processes	PSPEC	ASPEC
13	The _____ is used to describe all flow model processes the appear in the final level of refinement	CSPEC	ASPEC
14	The _____ model indicates how software will respond to external events	data	behavior
15	The _____ represents a sequence of activities that involves actor and the system	csase tool	activity
16	The _____ diagram indicates how events cause transitions from object to object	sequence diagram	activity
17	Which one depict the software requirements from the user's point of view.	behavioral based	flow based model
18	Which model depicts how input is transformed into output as data objects move through a system	behavioral based	flow based model
19			
20			

Unit 3

1	_____is a iterativeprocess through which requirements are translated into a blueprint for constructing the software	requirements gathering	coding
2	Who developeda set of software quality attributefor the software design	Barry Boehm	R.Pattis
3	Which quality attribute measure the response time, throughput and effeciency of the sysytem	Functionality	Usability

4	The quality attribute, Usability is assessed by considering the overall _____ of the system	consistency	Functionality
5	A _____ refers to a sequence of instructionsthat have a specific and limited functions	procedural abstraction	data abstraction
6	_____ represent architecture as an organized collection of programs components	process models	structural models
7	_____ models address the behavioral aspects of the program architecture	process models	structural models
8	Software is divided into separately named and addressable components is called _____	process	behavior
9	the _____ is a process of changing a software by which doesnot alter the external behavior of the code	refinement	cohesion
10	_____ is an indication of the relative functional strength of the module	refinement	cohesion
11	_____ is an indication of the relativeinterdependency among modules	cohesion	patterns
12	Refinement is a top-down design strategy which is actually a process of _____	eloboration	abstraction
13	A _____ is a named collection of data that describes a data object.	procedural abstraction	data abstraction
14	_____ implies a program control mechanism without specifying internal detail.	procedural abstraction	data abstraction
15	software architecture consider ____ levels of the design pyramid	3	2
16	Which action translates data objects into data structures	data design	component design
17	In data centered arcjitecture _____ resides at the centre of the architecture which is accessed frequently by other components	client software	data store
18	_____ represents the structure of data and program components that are required to build a computer-based syste,	architectural design	data design
19	KDD stand for	Knowledge Discovery of data manipulatio	Knowledge Discovery in database
20	the _____ classes defines all abstraction that are necessary for human computer interaction	primitive class	user interface
21	The _____ classes implement lower level business abstraction required to manage the business domain class	primitive class	user interface
22	_____ suggest that a method should send or receive messages from friend class	Law of cohesion	Law of meter
23	_____ is achieve by developing modules with single minded function and aversion of excessive interaction	refinement	refactoring
24	_____ suggest that the information contained in one module is inaccesible to othe modules	refinement	refactoring
25	Refinement is a process of _____	abstraction	eloboration
26	_____ is a process of breaking up of complex problem into a manageable piecies	refinement	refactoring
27	_____ is evaluated by measuring the frequency and severity of failure.	Usability	performance

28	iIn transform flow the information must entered and exit in _____ form	external world	internal world
29	Information flow is characterized by an single data item is called	context flow	transaction flow
30	A _____ diagram is mapped into a program structure using transform or transaction mapping	data flow	use case
31	_____ language provides a semantic and syntax for describing a software architecture	architectural description	architectural design
32	Design begins with the _____ model.	data	requirements
33	_____ focus on the design of the business or technical process that the system must accommodate.	framework models	dynamic models
34	_____ can be used to represent the functional hierarchy of a system.	framework models	dynamic models
35	_____ represent architecture as an organized collection of program components.	dynamic models	functional models
36	_____ increases the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.	framework models	dynamic models
37	_____ address the behavioural aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.	framework models	dynamic models
38	_____ is the place where quality is fostered in software engineering	model	data
39	_____ provides us with representations of software that can be assessed for quality.	design	specification
40	_____ describes a program should not have any bugs that inhibit its function	firmness	commodity
41	A program should be suitable for the purposes for which it is intended is called	firmness	commodity
42	The experience of using the program should be a pleasurable one is called	firmness	commodity
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			

planning	construction	analysis model
programmer	tester	analyst
behavior modling	structure modeling	
Data object	flow object	Data object
Data attributes	modality	Data attributes
cardinality	Data attributes	relationships
cardinality	Data attributes	Cardinality
Data attributes	cardinality	Cardinality
Data attributes	cardinality	Modality
data flow diagram	ERD	data flow diagram
text level diagram	zero level diagram	context level diagram
LSPEC	CSPEC	CSPEC
LSPEC	PSPEC	PSPEC
function	structural	behavior
use-case	swimlane	use-case
use-case	swimlane	sequence diagram
class based	scenario based model	scenario based model
class based	scenario based model	flow based model
software design	deployment	software design
M.C.Escher	Hewlett-Packard	Hewlett-Packard
Performance	Supportability	Performance

Supportability	Performance	consistency
behavior abstraction	structural abstraction	procedural abstraction
dynamic models	framework models	structural models
dynamic models	framework models	dynamic models
modules	data	Modules
patterns	refactoring	refactoring
patterns	refactoring	cohesion
coupling	functional dependency	coupling
refactoring	information hiding	eloboration
control abstraction	behavior abstraction	data abstraction
control abstraction	behavior abstraction	control abstraction

1	4	2
behavior design	functional design	data design
filter	pipes	data store
software design	behavioural design	architectural design
Knowing of database discovery process	Knowing discovery of database business	Knowledge Discovery in database
classes	domain	user interface
process classes	System class	process classes
Low of completeness	law of primitiveness	Law of meter
functional independence	information hiding	functional independence
functional independence	information hiding	information hiding
architecture	modularity	eloboration
modularity	arichiteture	modularity
supportability	reliability	reliability

top down	bottom up	external world
transform flow	contract flow	transaction flow
state diagram	activity diagram	data flow
architectural pattern	architecturaldefinition	architectural description
specification	code	requirements
process models	functional models	process models
process models	functional models	functional models
framework models	structural models	structural models
process models	functional models	framework models
process models	functional models	dynamic models
design	specification	design
data	prototype	design
delight	roman	firmness
delight	roman	commodity
delight	roman	delight

UNIT 3

SYLLABUS

Risk Management: Software Risks, Risk Identification Risk Projection and Risk Refinement, RMMM plan, **Quality Management-** Quality Concepts, Software Quality Assurance, Software Reviews, Metrics for Process and Projects

MATERIAL

A common definition of risk is an uncertain event that if it occurs, can have a positive or negative effect on a project's goals. The potential for a risk to have a positive or negative effect is an important concept. Why? Because it is natural to fall into the trap of thinking that risks have inherently negative effects. If you are also open to those risks that create positive opportunities, you can make your project smarter, streamlined and more profitable. Think of the adage – *"Accept the inevitable and turn it to your advantage."* That is what you do when you mine project risks to create opportunities.

Uncertainty is at the heart of risk. You may be unsure if an event is *likely* to occur or not. Also, you may be uncertain what its *consequences* would be *if* it did occur. Likelihood – the probability of an event occurring, and consequence – the impact or outcome of an event, are the two components that characterize the magnitude of the risk.

All risk management processes follow the same basic steps, although sometimes different jargon is used to describe these steps. Together these 5 risk management process steps combine to deliver a simple and effective risk management process.

Step 1: Identify the Risk. You and your team uncover, recognize and describe risksthat might affect your project or its outcomes. There are a number of techniques you can use to find project risks. During this step you start to prepare your Project Risk Register.

Step 2: Analyze the risk. Once risks are identified you determine the likelihood and consequence of each risk. You develop an understanding of the nature of the risk and its potential to affect project goals and objectives. This information is also input to your Project Risk Register.

Step 3: Evaluate or Rank the Risk. You evaluate or rank the risk by determining the risk magnitude, which is the combination of likelihood and consequence. You make decisions about whether the risk is acceptable or whether it is serious enough to warrant treatment. These risk rankings are also added to your Project Risk Register.

Step 4: Treat the Risk. This is also referred to as Risk Response Planning. During this step you assess your highest ranked risks and set out a plan to treat or modify these risks to achieve acceptable risk levels. How can you minimize the probability of the negative risks as well as enhancing the opportunities? You create risk mitigation strategies, preventive plans and contingency plans in this step. And you add the risk treatment measures for the highest ranking or most serious risks to your Project Risk Register.

Step 5: Monitor and Review the risk. This is the step where you take your Project Risk Register and use it to monitor, track and review risks.

Risk is about uncertainty. If you put a framework around that uncertainty, then you effectively de-risk your project. And that means you can move much more confidently to achieve your project goals. By identifying and managing a comprehensive list of project risks, unpleasant surprises and barriers can be reduced and golden opportunities discovered. The risk management process also helps to resolve problems when they occur, because those problems have been envisaged, and plans to treat them have already been developed and agreed. You avoid impulsive

reactions and going into "fire-fighting" mode to rectify problems that could have been anticipated. This makes for happier, less stressed project teams and stakeholders. The end result is that you minimize the impacts of project threats and capture the opportunities that occur.

RISK IDENTIFICATION

Risk identification is a process for identifying and recording potential project risks that can affect the project delivery. This step is crucial for efficient risk management throughout the project. The outputs of the risk identification are used as an input for risk analysis, and they reduce a project manager's uncertainty. It is an iterative process that needs to be continuously repeated throughout the duration of a project. The process needs to be rigorous to make sure that all possible risks are identified.

An effective risk identification process should include the following steps:

1. **Creating a systematic process** - The risk identification process should begin with project objectives and success factors.
2. **Gathering information from various sources** - Reliable and high quality information is essential for effective risk management.
3. **Applying risk identification tools and techniques** - The choice of the best suitable techniques will depend on the types of risks and activities, as well as organizational maturity.
4. **Documenting the risks** - Identified risks should be documented in a risk register and a risk breakdown structure, along with its causes and consequences.
5. **Documenting the risk identification process** - To improve and ease the risk identification process for future projects, the approach, participants, and scope of the process should be recorded.
6. **Assessing the process' effectiveness** - To improve it for future use, the effectiveness of the chosen process should be critically assessed after the project is completed.

RISK PROJECTION

Risk projection, also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. The project planner, along with other managers and technical staff, performs four risk projection activities: (1) establish a scale that reflects the perceived likelihood of a risk, (2) delineate the consequences of the risk, (3) estimate the impact of the risk on the project and the product, and (4)note the overall accuracy of the risk projection so that there will be no misunderstandings.

1. Define the risk referent levels for the project.
2. Attempt to develop a relationship between each (ri, li, xi) and each of the referent levels.
3. Predict the set of referent points that define a region of termination, bounded by a curve or areas of uncertainty.
4. Try to predict how compound combinations of risks will affect a referent level.

RISK REFINEMENT

This general condition can be refined in the following manner:

Subcondition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.

Subcondition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Subcondition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

RMMM PLAN

A risk management strategy can be included in the software project plan or the risk management steps can be organized into a separate Risk Mitigation, Monitoring and Management Plan. The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.

Some software teams do not develop a formal RMMM document. Rather, each risk is documented individually using a risk information sheet . In most cases, the RIS is maintained using a database system, so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.

Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. As we have already discussed, risk mitigation is a problem avoidance activity. Risk monitoring is a project tracking activity with three primary objectives:

(1) to assess whether predicted risks do, in fact, occur;

(2) to ensure that risk aversion steps defined for the risk are being properly applied; and

(3) to collect information that can be used for future risk analysis.

In many cases, the problems that occur during a project can be traced to more than one risk. Another job of risk monitoring is to attempt to allocate origin (what risk(s) caused which problemsthroughout the project).

QUALITY MANAGEMENT

The quality of software has improved significantly over the past two decades. One reason for this is that companies have used new technologies in their software development process such as object-oriented development, CASE tools, etc. In addition, a growing importance of software quality management and the adoption of quality management techniques from manufacturing can be observed. However, software quality significantly differs from the concept of quality generally used in manufacturing mainly for the next reasons [**1**]:

1. The software specification should reflect the characteristics of the product that the customer wants. However, the development organization may also have requirements such as maintainability that are not included in the specification.
2. Certain software quality attributes such as maintainability, usability, reliability cannot be exactly specified and measured.
3. At the early stages of software process it is very difficult to define a complete software specification. Therefore, although software may conform to its specification, users don't meet their quality expectations.

Software quality management is split into three main activities:

1. Quality assurance. The development of a framework of organizational procedures and standards that lead to high quality software.
2. Quality planning. The selection of appropriate procedures and standards from this framework and adapt for a specific software project.
3. Quality control. Definition of processes ensuring that software development follows the quality procedures and standards.

Quality management provides an independent check on the software and software development process. It ensures that project deliverables are consistent with organizational standards and goals.

12.1. Process and product quality

It is general, that the quality of the development process directly affects the quality of delivered products. The quality of the product can be measured and the process is improved until the proper quality level is achieved. Figure 12.1. illustrates the process of quality assessment based on this approach.

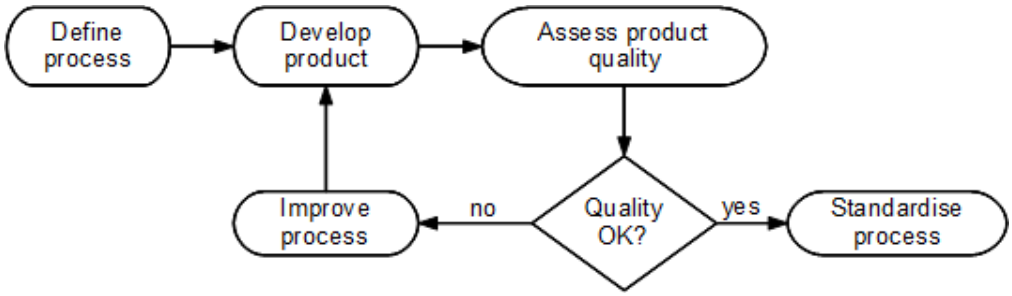


Figure 12.1. Process based quality assessment.

In manufacturing systems there is a clear relationship between production process and product quality. However, quality of software is highly influenced by the experience of software engineers. In addition, it is difficult to measure software quality attributes, such as maintainability, reliability, usability, etc., and to tell how process characteristics influence these attributes. However, experience has shown that process quality has a significant influence on the quality of the software.

Process quality management includes the following activities:

1. Defining process standards.
2. Monitoring the development process.
3. Reporting the software.

12.2. Quality assurance and standards

12.2.1. ISO

12.2.2. Documentation standards

Quality assurance is the process of defining how software quality can be achieved and how the development organization knows that the software has the required level of quality. The main activity of the quality assurance process is the selection and definition of standards that are applied to the software development process or software product. There are two main types of standards. The product standards are applied to the software product, i.e. output of the software process. The process standards define the processes that should be followed during software development. The software standards are based on best practices and they provide a framework for implementing the quality assurance process.

The development of software engineering project standards is a difficult and time consuming process. National and international bodies such as ANSI and the IEEE develop standards that can be applied to software development projects. Organizational standards, developed by quality assurance teams, should be based on these national and international standards. Table 12.1. shows examples of product and process standards.

Table 12.1. Examples of product and process standards.

Product standards
Requirements document structure
Method header format
Java programming style
Change request form

12.2.1. ISO

ISO 9000 is an international set of standards that can be used in the development of a quality management system in all industries. ISO 9000 standards can be applied to a range of organizations from manufacturing to service industries. ISO 9001 is the most general of these standards. It can be applied to organizations that design, develop and maintain products and develop their own quality processes. A supporting document (ISO 9000-3) interprets ISO 9001 for software development.

The ISO 9001 standard isn't specific to software development but includes general principles that can be applied to software development projects. The ISO 9001 standard describes various aspects of the quality process and defines the organizational standards and procedures that a company should define and follow during product development. These standards and procedures are documented in an organizational quality manual.

The ISO 9001 standard does not define the quality processes that should be used in the development process. Organizations can develop own quality processes and they can still be ISO 9000 compliant companies. The ISO 9000 standard only requires the definition of processes to be used in a company and it is not concerned with ensuring that these processes provide best practices and high quality of products. Therefore, the ISO 9000 certification doesn't means exactly that the quality of the software produced by an ISO 9000 certified companies will be better than that software from an uncertified company.

12.2.2. Documentation standards

Documentation standards in a software project are important because documents can represent the software and the software process. Standardized documents have a consistent appearance,

structure and quality, and should therefore be easier to read and understand. There are three types of documentation standards:

1. Documentation process standards. These standards define the process that should be followed for document production.
2. Document standards. These standards describe the structure and presentation of documents.
3. Documents interchange standards. These standards ensure that all electronic copies of documents are compatible.

12.3. Quality planning

Quality planning is the process of developing a quality plan for a project. The quality plan defines the quality requirements of software and describes how these are to be assessed. The quality plan selects those organizational standards that are appropriate to a particular product and development process. Quality plan has the following parts:

1. Introduction of product.
2. Product plans.
3. Process descriptions.
4. Quality goals.
5. Risks and risk management.

The quality plan defines the most important quality attributes for the software and includes a definition of the quality assessment process. Table 12.2. shows generally used software quality attributes that can be considered during the quality planning process.

Table 12.2. Software quality attributes.

Safety	Understandability
Security	Testability
Reliability	Adaptability
Resilience	Modularity
Robustness	Complexity
Maintainability	

12.4. Quality control

12.4.1. Quality reviews

Quality control provides monitoring the software development process to ensure that quality assurance procedures and standards are being followed. The deliverables from the software development process are checked against the defined project standards in the quality control process. The quality of software project deliverables can be checked by regular quality reviews and/or automated software assessment. Quality reviews are performed by a group of people. They review the software and software process in order to check that the project standards have been followed and that software and documents conform to these standards. Automated software assessment processes the software by a program that compares it to the standards applied to the development project.

12.4.1. Quality reviews

Quality reviews are the most widely used method of validating the quality of a process or product. They involve a group of people examining part or all of a software process, system, or its associated documentation to discover potential problems. The conclusions of the review are

formally recorded and passed to the author for correcting the discovered problems. Table 12.3. describes several types of review, including quality reviews.

Table 12.3. Types of review.

Review type
Design or program inspections
Progress reviews
Quality reviews

12.5. Software measurement and metrics

12.5.1. The measurement process

12.5.2. Product metrics

Software measurement provides a numeric value for some quality attribute of a software product or a software process. Comparison of these numerical values to each other or to standards draws conclusions about the quality of software or software processes. Software product measurements can be used to make general predictions about a software system and identify anomalous software components.

Software metric is a measurement that relates to any quality attributes of the software system or process. It is often impossible to measure the external software quality attributes, such as maintainability, understandability, etc., directly. In such cases, the external attribute is related to some internal attribute assuming a relationship between them and the internal attribute is measured to predict the external software characteristic. Three conditions must be hold in this case:

1. The internal attribute must be measured accurately.
2. A relationship must exist between what we can measure and the external behavioural attribute.
3. This relationship has to be well understood, has been validated and can be expressed in terms of a mathematical formula.

12.5.1. The measurement process

A software measurement process as a part of the quality control process is shown in Figure 12.2. The steps of measurement process are the followings:

1. Select measurements to be made. Selection of measurements that are relevant to answer the questions to quality assessment.
2. Select components to be assessed. Selection of software components to be measured.
3. Measure component characteristics. The selected components are measured and the associated software metric values computed.
4. Identify anomalous measurements. If any metric exhibit high or low values it means that component has problems.
5. Analyze anomalous components. If anomalous values for particular metrics have been identified these components have to be examined to decide whether the anomalous metric values mean that the quality of the component is compromised.

Generally each of the components of the system is analyzed separately. Anomalous measurements identify components that may have quality problems.

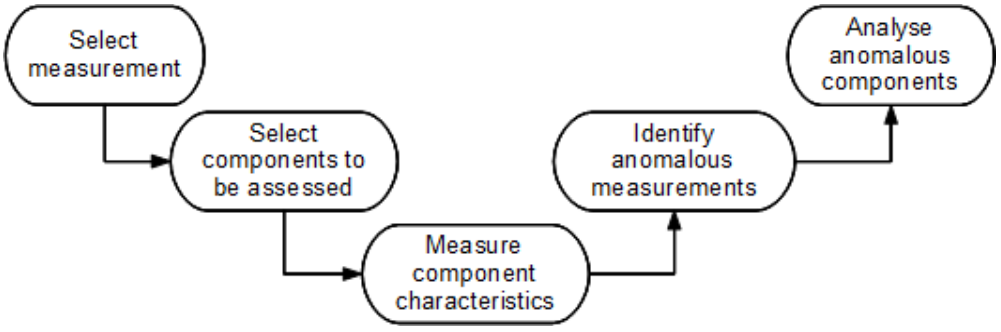


Figure 12.2. The software measurement process.

12.5.2. Product metrics

The software characteristics that can be easily measured such as size do not have a clear and consistent relationship with quality attributes such as understandability and maintainability. Product metrics has two classes:

1. Dynamic metrics. These metrics (for example execution time) are measured during the execution of a program.
2. Static metrics. Static metrics are based on measurements made of representations of the system such as the design, program or documentation.

Dynamic metrics can be related to the efficiency and the reliability of a program. Static metrics such as code size are related to software quality attributes such as complexity, understandability, maintainability, etc.

POSSIBLE QUESTIONS

1. Write note on risk identification.
2. what is meant by software risk?
3. write note on risk projection.
4. write on risk refinement.
5. define software quality.
6. write note on software quality assurance.
7. discuss the metrics of software.

UNIT III

		2	3	4	5
121	There are _____ major phases to any design process				
122	Diversification is the _____ of a repertoire of alternatives.	component	solution	acquisition	knowledge
123	During _____, the designer chooses and combines appropriate elements from the repertoire to meet the design objectives.	diversification	convergence	elimination	creation
124	_____ and _____ combine intuition and judgement based on experience in building similar entities.	elimination, converge	creation, converg	acquisition, creatic	diversification and convergence
125	_____ can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria.	design	analysis	principles	testing
126	The _____ must implement all of the explicit requirements contained in the analysis model	principles	testing	design	component
127	A _____ should exhibit an architectural structure that has been created using recognizable design patterns.	principles	testing	component	design
128	A _____ is composed of components that exhibit good design characteristics.	principles	testing	component	design
129	A _____ can be implemented in an evolutionary fashion thereby facilitating implementation and testing.	principles	testing	component	design
130	A _____ should be modular that is the software should be logically partitioned into elements that perform specific functions and sub functions.	design	principles	component	testing
131	A _____ should contain distinct representations of data, architecture, interfaces, and components.	design	principles	component	testing
132	A _____ should lead to data structures that are appropriate for the objects to be implemented and are drawn from recognizable data patterns.	design	principles	component	testing
133	. A _____ should lead to interfaces that reduce the complexity of connections between modules and with the external environment.	design	principles	component	testing
134	A _____ should be derived using a repeatable method that is driven by information obtained during software requirements analysis	principles	component	design	testing

135	The software _____ process encourages good design through the application of fundamental design principles, systematic methodology and thorough review.	principles	component	design	testing
136	The _____ must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.	principles	component	design	testing
137	The _____ should provide a complete picture of the software addressing the data, functional and behavioral domains from an implementation perspective.	principles	component	design	testing
138	The evolution of software _____ is a continuing process that has spanned the past four decades.	principles	component	design	testing
139	Procedural aspects of design definition evolved into a philosophy called _____.	top down programmir	bottom up progra	structured progran	object oriented progra
140	The design process should not suffer from _____.	analysis	tunnel vision	conceptual errors	integrity
141	The design should be _____ to the analysis model.	consistent	related	traceable	relevant
142	The design should not _____ the wheel.	minimize	maximize	integrate	reinvent
143	The design should _____ the intellectual distance	maximize	minimize	integrate	analyse
144	. The _____ is represented at a high level of abstraction	specification	analysis	quality	design specification
145	The design should exhibit _____ and integration.	uniformity	analysis	quality	review
146	The design should be _____ to accommodate change.	reviewed	analysed	assessed	structured
147	The design should be _____ to degrade gently, even when aberrant data, events, or operating conditions are encountered.	reviewed	analysed	assessed	structured
148	Design is not _____, coding is not design	coding	analysis	review	event
149	Design is not coding, _____ is not design.	coding	analysis	review	event
150	The design should be _____ for quality as it is being created not after the fact.	reviewed	assessed	structured	integrated
151	The design should be _____ to minimize conceptual errors.	reviewed	assessed	structured	integrated
152	Software design is both a _____ and a model.	model	process	data	function
153	_____ is the only way that we can accurately translate a customer's requirements into a finished software product or system.	specification	design	data	prototype
154	The design _____ is the equivalent of an architect's plan for a house.	analysis	process	model	function
155	At the highest level of _____, a solution is stated in broad terms, using the language of the problem environment.	refinement	modularity	abstraction	continuity
156	. A _____ is a named sequence of instructions that has a specific and limited function.	procedural abstraction	data abstraction	control abstraction	all of the above
157	A _____ is a named collection of data that describes a data object.	procedural abstraction	data abstraction	control abstraction	all of the above

158	_____ implies a program control mechanism without specifying internal detail.	procedural abstraction	data abstraction	control abstraction	all of the above
159	_____ is used to coordinate activities in an operating system.	synchronization semaphore	control abstraction	data abstraction	procedural abstraction
160	_____ is a top down design strategy originally proposed by Niklaus Wirth.	stepwise refinement	control abstraction	data abstraction	procedural abstraction
161	The designer's goal is to produce a model or representation of a _____ that will later be built	component	entity	data	raw material
162	The second phase of any design process is the gradual _____ of all but one particular configuration of components, and thus the creation of the final product.	acquisition	addition	elimination	creation
163	Design begins with the _____ model.	data	requirements	specification	code
164	Software design methodologies lack the _____ that are normally associated with more classical engineering design disciplines.	depth	flexibility	quantitative nature	all of the above
165	Software requirements, manifested by the _____ models, feed the design task.	data	functional	behavioral	all of the above
166	_____ is the place where quality is fostered in software engineering	model	data	design	specification
167	_____ provides us with representations of software that can be assessed for quality.	design	specification	data	prototype
168	Procedural aspects of design definition evolved into a philosophy called _____.	procedural programming	object oriented programming	structured programming	all of the above
169	Meyer defines _____ criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system.		2	3	4 5
170	. If a design method provides a systematic mechanism for decomposing the problem into sub problems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution. This is called _____.	modular decomposability	modular composability	modular understandability	modular continuity
171	If a design method enables existing (reusable) design components to be assembled into a new system, it will yield a modular solution that does not reinvent the wheel. This is called _____.	modular decomposability	modular composability	modular understandability	modular continuity
172	If a module can be understood as a stand alone unit (without reference to other modules), it will be easier to build and easier to change. This is called _____.	modular decomposability	modular composability	modular understandability	modular continuity

173	If small changes to the system requirements result in changes to individual modules, rather than system wide changes, the impact of change-induced side effects will be minimized. This is called _____.	modular decomposability	modular composability	modular understandability	modular continuity
174	If an aberrant condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized. This is called _____.	modular protection	modular composability	modular understandability	modular continuity
175	The aspect of the architectural design representation defines the components of a system and the manner in which those components are packaged and interact with one another. This property is called _____.	extra functional property	structural property	families of related systems	none of the above
176	_____ represent architecture as an organized collection of program components.	dynamic models	functional models	framework models	structural models
177	_____ increases the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.	framework models	dynamic models	process models	functional models
178	_____ address the behavioural aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.	framework models	dynamic models	process models	functional models
179	_____ focus on the design of the business or technical process that the system must accommodate.	framework models	dynamic models	process models	functional models
180	_____ can be used to represent the functional hierarchy of a system.	framework models	dynamic models	process models	functional models

component

diversification

acquisition, creation

design

component

design

principles

testing

design

component

principles

design

principles

component

principles

design

principles

structured programming
analysis
related
minimize
integrate
specification
uniformity
reviewed

structured
coding
coding

assessed
integrated
process
prototype

function

modularity
procedural
abstraction

data abstraction

control
abstraction
synchronization
semaphore
stepwise
refinement

component

elimination
requirements

all of the above

all of the above

design

design
structured
programming

modular
decomposability

modular
composability

modular
understandability

modular
continuity

modular
protection

structural
property

structural models

framework
models

dynamic models

process models

functional
models

Unit - 2

1	The _____ as a bridge between the systm decription and the design model	design	analysis model
2	The role of the software engineer in the requirement analysis is called____	designer	analyst
3	analysis modeling often begins with _____	data modeling	function modeling
4	A _____ can be an external entity	function object	structural object
5	_____ defines the properities of a data object	relationship	cardinality
6	Data objects are connected to one another in different ways is called _____	modality	relationships
7	_____ is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object	modality	relationships
8	_____ defines the maximum number of objects that can participate in a relationship	modality	relationships
9	_____ provide an indication of whether or not a particular data object must participate in the relationship	Modality	relationships
10	The _____ diagram takes an an input-process-output view of sysytem	use-case diagram	activity diagram
11	The level 0 DFD is called as _____ diagram	contract level diagram	context level diagram
12	The _____ describes the behavior of the system but not the inner working of the processes	PSPEC	ASPEC
13	The _____ is used to describe all flow model processes the appear in the final level of refinement	CSPEC	ASPEC
14	The _____ model indicates how software will respond to external events	data	behavior
15	The _____ represents a sequence of activities that involves actor and the system	csase tool	activity
16	The _____ diagram indicates how events cause transitions from object to object	sequence diagram	activity
17	Which one depict the software requirements from the user's point of view.	behavioral based	flow based model
18	Which model depicts how input is transformed into output as data objects move through a system	behavioral based	flow based model
19			
20			

Unit 3

1	_____is a iterativeprocess through which requirements are translated into a blueprint for constructing the software	requirements gathering	coding
2	Who developeda set of software quality attributefor the software design	Barry Boehm	R.Pattis
3	Which quality attribute measure the response time, throughput and effeciency of the sysytem	Functionality	Usability

4	The quality attribute, Usability is assessed by considering the overall _____ of the system	consistency	Functionality
5	A _____ refers to a sequence of instructionsthat have a specific and limited functions	procedural abstraction	data abstraction
6	_____ represent architecture as an organized collection of programs components	process models	structural models
7	_____ models address the behavioral aspects of the program architecture	process models	structural models
8	Software is divided into separately named and addressable components is called _____	process	behavior
9	the _____ is a process of changing a software by which doesnot alter the external behavior of the code	refinement	cohesion
10	_____ is an indication of the relative functional strength of the module	refinement	cohesion
11	_____ is an indication of the relativeinterdependency among modules	cohesion	patterns
12	Refinement is a top-down design strategy which is actually a process of _____	eloboration	abstraction
13	A _____ is a named collection of data that describes a data object.	procedural abstraction	data abstraction
14	_____ implies a program control mechanism without specifying internal detail.	procedural abstraction	data abstraction
15	software architecture consider ____ levels of the design pyramid	3	2
16	Which action translates data objects into data structures	data design	component design
17	In data centered arcjitecture _____ resides at the centre of the architecture which is accessed frequently by other components	client software	data store
18	_____ represents the structure of data and program components that are required to build a computer-based syste,	architectural design	data design
19	KDD stand for	Knowledge Discovery of data manipulatio	Knowledge Discovery in database
20	the _____ classes defines all abstraction that are necessary for human computer interaction	primitive class	user interface
21	The _____ classes implement lower level business abstraction required to manage the business domain class	primitive class	user interface
22	_____ suggest that a method should send or receive messages from friend class	Law of cohesion	Law of meter
23	_____ is achieve by developing modules with single minded function and aversion of excessive interaction	refinement	refactoring
24	_____ suggest that the information contained in one module is inaccesible to othe modules	refinement	refactoring
25	Refinement is a process of _____	abstraction	eloboration
26	_____ is a process of breaking up of complex problem into a manageable piecies	refinement	refactoring
27	_____ is evaluated by measuring the frequency and severity of failure.	Usability	performance

28	iIn transform flow the information must entered and exit in _____ form	external world	internal world
29	Information flow is characterized by an single data item is called	context flow	transaction flow
30	A _____ diagram is mapped into a program structure using transform or transaction mapping	data flow	use case
31	_____ language provides a semantic and syntax for describing a software architecture	architectural description	architectural design
32	Design begins with the _____ model.	data	requirements
33	_____ focus on the design of the business or technical process that the system must accommodate.	framework models	dynamic models
34	_____ can be used to represent the functional hierarchy of a system.	framework models	dynamic models
35	_____ represent architecture as an organized collection of program components.	dynamic models	functional models
36	_____ increases the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.	framework models	dynamic models
37	_____ address the behavioural aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.	framework models	dynamic models
38	_____ is the place where quality is fostered in software engineering	model	data
39	_____ provides us with representations of software that can be assessed for quality.	design	specification
40	_____ describes a program should not have any bugs that inhibit its function	firmness	commodity
41	A program should be suitable for the purposes for which it is intended is called	firmness	commodity
42	The experience of using the program should be a pleasurable one is called	firmness	commodity
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			

planning	construction	analysis model
programmer	tester	analyst
behavior modling	structure modeling	
Data object	flow object	Data object
Data attributes	modality	Data attributes
cardinality	Data attributes	relationships
cardinality	Data attributes	Cardinality
Data attributes	cardinality	Cardinality
Data attributes	cardinality	Modality
data flow diagram	ERD	data flow diagram
text level diagram	zero level diagram	context level diagram
LSPEC	CSPEC	CSPEC
LSPEC	PSPEC	PSPEC
function	structural	behavior
use-case	swimlane	use-case
use-case	swimlane	sequence diagram
class based	scenario based model	scenario based model
class based	scenario based model	flow based model
software design	deployment	software design
M.C.Escher	Hewlett-Packard	Hewlett-Packard
Performance	Supportability	Performance

Supportability	Performance	consistency
behavior abstraction	structural abstraction	procedural abstraction
dynamic models	framework models	structural models
dynamic models	framework models	dynamic models
modules	data	Modules
patterns	refactoring	refactoring
patterns	refactoring	cohesion
coupling	functional dependency	coupling
refactoring	information hiding	eloboration
control abstraction	behavior abstraction	data abstraction
control abstraction	behavior abstraction	control abstraction

1	4	2
behavior design	functional design	data design
filter	pipes	data store
software design	behavioural design	architectural design
Knowing of database discovery process	Knowing discovery of database business	Knowledge Discovery in database
classes	domain	user interface
process classes	System class	process classes
Low of completeness	law of primitiveness	Law of meter
functional independence	information hiding	functional independence
functional independence	information hiding	information hiding
architecture	modularity	eloboration
modularity	arichiteture	modularity
supportability	reliability	reliability

top down	bottom up	external world
transform flow	contract flow	transaction flow
state diagram	activity diagram	data flow
architectural pattern	architecturaldefinition	architectural description
specification	code	requirements
process models	functional models	process models
process models	functional models	functional models
framework models	structural models	structural models
process models	functional models	framework models
process models	functional models	dynamic models
design	specification	design
data	prototype	design
delight	roman	firmness
delight	roman	commodity
delight	roman	delight

UNIT 4
SYLLABUS

Design Engineering-Design Concepts, Architectural Design Elements, Software Architecture, Data Design at the Architectural Level and Component Level, Mapping of Data Flow into Software Architecture, Modeling Component Level Design

MATERIAL

4.1 Design Engineering:

The **engineering design process** is a formulation of a plan or scheme to assist an engineer in creating a product. The engineering design is defined as:

The process of devising a system, component, or process to meet desired needs. It is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation.

The engineering design process is a multi-step process including the research, conceptualization, feasibility assessment, establishing design requirements, preliminary design, detailed design, production planning and tool design, and finally production.

The sections to follow are not necessarily steps in the engineering design process, for some tasks are completed at the same time as other tasks. This is just a general summary of each step of the engineering design process.

Research

A significant amount of time is spent on research, locating, applying, and transferring information .Consideration should be given to the existing applicable literature, problems and successes associated with existing solutions, costs, and marketplace needs.

The source of information should be relevant, including existing solutions. Reverse engineering can be an effective technique if other solutions are available on the market. Other sources of information include the Internet, local libraries, available government documents, personal organizations, trade journals, vendor catalogs and individual experts available .

3.1.1 Design with the Context of Software Engineering:

Feasibility Assessment

The purpose of a feasibility assessment is to determine whether the engineer's project can proceed into the design phase. This is based on two criteria: the project needs to be based on an achievable idea, and it needs to be within cost constraints. It is of utmost importance to have an engineer with experience and good judgment to be involved in this portion of the feasibility study, for they know whether the engineer's project is possible or not.

4.1.2 Design Process and Design Quality

Preliminary Design

The preliminary design bridges the gap between the design concept and the detailed design phase. The preliminary design phase is also called *embodiment design*. In this task, the overall system configuration is defined, and schematics, diagrams, and layouts of the project will provide early project configuration. During detailed design and optimization, the parameters of the part being created will change, but the preliminary design focuses on creating the general framework to build the project on.

Detailed Design

The detailed design portion of the engineering design process is the task where the engineer can completely describe a product through solid modeling and drawings. Some specifications include:

- Operating parameters
- Operating and no operating environmental stimuli
- Test requirements
- External dimensions
- Maintenance and testability provisions
- Materials requirements
- Reliability requirements
- External surface treatment
- Design life
- Packaging requirements
- External marking

The advancement of computer-aided design, or CAD, programs have made the detailed design phase more efficient. This is because a CAD program can provide optimization, where it can reduce volume without hindering the part's quality. It can also calculate stress and displacement using the finite element method to determine stresses throughout the part. It is the engineer's responsibility to determine whether these stresses and displacements are allowable, so the part is safe.

Production Planning and Tool Design

The production planning and tool design is nothing more than planning how to mass produce the project and which tools should be used in the manufacturing of the part. Tasks to complete in this step include selecting the material, selection of the production processes, determination of the sequence of operations, and selection of tools, such as jigs, fixtures, and tooling. This task also involves testing a working prototype to ensure the created part meets qualification standards.

Production

With the completion of qualification testing and prototype testing, the engineering design process is finalized. The part must now be manufactured, and the machines must be inspected regularly to make sure that they do not break down and slow production

As a software developer the similarities between how we build and develop the software and how architects design buildings has always struck me. In this blow, I'd like to talk about how the architecture design concept of software engineering.

4.1.3 Design Concepts:

- **Abstraction** – allows designers to focus on solving a problem without being concerned about irrelevant lower level details (*procedural abstraction* - named sequence of events and *data abstraction* – named collection of data objects)
- **Software Architecture** – overall structure of the software components and the ways in which that structure provides conceptual integrity for a system
    - Structural models – architecture as organized collection of components
    - Framework models – attempt to identify repeatable architectural patterns
    - Dynamic models – indicate how program structure changes as a function of external events
    - Process models – focus on the design of the business or technical process that system must accommodate
    - Functional models – used to represent system functional hierarchy
- **Design Patterns** – description of a design structure that solves a particular design problem within a specific context and its impact when applied
- **Separation of concerns** – any complex problem is solvable by subdividing it into pieces that can be solved independently
- **Modularity** - the degree to which software can be understood by examining its components independently of one another
- **Information Hiding** – information (data and procedure) contained within a module is inaccessible to modules that have no need for such information
- **Functional Independence** – achieved by developing modules with single-minded purpose and an aversion to excessive interaction with other models
    - Cohesion - qualitative indication of the degree to which a module focuses on just one thing
    - Coupling - qualitative indication of the degree to which a module is connected to other modules and to the outside world
- **Refinement** – process of elaboration where the designer provides successively more detail for each design component
- **Aspects** – a representation of a cross-cutting concern that must be accommodated as refinement and modularization occur
- **Refactoring** – process of changing a software system in such a way internal structure is improved without altering the external behavior or code design

4.2 Creating an Architectural Design:

What is Software Architecture?

Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman derived and refined a definition of architecture based on work by Mary Shaw and David Garlan (Shaw and Garlan 1996). Their definition is:

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

Why is Architecture Important?

Like any other complex structure, software must be built on a solid foundation. Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk. Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements. The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.

Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals. For each of these areas, you should outline key scenarios and identify important quality attributes (for example, reliability or scalability) and key areas of satisfaction and dissatisfaction. Where possible, develop and consider metrics that measure success in each of these areas.

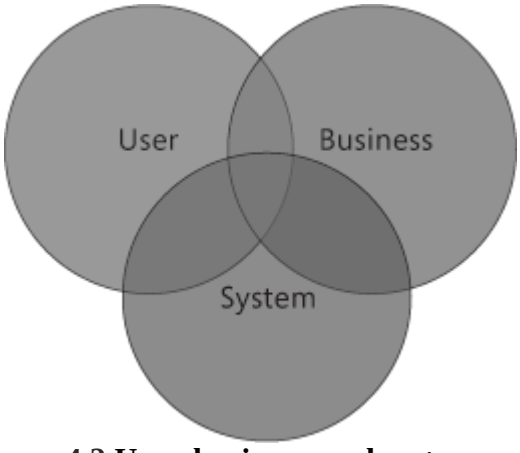


Figure 4.2 User, business, and system goals

Tradeoffs are likely, and a balance must often be found between competing requirements across these three areas. For example, the overall user experience of the solution is very often a function of the business and the IT infrastructure, and changes in one or the other can significantly affect the resulting user experience. Similarly, changes in the user experience requirements can have significant impact on the business and IT infrastructure requirements. Performance might be a major user and business goal, but the system administrator may not be able to invest in the hardware required to meet that goal 100 percent of the time. A balance point might be to meet the goal only 80 percent of the time.

Architecture focuses on how the major elements and components within an application are used by, or interact with, other major elements and components within the application. The selection of data structures and algorithms or the implementation details of individual components are design concerns. Architecture and design concerns very often overlap. Rather than use hard and fast rules to distinguish between architecture and design, it makes sense to combine these two areas. In some cases, decisions are clearly more architectural in nature. In other cases, the decisions are more about design, and how they help you to realize that architecture.

By following the processes described in this guide, and using the information it contains, you will be able to construct architectural solutions that address all of the relevant concerns, can be deployed on your chosen infrastructure, and provide results that meet the original aims and objectives.

Consider the following high level concerns when thinking about software architecture:

- How will the users be using the application?
- How will the application be deployed into production and managed?
- What are the quality attribute requirements for the application, such as security, performance, concurrency, internationalization, and configuration?
- How can the application be designed to be flexible and maintainable over time?
- What are the architectural trends that might impact your application now or after it has been deployed?

4.2.1 Software Architecture:

The Goals of Architecture

An ideal architecture should be a perfect conversion between business requirements and technique requirements by understanding user cases and then defining a clear and neat way to implement those requirements by programming the software.

A good design is sufficiently flexible to be able to handle all of the user case studies and scenarios, both functional and quality requirements, efficient in implementation details.

The Principles of Architecture Design

Design the architecture with evolution in mind so that it will be able to adapt to requirements that are not fully known at the start of the design process, do not try to over engineer the architecture, and make assumptions that you can't verify.

Instead you should keep your options open for future changes, identify the foundational parts of the architecture that represent the greatest risk if you get them wrong.

Key Architecture Principles

Building software for change instead of building to last. There are always new requirements and feedbacks.

Identifying critical decisions. Identify the areas where mistakes and further changes are most often made, getting these key engineering decisions right the first time so the design is more flexible.

Start with base-line architecture to create the big picture, and then evolve the details and iteratively test and improve the architecture. Do not try to get every tiny detail right on the first attempt, get the big decision right first, and then focus on the details.

4.2.2 Data Design:

This section describes data design at both the architectural and component levels. At the architecture level, data design is the process of creating a model of the information represented at a high level of abstraction (using the customer's view of data).

Data Design at the Architectural Level

The challenge is extract useful information from the data environment, particularly when the information desired is cross-functional.

To solve this challenge, the business IT community has developed *data mining* techniques, also called *knowledge discovery in databases* (KDD), that navigate through existing databases in an attempt to extract appropriate business-level information.

However, the existence of multiple databases, their different structures, the degree of detail contained with the databases, and many other factors make data mining difficult within an existing database environment.

An alternative solution, called a *data warehouse*, adds on additional layer to the data architecture.

A data warehouse is a separate data environment that is not directly integrated with day-to-day applications that encompasses all data used by a business.

In a sense, a data warehouse is a large, independent database that has access to the data that are stored in databases that serve as the set of applications required by a business.

Data Design at the Component Level

At the component level, data design focuses on specific data structures required to realize the data objects to be manipulated by a component.

- refine data objects and develop a set of data abstractions
- implement data object attributes as one or more data structures
- review data structures to ensure that appropriate relationships have been established

- simplify data structures as required

Set of principles for data specification:

1. The systematic analysis principles applied to function and behavior should also be applied to data.
2. All data structures and the operations to be performed on each should be identified.
3. A data dictionary should be established and used to define both data and program design.
4. Low level data design decisions should be deferred until late in the design process.
5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.
6. A library of useful data structures and the operations that may be applied to them should be developed.
7. A software design and programming language should support the specification and realization of abstract data types.

4.2.3 Mapping Data Flow into Software Architecture:

This section describes the general process of mapping requirements into software architectures during the structured design process. The technique described in this chapter is based on analysis of the data flow diagram discussed in Chapter 8.

An Architectural Design Method

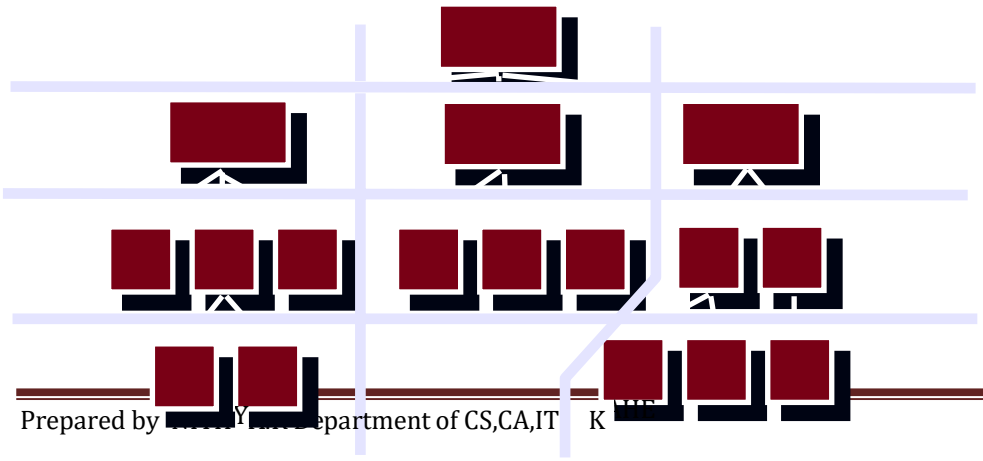
Customer requirements

Four bedrooms, three baths, lots of glass…

Deriving Program Architecture

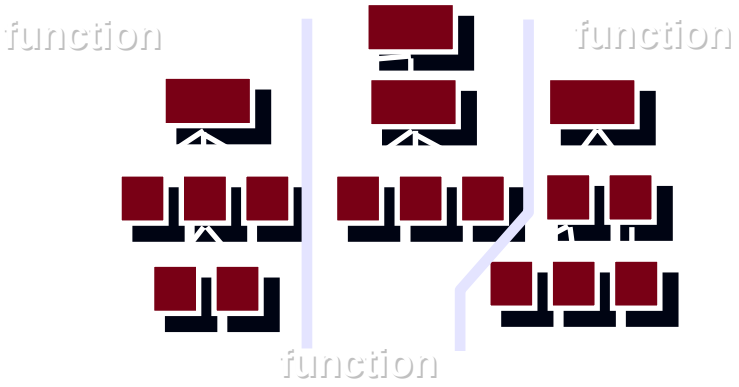
Partitioning the Architecture

"Horizontal" and "Vertical" partitioning are required



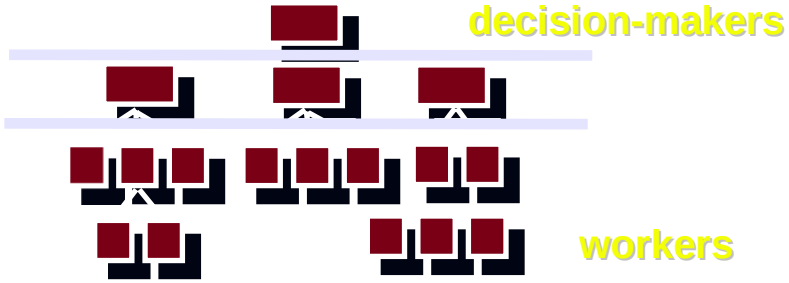
Horizontal Partitioning:

- define separate branches of the module hierarchy for each major function
- use control modules to coordinate communication between functions



Vertical Factoring **Partitioning:**

- design so that decision making and work are stratified
- decision making modules should reside at the top of the architecture



Why Partitioned Architecture?

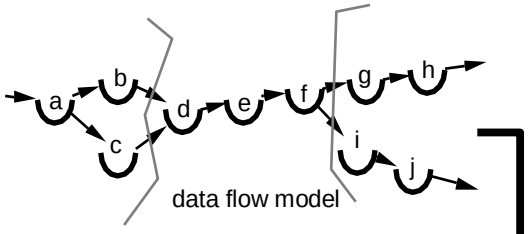
- results in software that is easier to test
- leads to software that is easier to maintain
- results in propagation of fewer side effects
- results in software that is easier to extend

- objective: to derive a program architecture that is partitioned
- approach:
  - the DFD is mapped into a program architecture
  - the PSPEC and STD are used to indicate the content of each module
- notation: structure chart

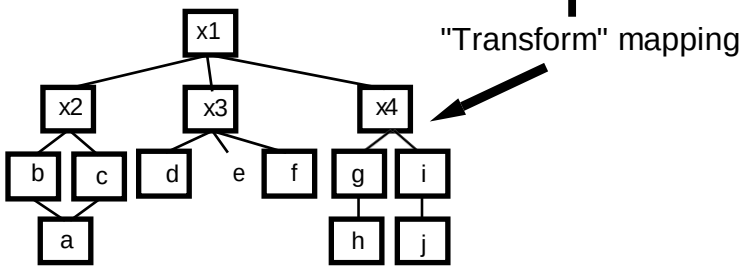
Flow Characteristics

General Mapping Approach

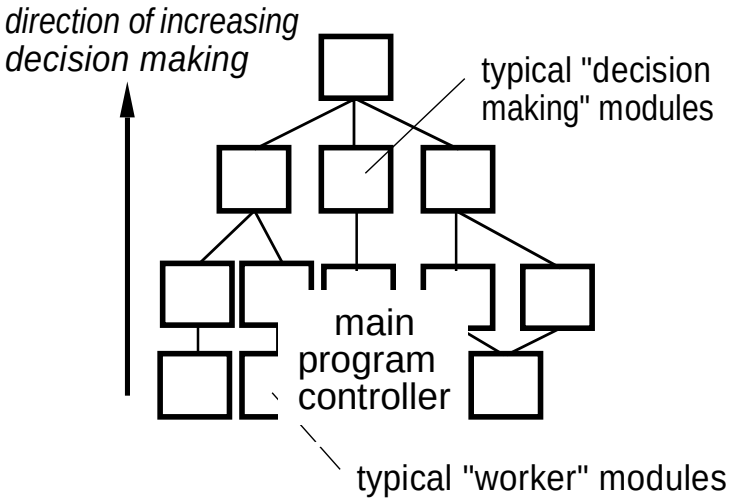
Isolate incoming and outgoing flow boundaries; for transaction flows, isolate the transaction center.
Working from the boundary outward, map DFD transforms into corresponding modules.
Add control modules as required.
Refine the resultant program structure using effective modularity concepts.



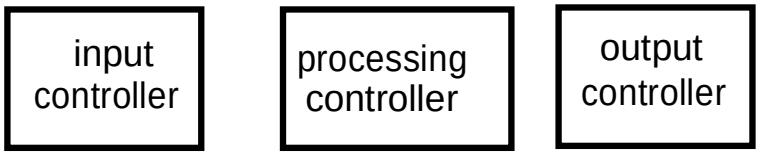
Factoring

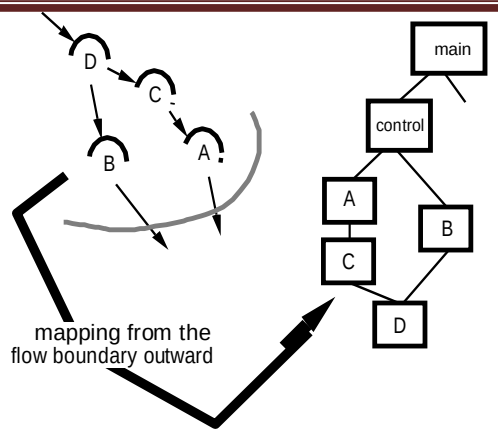


First Level Factoring

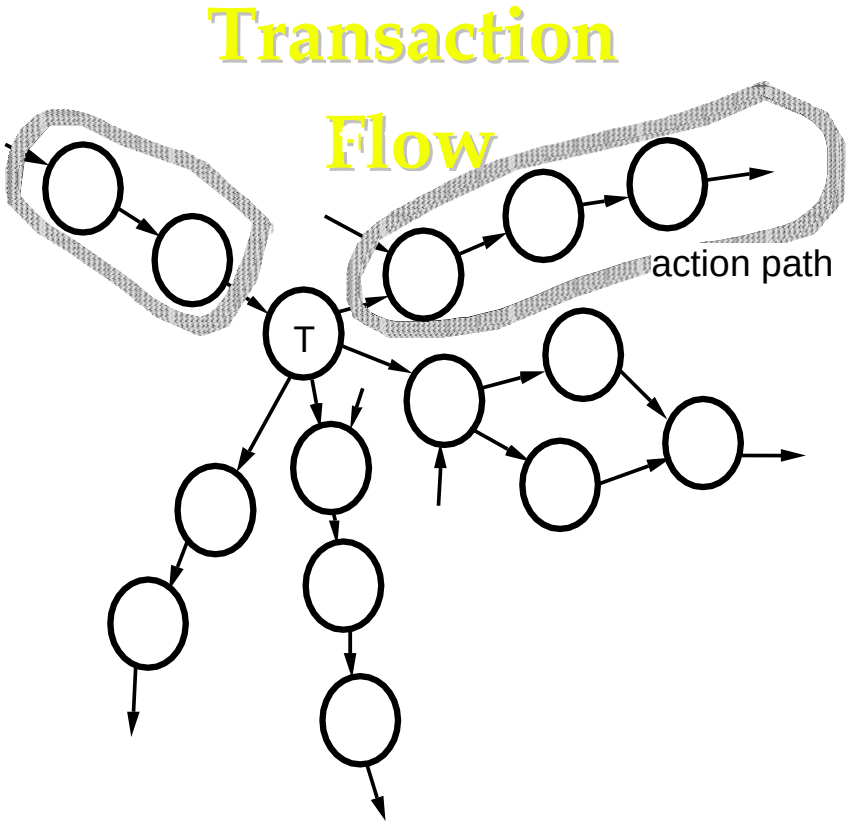


Second Level Mapping





Transaction Flow



Refining the Analysis Model

1. Write an English language processing narrative for the level 01 flow model
2. Apply noun/verb parse to isolate processes, data items, store and entities
3. Develop level 02 and 03 flow models
4. Create corresponding data dictionary entries
5. Refine flow models as appropriate

POSSIBLE QUESTIONS

1.what is meant by design engineering in software?
2.write a note on architecturai design.
3.what is meant by software architecture?
4.write note on data design at architectural design.
5.write a note on component level data design.
6.wnat is meant by dataflow mapping?
7.discuss about the design model.

UNIT IV

181	Interface design focuses on _____ areas of concern.	2	3	4	5	
182	_____ and anxiety are part of daily life for many users of computerized information systems	sadness	frustration	happiness	enjoyment	
183	. Frustration and _____ are part of daily life for many users of computerized information system	sadness	happiness	enjoyment	anxiety	
184	_____ creates effective communication medium between a human and a computer.	user interface design	architectural design	code design	procedure design	
185	_____ identifies interface objects and actions and then creates a screen layout that form the basis for a user interface prototype.	design	coding	testing	analysis	
186	_____ begins with the identification of user, task and environmental requirements.	user interface design	architectural design	code design	procedure design	
187	There are _____ golden rules.	2	3	4	5	
188	We should define interaction modes in a way that does not force a user into unnecessary or undesired actions.	interaction modes	interface constraints	design principles	design analysis	
189	We should provide _____ interaction.	rigid	flexible	encouraging	enthusiastic	
190	We should design for direct interaction with _____ that appear on the screen	code	class	objects	user	
191	We should hide technical _____ from the casual user	reactions	actions	internals	interactions	
192	We should streamline _____ as skill levels advance and allow the interaction to be customized.	internals	interaction	actions	reactions	
193	. We should allow user interaction to be _____ and undoable	interruptible	flexible	rigid	encouraging	
194	We should allow user interaction to interruptible and _____.	undoable	flexible	rigid	encouraging	
195	We should define shortcuts that are _____.	encouraging	intuitive	default	past actions	
196	We should define _____ that are intuitive.	shortcuts	broad area	interruptible actions	interactions	
197	We should disclose information in a _____ fashion.	open	progressive	streamline	flexible	
198	The visual layout of the _____ should be based on a real world metaphor.	interaction modes	interface	design	structure	
199	The interface should present and acquire _____ in a consistent fashion.	information	task	knowledge	idea	
200	The interface should present and acquire information in a _____ fashion.	consistent	inconsistent	rigid	flexible	

201	A _____ of the entire system incorporates data, architectural interface, and procedural representations of the software	data model	design model	user model	system image
202	The software engineer creates a _____.	design model	data model	interface model	system image
203	The end user develops a mental image that is often called the _____.	design model	user model	data model	system image
204	The implementers of the system create a _____.	design model	system image	data model	user model
205	Users are categorized into _____ types.	2	3	4	5
206	Users with no syntactic knowledge of the system and little semantic knowledge of the application or computer usage are called _____.	knowledgeable intermittent users	knowledgeable frequent users	novices	all of the above
207	Users with reasonable semantic knowledge of the application but relatively low recall of syntactic information necessary to use the interface are called _____.	novices	knowledgeable, intermittent users	knowledgeable, frequent users	all of the above
208	Users with good semantic and syntactic knowledge that often leads to the “power-user syndrome” are called _____.	novices	knowledgeable, intermittent users	knowledgeable, frequent users	all of the above
209	Individuals who look for shortcuts and abbreviated modes of interaction are called _____.	novices	knowledgeable, intermittent users	knowledgeable, frequent users	all of the above
210	The _____ is the image of the system that end-users carry in their heads.	user’s model	data model	design model	system image
211	Stepwise elaboration is called _____.	functional decomposition	data abstraction	modularity	modular protection
212	_____ is the only way that we can accurately translate a customer’s requirements into a finished software product or system.	specification	design	data	prototype
213	Validation focuses on _____ criteria.	2	3	4	5
214	Task analysis can be applied in _____ ways.	2	3	4	5
215	Task analysis for interface design used _____ approach.	object oriented approach	top down approach	bottom up approach	all of the above
216	The overall approach to task analysis, a human engineer must first _____ and classify tasks.	discuss	define	describe	list
217	There are _____ steps in interface design activities.	4	5	6	7
218	_____ refers to the deviation from average time.	system response time	variability	system mean time	all of the above
219	System response time has _____ important characteristics.	2	3	4	5
220	A _____ is designed into the software from the beginning.	integrated help facility	system response time	variability	all of the above
221	Component level design also called _____.	procedural abstraction	procedural design	stepwise refinement	decomposition

222	_____ must be translated into operational software	data	architectural	interface design	all of the above
223	A _____ performs component level design.	user	top level management	software engineer	middle level management
224	The _____ represents the software in a way that allows one to review the details of the design for correctness and consistency with earlier design representations.	component level design	procedural design	data design	data design
225	Design, representations of data, architecture, and interfaces form the foundation for _____.	procedural design	component level design	data design	code design
226	_____ notation is used to represent the design.	graphical	tabular	text-based	all of the above
227	Any program, regardless of application area or technical complexity, can be designed and implemented using only the _____ structured constructs.	2	3	4	5
228	A box in a flowchart is used to indicate a _____.	processing step	logical condition	flow of control	start
229	A diamond in a flowchart is used to indicate a _____.	processing step	logical condition	flow of control	start
230	The arrows in a flowchart is used to indicate a _____.	processing step	logical condition	flow of control	start
231	A picture is worth a _____ words.	100	1000	10000	100000
232	The following construct is fundamental to structured programming.	sequence	condition	repetition	all of the above
233	_____ implements processing steps that are essential in the specification of any algorithm.	sequence	condition	repetition	selection
234	_____ provides the facility for selected processing steps that are essential in the specification of any algorithm	sequence	condition	repetition	selection
235	_____ allows for looping.	sequence	condition	repetition	selection
236	Another graphical design tool, the _____ evolved from a desire to develop a procedural design representation that would not allow violation of the structured constructs.	box diagram	flowchart	transition diagram	decision table
237	PDL is the abbreviation of _____.	Process Design Language	Program Design Language	Program Document Language	Program Document Language
238	A design language should have the _____ characters.	2	3	4	5
239	Design notation should support the development of modular software and provide a means for interface specification. This attribute of design notation is called _____.	modularity	simplicity	ease of editing	maintainability
240	. Design notation should be relatively simple to learn, relatively easy to use, and generally easy to read. This attribute of the design notation is called _____.	modularity	simplicity	) ease of editing	maintainability

241	The procedural design may require modification as the software process proceeds. The ease with which a design representation can be edited can help facilitate each software engineering task is called _____.	modularity	simplicity	ease of editing	maintainability
242	Notation that can be input directly into a computer-based development system offers significant benefits. This attribute of design notation is called _____.	machine readability	maintainability	structure enforcement	automatic processing
243	Maintenance of the software configuration nearly always means maintenance of the procedural design representation. This attribute of design notation is called _____.	machine readability	maintainability	structure enforcement	automatic processing
244	Design notation that enforces the use of only the structured constructs promotes good design practices is called _____.	machine readability	maintainability	structure enforcement	automatic processing
245	A procedural design contains information that can be processed to give the designer new or better insights into the correctness and quality of a design. Such insight can be enhanced with reports provides via software design tools is called _____.	machine readability	maintainability	structure enforcement	automatic processing
246	. The ability to represent local and global data is an essential element of component-level design. Ideally, design notation should represent such data directly is called _____.	automatic processing	data representation	logic verification	code-to” ability
247	Automatic verification of design logic is a goal that is paramount during software testing. Notation that enhances the ability to verify logic greatly improves testing adequacy is called _____.	automatic processing	data representation	logic verification	“code-to” ability
248	The software engineering task that follows component level design is code generation. Notation that may be converted easily to source code reduces effort and error. This attribute of design notation is called _____.	automatic processing	data representation	logic verification	“code-to” ability
249	The user interface design process encompasses _____ distinct framework activities.	2	3	4	5
250	Validation focuses on _____.	the ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements	the degree to which the interface is easy to use and easy to learn.	the user’s acceptance of the interface as a useful tool in their work.	all of the above.

3

sadness

anxiety

user interface design

design
user interface
design

3

interaction
modes
flexible

objects
internals

interaction

interruptible
undoable
intuitive

shortcuts
progressive

interface

information

consistent

all of the above
software
engineer

component level
design

component level design

graphical

3

processing step
logical condition
flow of control
1000
all of the above

sequence

condition
repetition

box diagram

Program Design
Language
4

modularity

simplicity

ease of editing

machine
readability

maintainability

structure
enforcement

automatic
processing

data
representation

logic verification

“code-to” ability

all of the above.

Unit - 2

1	The _____ as a bridge between the systm decription and the design model	design	analysis model
2	The role of the software engineer in the requirement analysis is called____	designer	analyst
3	analysis modeling often begins with _____	data modeling	function modeling
4	A _____ can be an external entity	function object	structural object
5	_____ defines the properities of a data object	relationship	cardinality
6	Data objects are connected to one another in different ways is called _____	modality	relationships
7	_____ is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object	modality	relationships
8	_____ defines the maximum number of objects that can participate in a relationship	modality	relationships
9	_____ provide an indication of whether or not a particular data object must participate in the relationship	Modality	relationships
10	The _____ diagram takes an an input-process-output view of sysytem	use-case diagram	activity diagram
11	The level 0 DFD is called as _____ diagram	contract level diagram	context level diagram
12	The _____ describes the behavior of the system but not the inner working of the processes	PSPEC	ASPEC
13	The _____ is used to describe all flow model processes the appear in the final level of refinement	CSPEC	ASPEC
14	The _____ model indicates how software will respond to external events	data	behavior
15	The _____ represents a sequence of activities that involves actor and the system	csase tool	activity
16	The _____ diagram indicates how events cause transitions from object to object	sequence diagram	activity
17	Which one depict the software requirements from the user's point of view.	behavioral based	flow based model
18	Which model depicts how input is transformed into output as data objects move through a system	behavioral based	flow based model
19			
20			

Unit 3

1	_____is a iterativeprocess through which requirements are translated into a blueprint for constructing the software	requirements gathering	coding
2	Who developeda set of software quality attributefor the software design	Barry Boehm	R.Pattis
3	Which quality attribute measure the response time, throughput and effeciency of the sysytem	Functionality	Usability

4	The quality attribute, Usability is assessed by considering the overall _____ of the system	consistency	Functionality
5	A _____ refers to a sequence of instructionsthat have a specific and limited functions	procedural abstraction	data abstraction
6	_____ represent architecture as an organized collection of programs components	process models	structural models
7	_____ models address the behavioral aspects of the program architecture	process models	structural models
8	Software is divided into separately named and addressable components is called _____	process	behavior
9	the _____ is a process of changing a software by which doesnot alter the external behavior of the code	refinement	cohesion
10	_____ is an indication of the relative functional strength of the module	refinement	cohesion
11	_____ is an indication of the relativeinterdependency among modules	cohesion	patterns
12	Refinement is a top-down design strategy which is actually a process of _____	eloboration	abstraction
13	A _____ is a named collection of data that describes a data object.	procedural abstraction	data abstraction
14	_____ implies a program control mechanism without specifying internal detail.	procedural abstraction	data abstraction
15	software architecture consider ____ levels of the design pyramid	3	2
16	Which action translates data objects into data structures	data design	component design
17	In data centered arcjitecture _____ resides at the centre of the architecture which is accessed frequently by other components	client software	data store
18	_____ represents the structure of data and program components that are required to build a computer-based syste,	architectural design	data design
19	KDD stand for	Knowledge Discovery of data manipulatio	Knowledge Discovery in database
20	the _____ classes defines all abstraction that are necessary for human computer interaction	primitive class	user interface
21	The _____ classes implement lower level business abstraction required to manage the business domain class	primitive class	user interface
22	_____ suggest that a method should send or receive messages from friend class	Law of cohesion	Law of meter
23	_____ is achieve by developing modules with single minded function and aversion of excessive interaction	refinement	refactoring
24	_____ suggest that the information contained in one module is inaccesible to othe modules	refinement	refactoring
25	Refinement is a process of _____	abstraction	eloboration
26	_____ is a process of breaking up of complex problem into a manageable piecies	refinement	refactoring
27	_____ is evaluated by measuring the frequency and severity of failure.	Usability	performance

28	iIn transform flow the information must entered and exit in _____ form	external world	internal world
29	Information flow is characterized by an single data item is called	context flow	transaction flow
30	A _____ diagram is mapped into a program structure using transform or transaction mapping	data flow	use case
31	_____ language provides a semantic and syntax for describing a software architecture	architectural description	architectural design
32	Design begins with the _____ model.	data	requirements
33	_____ focus on the design of the business or technical process that the system must accommodate.	framework models	dynamic models
34	_____ can be used to represent the functional hierarchy of a system.	framework models	dynamic models
35	_____ represent architecture as an organized collection of program components.	dynamic models	functional models
36	_____ increases the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.	framework models	dynamic models
37	_____ address the behavioural aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.	framework models	dynamic models
38	_____ is the place where quality is fostered in software engineering	model	data
39	_____ provides us with representations of software that can be assessed for quality.	design	specification
40	_____ describes a program should not have any bugs that inhibit its function	firmness	commodity
41	A program should be suitable for the purposes for which it is intended is called	firmness	commodity
42	The experience of using the program should be a pleasurable one is called	firmness	commodity
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			

planning	construction	analysis model
programmer	tester	analyst
behavior modling	structure modeling	
Data object	flow object	Data object
Data attributes	modality	Data attributes
cardinality	Data attributes	relationships
cardinality	Data attributes	Cardinality
Data attributes	cardinality	Cardinality
Data attributes	cardinality	Modality
data flow diagram	ERD	data flow diagram
text level diagram	zero level diagram	context level diagram
LSPEC	CSPEC	CSPEC
LSPEC	PSPEC	PSPEC
function	structural	behavior
use-case	swimlane	use-case
use-case	swimlane	sequence diagram
class based	scenario based model	scenario based model
class based	scenario based model	flow based model
software design	deployment	software design
M.C.Escher	Hewlett-Packard	Hewlett-Packard
Performance	Supportability	Performance

Supportability	Performance	consistency
behavior abstraction	structural abstraction	procedural abstraction
dynamic models	framework models	structural models
dynamic models	framework models	dynamic models
modules	data	Modules
patterns	refactoring	refactoring
patterns	refactoring	cohesion
coupling	functional dependency	coupling
refactoring	information hiding	eloboration
control abstraction	behavior abstraction	data abstraction
control abstraction	behavior abstraction	control abstraction

1	4	2
behavior design	functional design	data design
filter	pipes	data store
software design	behavioural design	architectural design
Knowing of database discovery process	Knowing discovery of database business	Knowledge Discovery in database
classes	domain	user interface
process classes	System class	process classes
Low of completeness	law of primitiveness	Law of meter
functional independence	information hiding	functional independence
functional independence	information hiding	information hiding
architecture	modularity	eloboration
modularity	arichiteture	modularity
supportability	reliability	reliability

top down	bottom up	external world
transform flow	contract flow	transaction flow
state diagram	activity diagram	data flow
architectural pattern	architecturaldefinition	architectural description
specification	code	requirements
process models	functional models	process models
process models	functional models	functional models
framework models	structural models	structural models
process models	functional models	framework models
process models	functional models	dynamic models
design	specification	design
data	prototype	design
delight	roman	firmness
delight	roman	commodity
delight	roman	delight

UNIT 5
SYLLABUS

Testing Strategies & Tactics: Software Testing Fundamentals, Strategic Approach to Software Testing, Test Strategies for Conventional Software, Validation Testing, System testing Black-Box Testing, White-Box Testing and their type, Basis Path Testing

MATERIAL

5.1 Testing Tactics:

- The development of software systems involves a series of production activities where opportunities for injection of human fallibilities are enormous.
- Errors may begin to occur at the very commencement of the process where the objectives may be erroneously or imperfectly specified, as well as [in] later design and development stages.
- Because of human inability to perform and communicate with perfection, software development is accompanied by a quality assurance activity.
- Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation.
- The increasing visibility of software as a system element and the attendant "costs" associated with a software failure are motivating forces for well-planned, thorough testing. It is not unusual for a software development organization to expend between 30 and 40 percent of total project effort on testing.
- In the extreme, testing of human-rated software (e.g., flight control, nuclear reactor monitoring) can cost three to five times as much as all other software engineering steps combined!

5.1.1 Software Testing Fundamentals:

- Testing presents an interesting anomaly for the software engineer. During earlier software engineering activities, the engineer attempts to build software from an abstract concept to a tangible product.
- The engineer creates a series of test cases that are intended to "demolish" the software that has been built.
- In fact, testing is the one step in the software process that could be viewed (psychologically, at least) as destructive rather than constructive.
- Software engineers are by their nature constructive people.
- Testing requires that the developer discard preconceived notions of the "correctness" of software just developed and overcome a conflict of interest that occurs when errors are uncovered.
- Beizer describes this situation effectively when he states: There's a myth that if we were really good at programming, there would be no bugs to catch. If only we could really concentrate, if only everyone used structured programming, top down design, decision tables, if programs were written in SQUISH, if we had the right silver bullets, then there would be no bugs. So goes the myth. There are bugs, the myth

says, because we are bad at what we do; and if we are bad at it, we should feel guilty about it. Therefore, testing and test case design is an admission of failure, which instills a goodly dose of guilt.

Testing Objectives

- Glen Myers states a number of rules that can serve well as testing objectives:
  1. Testing is a process of executing a program with the intent of finding an error.
  2. A good test case is one that has a high probability of finding an as-yet undiscovered error.
  3. A successful test is one that uncovers an as-yet-undiscovered error.
- If testing is conducted successfully (according to the objectives stated previously), it will uncover errors in the software.
- Also testing demonstrates that software functions appear to be working according to specification, that behavioral and performance requirements appear to have been met.
- In addition, data collected as testing is conducted provide a good indication of software reliability and some indication of software quality as a whole.
- But testing cannot show the absence of errors and defects, it can show only that software errors and defects are present.

Testing Principles

- Before applying methods to design effective test cases, a software engineer must understand the basic principles that guide software testing. Davis [DAV95] suggests a set of testing principles.

  • **All tests should be traceable to customer requirements**.

  The objective of software testing is to uncover errors. It follows that the most severe defects (from the customer's point of view) are those that cause the program to fail to meet its requirements.

  • **Tests should be planned long before testing begins**.

  Test planning can begin as soon as the requirements model is complete.

  Detailed definition of test cases can begin as soon as the design model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.

  • **The Pareto principle applies to software testing**.

  Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.

  • **Testing should begin "in the small" and progress toward testing "in the large."**

  The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

  • **Exhaustive testing is not possible**.

The number of path permutations for even a moderately sized program is exceptionally large. For this reason, it is impossible to execute every combination of paths during testing. It is possible, however, to adequately cover program logic and to ensure that all conditions in the component-level design have been exercised.

• **To be most effective, testing should be conducted by an independent third party.**

Testability

- Software testability is simply how easily a computer program can be tested.
- Since testing is so profoundly difficult, it pays to know what can be done to streamline it.
- Sometimes programmers are willing to do things that will help the testing process and a checklist of possible design points, features, etc., can be useful in negotiating with them.
- "Testability" occurs as a result of good design. Data design, architecture, interfaces, and component-level detail can either facilitate testing or make it difficult.

The **checklist** that follows provides a set of characteristics that lead to testable software.

Operability. "The better it works, the more efficiently it can be tested."

• The system has few bugs (bugs add analysis and reporting overhead to the test process).
• No bugs block the execution of tests.
• The product evolves in functional stages (allows simultaneous development and testing).

Observability. "What you see is what you test."

• Distinct output is generated for each input.
• System states and variables are visible or queriable during execution.
• Past system states and variables are visible or queriable (e.g., transaction logs).
• All factors affecting the output are visible.
• Incorrect output is easily identified.
• Internal errors are automatically detected through self-testing mechanisms.
• Internal errors are automatically reported.
• Source code is accessible.

Controllability. "The better we can control the software, the more the testing can be automated and optimized."

• All possible outputs can be generated through some combination of input.
• All code is executable through some combination of input.
• Software and hardware states and variables can be controlled directly by the test engineer.
• Input and output formats are consistent and structured.
• Tests can be conveniently specified, automated, and reproduced.

Decomposability. "By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting."

- The software system is built from independent modules.
- Software modules can be tested independently.

Simplicity. "The less there is to test, the more quickly we can test it."

- Functional simplicity (e.g., the feature set is the minimum necessary to meet requirements).
- Structural simplicity (e.g., architecture is modularized to limit the propagation of faults).
- Code simplicity (e.g., a coding standard is adopted for ease of inspection and maintenance).

Stability. "The fewer the changes, the fewer the disruptions to testing."

- Changes to the software are infrequent.
- Changes to the software are controlled.
- Changes to the software do not invalidate existing tests.
- The software recovers well from failures.

Understandability. "The more information we have, the smarter we will test."

- The design is well understood.
- Dependencies between internal, external, and shared components are well understood.
- Changes to the design are communicated.
- Technical documentation is instantly accessible.
- Technical documentation is well organized.
- Technical documentation is specific and detailed.
- Technical documentation is accurate.

- Kaner, Falk, and Nguyen suggest the following **attributes of a "good" test**:
  1. **A good test has a high probability of finding an error.**
     - To achieve this goal, the tester must understand the software and attempt to develop a mental picture of how the software might fail.
     - Ideally, the classes of failure are probed. For example, one class of potential failure in a GUI (graphical user interface) is a failure to recognize proper mouse position.
     - A set of tests would be designed to exercise the mouse in an attempt to demonstrate an error in mouse position recognition.
  2. **A good test is not redundant.**
     - Testing time and resources are limited. There is no point in conducting a test that has the same purpose as another test. Every test should have a different purpose.
  3. **A good test should be "best of breed".**
     - In a group of tests that have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests.
     - In such cases, the test that has the highest likelihood of uncovering a whole class of errors should be used.

4. **A good test should be neither too simple nor too complex.**
   - Although it is sometimes possible to combine a series of tests into one test case, the possible side effects associated with this approach may mask errors.
   - In general, each test should be executed separately.

5.1.2 White-Box Testing

- White-box testing, called **glass-box testing** is a test case design method that uses the control structure of the procedural design to derive test cases.
- Using white-box testing methods, the software engineer can derive test cases that
  1. guarantee that all independent paths within a module have been exercised at least once,
  2. exercise all logical decisions on their true and false sides,
  3. execute all loops at their boundaries and within their operational bounds, and
  4. exercise internal data structures to ensure their validity.
- "Why spend time and energy worrying about (and testing) logical minutiae when we might better expend effort ensuring that program requirements have been met?" or "Why don't we spend all of our energy on black-box tests?"
- The answer is :
  - **Logic errors and incorrect assumptions are inversely proportional to the probability that a program path will be executed.** Errors tend to creep into our work when we design and implement function, conditions, or controls that are out of the mainstream. Everyday processing tends to be well understood (and well scrutinized), while "special case" processing tends to fall into the cracks.
  - **We often believe that a logical path is not likely to be executed when, in fact, it may be executed on a regular basis.** The logical flow of a program is sometimes counterintuitive, meaning that our unconscious assumptions about flow of control and data may lead us to make design errors that are uncovered only once path testing commences.
  - **Typographical errors are random.** When a program is translated into programming language source code, it is likely that some typing errors will occur. Many will be uncovered by syntax and type checking mechanisms, but others may go undetected until testing begins. It is as likely that a typo will exist on an obscure logical path as on a mainstream path.
- Each of these reasons provides an argument for conducting white-box tests. Black-box testing, no matter how thorough, may miss the kinds of errors noted here. White-box testing is far more likely to uncover them.

5.1.3 Basis Path Testing

- Basis path testing is a white-box testing technique first proposed by **Tom McCabe** in 1976.
- The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.

- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

Flow Graph Notation

- The flow graph depicts logical control flow using the notation illustrated in Fig 5.1.

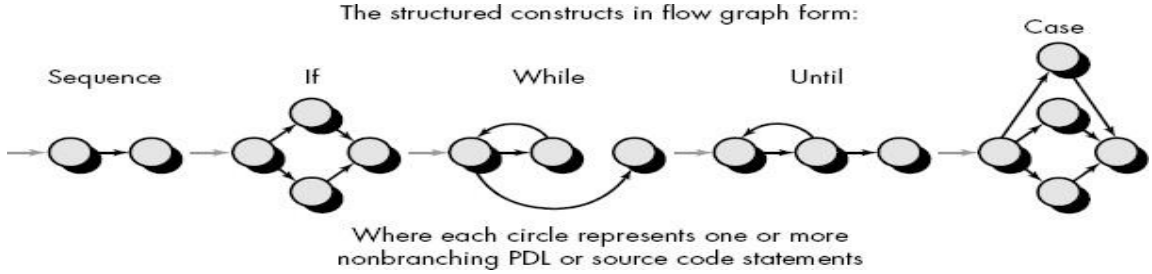


Fig 5.1 Flow graph notation

- Each structured construct has a corresponding flow graph symbol. To illustrate the use of a flow graph, we consider the procedural design representation in Fig 5.2A. Here, a flowchart is used to depict program control structure.

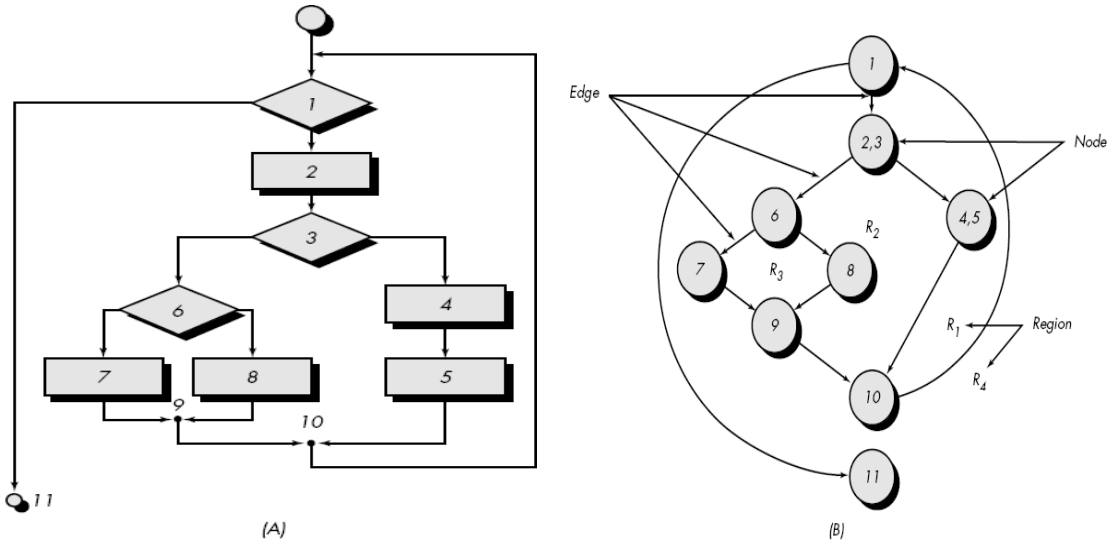


Fig 5.2 Flowchart, (A) and flow graph (B)

- Fig 5.2B maps the flowchart into a corresponding flow graph (assuming that no compound conditions are contained in the decision diamonds of the flowchart).
- Referring to Fig 5.2B, each circle, called a flow graph node, represents one or more procedural statements.
- A sequence of process boxes and a decision diamond can map into a singlenode.
- The arrows on the flow graph, called edges or links, represent flow of control and are analogous to flowchart arrows.
- An edge must terminate at a node, even if the node does not represent any procedural statements (e.g., see the symbol for the if-then-else construct).
- Areas bounded by edges and nodes are called regions. When counting regions, we include the area outside the graph as a region.4

- When compound conditions are encountered in a procedural design, the generation of a flow graph becomes slightly more complicated.
- A compound condition occurs when one or more Boolean operators (logical OR, AND, NAND, NOR) is present in a conditional statement.
- Referring to Fig 5.3, the PDL segment translates into the flow graph shown.
- **Note:** A separate node is created for each of the conditions a and b in the statement IF a OR b. Each node that contains a condition is called a predicate node and is characterized by two or more edges emanating from it.

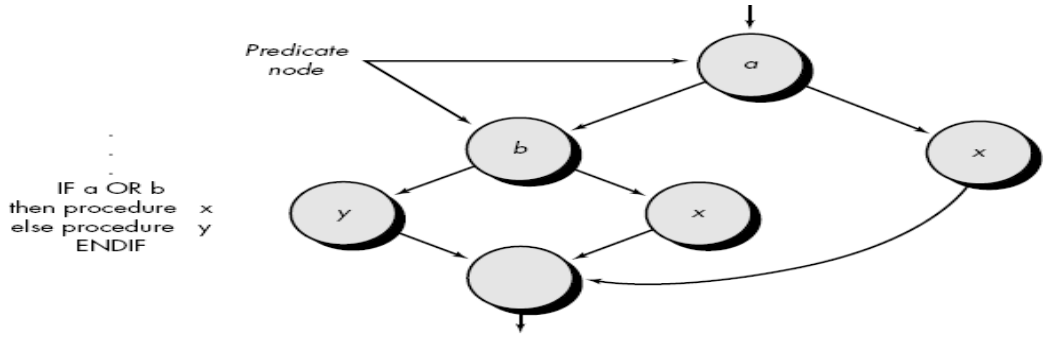


Fig 5.3 Compound logic

Cyclomatic Complexity

- **Cyclomatic complexity** is a software metric that provides a quantitative measure of the logical complexity of a program.
- Cyclomatic complexity has a foundation in graph theory and provides us with extremely useful software metric.
- Cyclomatic complexity is defined by the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
- **An independent path** is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.
- For example, a set of independent paths for the flow graph illustrated in Fig 5.2B is
  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
- **Note: Each new path introduces a new edge**.
- The path **1-2-3-4-5-10-1-2-3-6-8-9-10-1-11** is not considered to be an independent path because it is simply a combination of already specified paths and does not traverse any new edges.
- Paths 1, 2, 3, and 4 constitute a basis set for the flow graph in Fig 5.2B. That is, if tests can be designed to force execution of these paths (a basis set), every statement in the program will have been guaranteed to be executed at least one time and every condition will have been executed on its true and false sides.

- **Note:** The basis set is not unique. In fact, a number of different basis sets can be derived for a given procedural design.
- **How do we know how many paths to look for?** The computation of cyclomatic complexity provides the answer.
- Complexity is computed in one of three ways:
  1. The number of regions of the flow graph corresponds to the cyclomatic complexity.
  2. Cyclomatic complexity, V(G), for a flow graph, G, is defined as V(G) = E - N + 2 where E is the number of flow graph edges, N is the number of flow graph nodes.
  3. Cyclomatic complexity, V(G), for a flow graph, G, is also defined as V(G) = P +1 where P is the number of predicate nodes contained in the flow graph G.
- The Cyclomatic complexity of the flow graph in Fig 5.2B, can be computed using each of the algorithms just noted:
  1. The flow graph has four regions.
  2. V(G) = 11 edges - 9 nodes + 2 = 4.
  3. V(G) = 3 predicate nodes + 1 = 4.

  Therefore, the cyclomatic complexity of the flow graph in Figure 17.2B is 4.
- **Important**: the value for V(G) provides us with an upper bound for the number of independent paths that form the basis set and, by implication, an upper bound on the number of tests that must be designed and executed to guarantee coverage of all program statements.

Deriving Test Cases

- The basis path testing method can be applied to a procedural design or to source code.
- The procedure average, depicted in PDL in Fig 5.4, will be used as an example to illustrate each step in the test case design method.
- The following steps can be applied to derive the basis set:

```
PROCEDURE average;

   * This procedure computes the average of 100 or fewer
     numbers that lie between bounding values; it also computes the
     sum and the total number valid.

   INTERFACE RETURNS average, total.input, total.valid;
   INTERFACE ACCEPTS value, minimum, maximum;

   TYPE value[1:100] IS SCALAR ARRAY;
   TYPE average, total.input, total.valid;
      minimum, maximum, sum IS SCALAR;
   TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;          2
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100   3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum   6
5        THEN increment total.valid by 1;
7                sum = s sum + value[i]
         ELSE skip
8     ENDIF
      increment i by 1;
9  ENDDO
   IF total.valid > 0   10
11    THEN average = sum / total.valid;
12    ELSE average = -999;
13 ENDIF
END average
```

Fig 5.4 PDL for test case design with nodes identified

1. **Using the design or code as a foundation, draw a corresponding flow graph**.

A flow graph is created using the symbols and construction rules. Referring to the PDL for average in Fig 5.4, a flow graph is created by numbering those PDL statements that will be mapped into corresponding flow graph nodes. The corresponding flow graph is in Fig 5.5.

2. **Determine the cyclomatic complexity of the resultant flow graph**.

The Cyclomatic complexity, V(G), is determined by applying the data flow testing algorithms. It should be noted that V (G) can be determined without developing a flow graph by counting all conditional statements in the PDL (for the procedure average, compound conditions count as two) and adding 1. Referring to Fig 5.5,

V(G) = 6 regions
V(G) = 17 edges - 13 nodes + 2 = 6
V(G) = 5 predicate nodes + 1 = 6

3. **Determine a basis set of linearly independent paths**.

The value of V(G) provides the number of linearly independent paths through the program control structure. In the case of procedure average, we expect to specify six paths:

path 1: 1-2-10-11-13
path 2: 1-2-10-12-13
path 3: 1-2-3-10-11-13
path 4: 1-2-3-4-5-8-9-2-. . .
path 5: 1-2-3-4-5-6-8-9-2-. . .
path 6: 1-2-3-4-5-6-7-8-9-2-. . .

The ellipsis (. . .) following paths 4, 5, and 6 indicates that any path through the remainder of the control structure is acceptable. It is often worthwhile to identify predicate nodes as an aid in the derivation of test cases. In this case, nodes 2, 3, 5, 6, and 10 are predicate nodes.

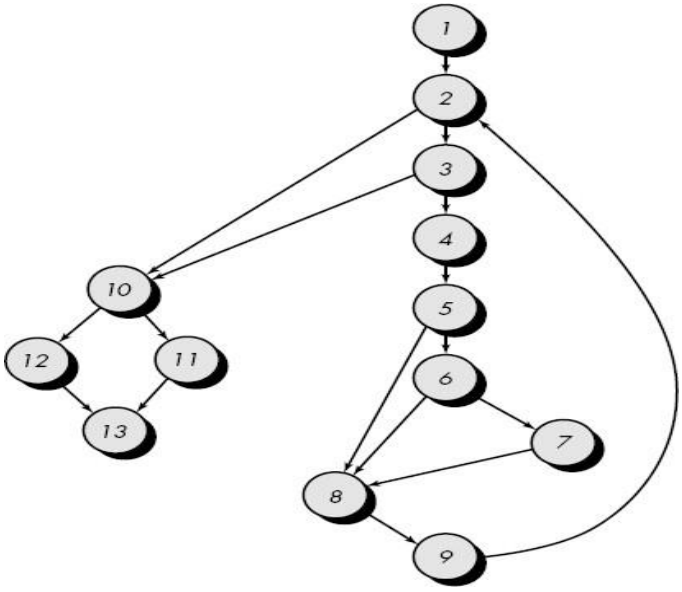


Fig 5.5 Flow graph for the procedure average

4. Prepare test cases that will force execution of each path in the basis set.

Data should be chosen so that conditions at the predicate nodes are appropriately set as each path is tested. Test cases that satisfy the basis set just described are

Path 1 test case:

value(k) = valid input, where $k < i$ for $2 \leq i \leq 100$

value(i) = -999 where $2 \leq i \leq 100$

Expected results: Correct average based on k values and proper totals.

Note: Path 1 cannot be tested stand-alone but must be tested as part of path 4, 5, and 6 tests.

Path 2 test case:

value(1) = -999

Expected results: Average = -999; other totals at initial values.

Path 3 test case:

Attempt to process 101 or more values.

First 100 values should be valid.

Expected results: Same as test case 1.

Path 4 test case:

value(i) = valid input where $i < 100$

value(k) < minimum where $k < i$

Expected results: Correct average based on k values and proper totals.

Path 5 test case:

value(i) = valid input where $i < 100$

value(k) > maximum where $k <= i$

Expected results: Correct average based on n values and proper totals.

Path 6 test case:

value(i) = valid input where $i < 100$

Expected results: Correct average based on n values and proper totals.

- Each test case is executed and compared to expected results. Once all test cases have been completed, the tester can be sure that all statements in the program have been executed at least once.
- **Note:** Some independent paths (e.g., path 1 in our example) cannot be tested in stand-alone fashion. That is, the combination of data required to traverse the path cannot be achieved in the normal flow of the program. In such cases, these paths are tested as part of another path test.

Graph Matrices

- The procedure for deriving the flow graph and determining a set of basis paths is amenable to mechanization.
- To develop a software tool that assists in basis path testing, a data structure, called a graph matrix, can be quite useful.
- A **graph matrix** is a square matrix whose size (i.e., number of rows and columns) is equal to the number of nodes on the flow graph. Each row and column corresponds to an identified node, and matrix entries correspond to connections (an edge) between nodes.

- A simple example of a flow graph and its corresponding graph matrix is shown in Fig 5.6.
- Referring to the figure, each node on the flow graph is identified by numbers, while each edge is identified by letters. A letter entry is made in the matrix to correspond to a connection between two nodes. For example, node 3 is connected to node 4 by edge b.
- So, the graph matrix is nothing more than a tabular representation of a flow graph. However, by adding a link weight to each matrix entry, the graph matrix can become a powerful tool for evaluating program control structure during testing.
- The link weight provides additional information about control flow. In its simplest form, the link weight is 1 (a connection exists) or 0 (a connection does not exist). But link weights can be assigned other interesting properties:
  - The probability that a link (edge) will be executed.
  - The processing time expended during traversal of a link.
  - The memory required during traversal of a link.
  - The resources required during traversal of a link

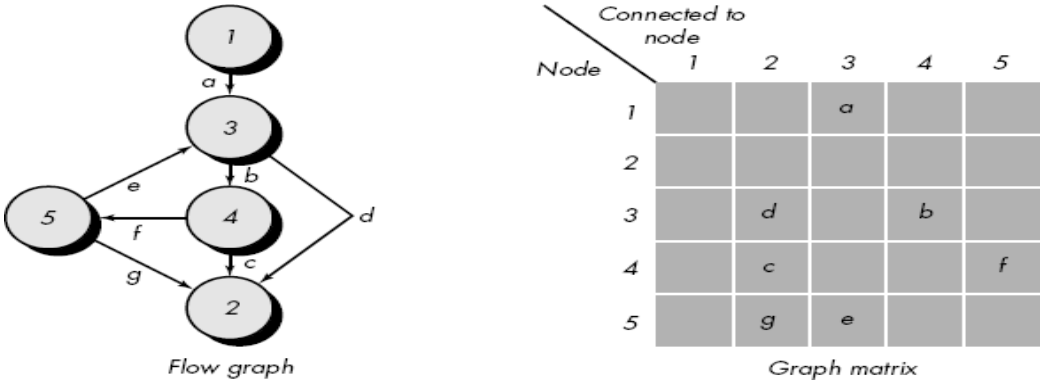


Fig 5.6 Graph matrix

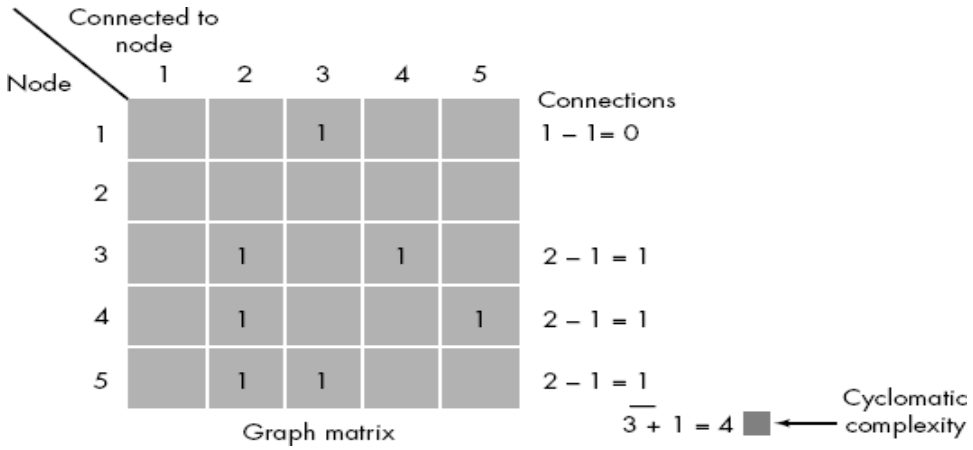


Fig 5.7 Connection matrix

- To illustrate, we use the simplest weighting to indicate connections (0 or 1). The graph matrix in Fig 5.6 is redrawn as shown in Fig 5.7. Each letter has been replaced

with a 1, indicating that a connection exists (zeros have been excluded for clarity). Represented in this form, the graph matrix is called a **connection matrix**.

- Referring to Fig 5.7, each row with two or more entries represents a predicate node. Therefore, performing the arithmetic shown to the right of the connection matrix provides us with still another method for determining cyclomatic complexity.

Beizer [BEI90] provides a thorough treatment of additional mathematical algorithms that can be applied to graph matrices. Using these techniques, the analysis required to design test cases can be partially or fully automated.

5.2 Control Structure Testing

- The basis path testing technique is one of a number of techniques for control structure testing.
- Other variations on control structure testing are discussed. These broaden testing coverage and improve quality of white-box testing.

5.2.1 Condition Testing

- **Condition testing** is a test case design method that exercises the logical conditions contained in a program module.
- A simple condition is a Boolean variable or a relational expression, possibly preceded with one NOT (¬) operator.
- A **relational expression** takes the form **E1 <relational-operator> E2** where E1 and E2 are arithmetic expressions and <relational-operator> is one of the following: <, ≤, =, ≠ (nonequality), >, or ≥.
- A **compound condition** is composed of two or more simple conditions, Boolean operators, and parentheses. We assume that Boolean operators allowed in a compound condition include OR (|), AND (&) and NOT (¬).
- A condition without relational expressions is referred to as a **Boolean expression**. Therefore, the possible types of elements in a condition include a Boolean operator, a Boolean variable, a pair of Boolean parentheses (surrounding a simple or compound condition), a relational operator, or an arithmetic expression.
- If a condition is incorrect, then at least one component of the condition is incorrect. Therefore, types of errors in a condition include the following:
  - Boolean operator error (incorrect/missing/extra Boolean operators).
  - Boolean variable error.
  - Boolean parenthesis error.
  - Relational operator error.
  - Arithmetic expression error.
- The condition testing method focuses on testing each condition in the program.
- Condition testing strategies have two advantages.
  1. Measurement of test coverage of a condition is simple.
  2. Test coverage of conditions in a program provides guidance for the generation of additional tests for the program.
- The purpose of condition testing is to detect not only errors in the conditions of a program but also other errors in the program.
- A number of condition testing strategies have been proposed.

- **Branch testing** is probably the simplest condition testing strategy. For a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once.
- **Domain testing** requires three or four tests to be derived for a relational expression. For a relational expression of the form **E1 <relational-operator> E2** three tests are required to make the value of E1 greater than, equal to, or less than that of E2. If <relational-operator> is incorrect and E1 and E2 are correct, then these three tests guarantee the detection of the relational operator error. To detect errors in E1 and E2, a test that makes the value of E1 greater or less than that of E2 should make the difference between these two values as small as possible.

5.2.2 Data Flow Testing

- The data flow testing method selects test paths of a program according to the locations of definitions and uses of variables in the program.
- To illustrate the data flow testing approach, assume that each statement in a program is assigned a unique statement number and that each function does not modify its parameters or global variables.
- For a statement with S as its statement number,
  DEF(S) = {X | statement S contains a definition of X}
  USE(S) = {X | statement S contains a use of X}
  If statement S is an if or loop statement, its DEF set is empty and its USE set is based on the condition of statement S. The definition of variable X at statement S is said to be live at statement S' if there exists a path from statement S to statement S' that contains no other definition of X.
- A definition-use (DU) chain of variable X is of the form [X, S, S'], where S and S' are statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is live at statement S'.
- Data flow testing strategies are useful for selecting test paths of a program containing nested if and loop statements. To illustrate this, consider the application of DU testing to select test paths for the PDL that follows:

```
proc x
        B1;
        do while C1
                if C2
                 then
                        if C4
                        then B4;
                        else B5;
                        endif;
                else
                        if C3
                        then B2;
                        else B3;
                        endif;
                endif;
        enddo;
```

B6;
end proc;

- To apply the DU testing strategy to select test paths of the control flow diagram, we need to know the definitions and uses of variables in each condition or block in the PDL.
- Assume that variable X is defined in the last statement of blocks B1, B2, B3, B4, and B5 and is used in the first statement of blocks B2, B3, B4, B5, and B6. The DU testing strategy requires an execution of the shortest path from each of Bi, $0 < i \leq 5$, to each of Bj, $1 < j \leq 6$. Although there are 25 DU chains of variable X, we need only five paths to cover these DU chains. The reason is that five paths are needed to cover the DU chain of X from Bi, $0 < i \leq 5$, to B6 and other DU chains can be covered by making these five paths contain iterations of the loop.
- Since the statements in a program are related to each other according to the definitions and uses of variables, the data flow testing approach is effective for error detection.
- However, the problems of measuring test coverage and selecting test paths for data flow testing are more difficult than the corresponding problems for condition testing.

5.2.3 Loop Testing

- Loops are the cornerstone for the vast majority of all algorithms implemented in software.
- Loop testing is a white-box testing technique that focuses exclusively on the validity of loop constructs.
- Four different classes of loops can be defined: simple loops, concatenated loops, nested loops, and unstructured loops (Fig 5.8).

Simple loops.

- The following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop.
  1. Skip the loop entirely.
  2. Only one pass through the loop.
  3. Two passes through the loop.
  4. m passes through the loop where $m < n$.
  5. n -1, n, n + 1 passes through the loop.

Nested loops.

- If we were to extend the test approach for simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increases.
- Beizer suggests an approach that will help to reduce the number of tests:
  1. Start at the innermost loop. Set all other loops to minimum values.
  2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.
  3. Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to "typical" values.
  4. Continue until all loops have been tested.

Concatenated loops.

- Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other. However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent. When the loops are not independent, the approach applied to nested loops is recommended.

Unstructured loops.

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

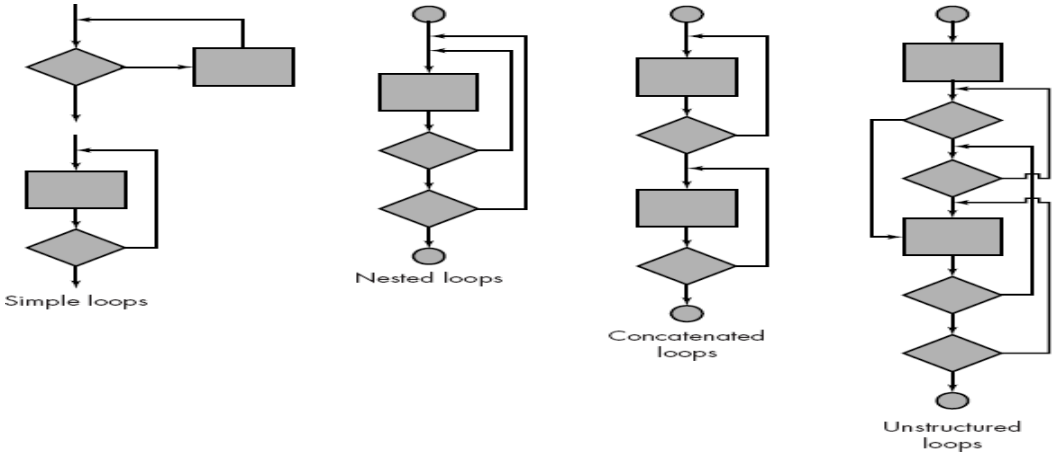


Fig 5.8 Classes of loops

5.3 BLACK-BOX TESTING

- Black-box testing, also called behavioral testing, focuses on the functional requirements of the software.
- That is, black-box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to white-box techniques.
- Rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.

- Black-box testing attempts to find errors in the following categories:
  1. Incorrect or missing functions,
  2. Interface errors,
  3. Errors in data structures or external data base access,
  4. Behavior or performance errors, and
  5. Initialization and termination errors.
- Unlike white-box testing, which is performed early in the testing process, blackbox testing tends to be applied during later stages of testing.

- Black-box testing purposely disregards control structure, attention is focused on the information domain.
- Tests are designed to answer the following questions:
  • How is functional validity tested?
  • How is 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 a data class isolated?
  • What data rates and data volume can the system tolerate?
  • What effect will specific combinations of data have on system operation?
- Black-box techniques, we derive a set of test cases that satisfy the following criteria:
  1. Test cases that reduce, by a count that is greater than one, the number of additional test cases that must be designed to achieve reasonable testing and
  2. Test cases that tell us something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

5.3.1 Graph-Based Testing Methods

- The first step in black-box testing is to understand the objects that are modeled in software and the relationships that connect these objects.
- Next step is to define a series of tests that verify "all objects have the expected relationship to one another".
- To accomplish these steps, the software engineer begins by creating a **graph**—a collection of nodes that represent objects; links that represent the relationships between objects; node weights that describe the properties of a node (e.g., a specific data value or state behavior); and link weights that describe some characteristic of a link.

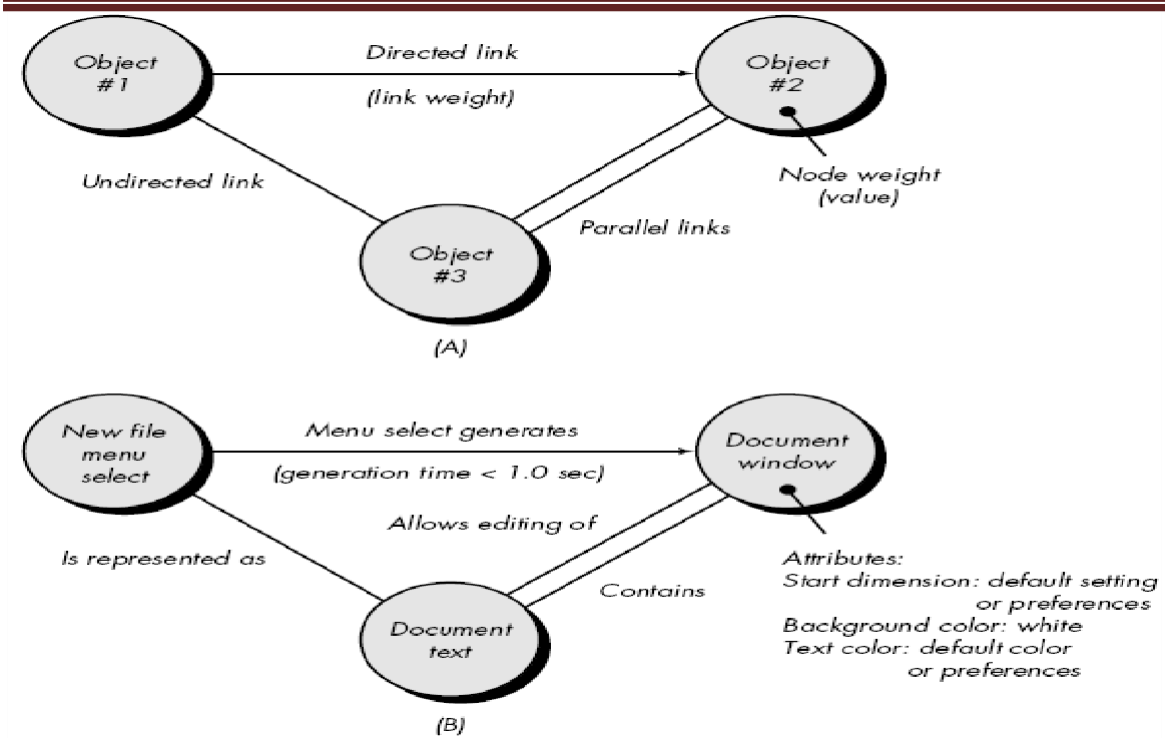


Fig 5.9 (A) Graph notation (B) Simple example

- The symbolic representation of a graph is shown in Fig 5.9A.
- Nodes are represented as circles connected by links that take a number of different forms. A directed link (represented by an arrow) indicates that a relationship moves in only one direction.
- A bidirectional link, called a **symmetric link**, implies that the relationship applies in both directions. Parallel links are used when a number of different relationships are established between graph nodes.
- Eg. consider a portion of a graph for a word-processing application (Fig 5.9B) where
  Object #1 = new file menu select
  Object #2 = document window
  Object #3 = document text
- Referring to the figure, a menu select on new file generates a document window.
- The node weight of document window provides a list of the window attributes that are to be expected when the window is generated.
- The link weight indicates that the window must be generated in less than 1.0 second.
- An undirected link establishes a symmetric relationship between the new file menu select and document text, and parallel links indicate relationships between document window and document text.
- In reality, a far more detailed graph would have to be generated as a precursor to test case design.
- The software engineer then derives test cases by traversing the graph and covering each of the relationships shown. These test cases are designed in an attempt to find errors in any of the relationships.
- Beizer describes a number of behavioral testing methods that can make use of graphs:

- **Transaction flow modeling**. The nodes represent steps in some transaction (e.g., the steps required to make an airline reservation using an on-line service), and the links represent the logical connection between steps (e.g., flight.information.input is followed by validation/availability.processing).
- **Finite state modeling**. The nodes represent different user observable states of the software (e.g., each of the "screens" that appear as an order entry clerk takes a phone order), and the links represent the transitions that occur to move from state to state (e.g., order-information is verified during inventory-availability look-up and is followed by customer-billing-information input). The state transition diagram can be used to assist in creating graphs of this type.
- **Data flow modelin**g. The nodes are data objects and the links are the transformations that occur to translate one data object into another. For example, the node FICA.tax.withheld (FTW) is computed from gross.wages (GW) using the relationship, FTW = 0.62 - GW.
- **Timing modeling**. The nodes are program objects and the links are the sequential connections between those objects. Link weights are used to specify the required execution times as the program executes.

- Graph-based testing begins with the definition of all nodes and node weights. That is, objects and attributes are identified. The data model can be used as a starting point, but it is important to note that many nodes may be program objects (not explicitly represented in the data model). To provide an indication of the start and stop points for the graph, it is useful to define entry and exit nodes.
- Once nodes have been identified, links and link weights should be established.
- In general, links should be named, although links that represent control flow between program objects need not be named.
- Each relationship is studied separately so that test cases can be derived.
- The transitivity of sequential relationships is studied to determine how the impact of relationships propagates across objects defined in a graph. Transitivity can be illustrated by considering three objects, X, Y, and Z. Consider the following relationships:

  X is required to compute Y
  Y is required to compute Z

Therefore, a transitive relationship has been established between X and Z:
X is required to compute Z

- Based on this transitive relationship, tests to find errors in the calculation of Z must consider a variety of values for both X and Y.
- The symmetry of a relationship (graph link) is also an important guide to the design of test cases.
- As test case design begins, the first objective is to achieve node coverage. By this we mean that tests should be designed to demonstrate that no nodes have been inadvertently omitted and that node weights (object attributes) are correct.
- Next, link coverage is addressed. Each relationship is tested based on its properties. For example, a symmetric relationship is tested to demonstrate that it is, in fact, bidirectional. A transitive relationship is tested to demonstrate that transitivity is present.

- A reflexive relationship is tested to ensure that a null loop is present. When link weights have been specified, tests are devised to demonstrate that these weights are valid. Finally, loop testing is invoked (Section 17.5.3).

5.3.2 Equivalence Partitioning

- **Equivalence partitioning** is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.
- An ideal test case single-handedly uncovers a class of errors (e.g., incorrect processing of all character data) that might otherwise require many cases to be executed before the general error is observed.
- Equivalence partitioning strives to define a test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed.
- Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition.
- An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition.
- Equivalence classes may be defined according to the following guidelines:
  1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
  2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
  3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
  4. If an input condition is Boolean, one valid and one invalid class are defined.
- Example, consider data maintained as part of an automated banking application.

The user can access the bank using a personal computer, provide a six-digit password, and follow with a series of typed commands that trigger various banking functions. During the log-on sequence, the software supplied for the banking application accepts data in the form

area code—blank or three-digit number prefix—
three-digit number not beginning with 0 or 1
suffix—four-digit number
password—six digit alphanumeric string
commands—check, deposit, bill pay, and the like

The input conditions associated with each data element for the banking application can be specified as

Area code: Input condition, Boolean—the area code may or may not be present.
Input condition, range—values defined between 200 and 999, with specific exceptions.

Prefix: Input condition, range—specified value >200
Input condition, value—four-digit length

Password: Input condition, Boolean—a password may or may not be present.
Input condition, value—six-character string.

Command: Input condition, set—containing commands noted previously.

Applying the guidelines for the derivation of equivalence classes, test cases for each input domain data item can be developed and executed. Test cases are selected so that the largest number of attributes of an equivalence class are exercised at once.

5.3.3 Boundary Value Analysis

- Boundary value analysis leads to a selection of test cases that exercise bounding values.
- Boundary value analysis is a test case design technique that complements equivalence partitioning.
- Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.
- Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:
    1. If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and just below a and b.
    2. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers.
    3. Apply guidelines 1 and 2 to output conditions.
    4. If internal program data structures have prescribed boundaries (e.g., an array has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.
- By applying these guidelines, boundary testing will be more complete, thereby having a higher likelihood for error detection.

5.3.4 Comparison Testing

- There are some situations (e.g., aircraft avionics, automobile braking systems) in which the reliability of software is absolutely critical.
- In such applications redundant hardware and software are often used to minimize the possibility of error.
- When redundant software is developed, separate software engineering teams develop independent versions of an application using the same specification.
- In such situations, each version can be tested with the same test data to ensure that all provide identical output. Then all versions are executed in parallel with real-time comparison of results to ensure consistency.
- Researchers have suggested that independent versions of software be developed for critical applications, even when only a single version will be used in the delivered computer-based system.
- These independent versions form the basis of a black-box testing technique called **comparison testing** or **back-to-back testing**.
- When multiple implementations of the same specification have been produced, test cases designed using other black-box techniques (e.g., equivalence partitioning) are provided as input to each version of the software.
- If the output from each version is the same, it is assumed that all implementations are correct. If the output is different, each of the applications is investigated to determine

if a defect in one or more versions is responsible for the difference. In most cases, the comparison of outputs can be performed by an automated tool.

- Comparison testing is not foolproof. If the specification from which all versions have been developed is in error, all versions will likely reflect the error. In addition, if each of the independent versions produces identical but incorrect results, condition testing will fail to detect the error.

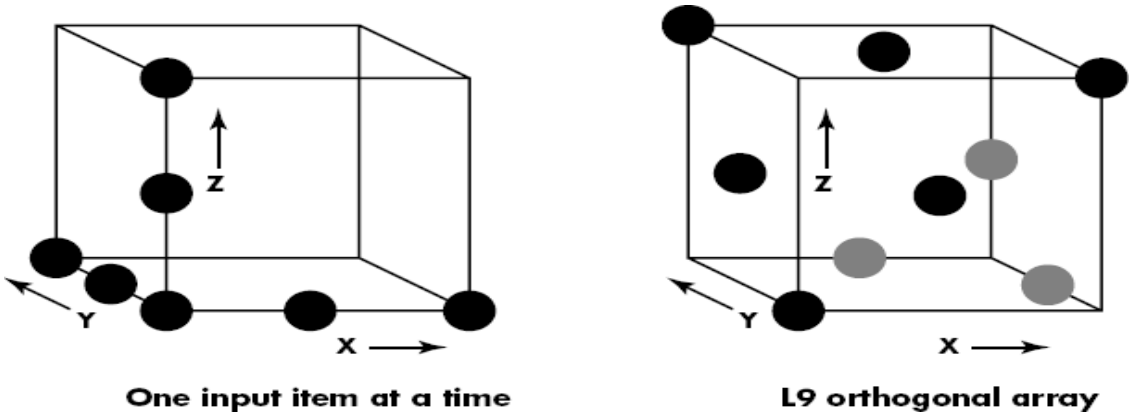


Fig 5.10 A geometric view of test cases

5.3.5 Orthogonal Array Testing

- There are many applications in which the input domain is relatively limited. That is, the number of input parameters is small and the values that each of the parameters may take are clearly bounded. When these numbers are very small, it is possible to consider every input permutation and exhaustively test processing of the input domain.
- However, as the number of input values grows and the number of discrete values for each data item increases, exhaustive testing become impractical or impossible.
- Orthogonal array testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing. T
- he orthogonal array testing method is particularly useful in finding errors associated with region faults—an error category associated with faulty logic within a software component.

5.4 Quality Concepts:

"What is software quality?" Software quality assurance (SQA) is an umbrella activity that is applied throughout the software process.

What is it? It's not enough to tell 'software quality is important', you have to

(1) explicitly define what is meant when you say "software quality,"
(2) create a set of activities that will help ensure that every software engineering work product exhibits high quality,
(3) perform quality assurance activities on every software project,

(4) use metrics to develop strategies for improving your software process and, as a consequence, the quality of the end product.

Who does it? Everyone involved in the software engineering process is responsible for quality.

Why is it important? You can do it right, or you can do it over again. If a software team stresses quality in all software engineering activities, it reduces the amount of rework that it must do. That results in lower costs, and more importantly, improved time-to-market.

What are the steps? Before software quality assurance activities can be initiated, it is important to define 'software quality' at a number of different levels of abstraction. Once you understand what quality is, a software team must identify a set of SQA activities that will filter errors out of work products before they are passed on.

What is the work product? A Software Quality Assurance Plan is created to define a software team's SQA strategy. During analysis, design, and code generation, the primary SQA work product is the formal technical review summary report. During testing, test plans and procedures are produced. Other work products associated with process improvement may also be generated.

How do I ensure that I've done it right? Find errors before they become defects! That is, work to improve your defect removal efficiency, thereby reducing the amount of rework that your software team has to perform.

Concepts

- Variation between samples, applies to all products of human as well as natural creation. For example, if two "identical" circuit boards are examined closely enough, we may observe that the copper pathways on the boards differ slightly in geometry, placement, and thickness. In addition, the location and diameter of the holes drilled in the boards varies as well.
- All engineered and manufactured parts exhibit variation.
- The variation between samples may not be obvious without the aid of precise equipment to measure the geometry, electrical characteristics, or other attributes of the parts.
- However, with sufficiently sensitive instruments, we will likely come to the conclusion that no two samples of any item are exactly alike.
- Variation control is the heart of quality control. A manufacturer wants to minimize the variation among the products that are produced, even when doing something relatively simple like duplicating diskettes.
- Surely, this cannot be a problem—duplicating diskettes is a trivial manufacturing operation, and we can guarantee that exact duplicates of the software are always created. So even a "simple" process such as disk duplication may encounter problems due to variation between samples.
- But how does this apply to software work? How might a software development organization need to control variation? From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time.

- In general, we would like to make sure our testing program covers a known percentage of the software, from one release to another.
- Not only do we want to minimize the number of defects that are released to the field, we'd like to ensure that the variance in the number of bugs is also minimized from one release to another.

Quality:

- The American Heritage Dictionary defines quality as "a characteristic or attribute of something."
- As an attribute of an item, quality refers to measurable characteristics— things we are able to compare to known standards such as length, color, electrical properties, and malleability.
- However, software, largely an intellectual entity, is more challenging to characterize than physical objects.
- These properties include cyclomatic complexity, cohesion, number of function points, lines of code, and many others, discussed in Chapters 19 and 24. When we examine an item based on its measurable characteristics, two kinds of quality may be encountered: quality of design and quality of conformance.
- Quality of design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design.
- As higher-grade materials are used, tighter tolerances and greater levels of performance are specified, the design quality of a product increases, if the product is manufactured according to specifications.
- Quality of conformance is the degree to which the design specifications are followed during manufacturing. Again, the greater the degree of conformance, the higher is the level of quality of conformance.
- In software development, quality of design encompasses requirements, specifications, and the design of the system.
- Quality of conformance is an issue focused primarily on implementation. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high.
- But are quality of design and quality of conformance the only issues that software engineers must consider? Robert Glass [GLA98] argues that a more "intuitive" relationship is in order:
  User satisfaction = compliant product + good quality + delivery within budget and schedule
- DeMarco [DEM99] reinforces this view when he states: "A product's quality is a function of how much it changes the world for the better." This view of quality contends that if a software product provides substantial benefit to its end-users, they may be willing to tolerate occasional reliability or performance problems.

5.4.2 Quality Control

- Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it.
- Quality control includes a feedback loop to the process that created the work product. The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications.
- This approach views quality control as part of the manufacturing process.
- Quality control activities may be fully automated, entirely manual, or a combination of automated tools and human interaction.
- A key concept of quality control is that all work products have defined, measurable specifications to which we may compare the output of each process. The feedback loop is essential to minimize the defects produced.

5.4.3 Quality Assurance

- Quality assurance consists of the auditing and reporting functions of management.
- The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.
- Of course, if the data provided through quality assurance identify problems, it is management's responsibility to address the problems and apply the necessary resources to resolve quality issues.

5.4.4 Cost of Quality

- The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities.
- Cost of quality studies are conducted to provide a base-line for the current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison.
- The basis of normalization is almost always dollars. Once we have normalized quality costs on a dollar basis, we have the necessary data to evaluate where the opportunities lie to improve our processes.
- Furthermore, we can evaluate the effect of changes in dollar-based terms.
- Quality costs may be divided into costs associated with prevention, appraisal, and failure. Prevention costs include
    - quality planning
    - formal technical reviews
    - test equipment
    - training
- Appraisal costs include activities to gain insight into product condition the "first time through" each process. Examples of appraisal costs include
    - In-process and interprocess inspection
    - Equipment calibration and maintenance
    - Testing

- Failure costs are those that would disappear if no defects appeared before shipping a product to customers. Failure costs may be subdivided into internal failure costs and external failure costs. Internal failure costs are incurred when we detect a defect in our product prior to shipment. Internal failure costs include
  - Rework
  - Repair
  - Failure mode analysis
- External failure costs are associated with defects found after the product has been shipped to the customer. Examples of external failure costs are
  - Complaint resolution
  - Product return and replacement
  - Help line support
  - Warranty work
- As expected, the relative costs to find and repair a defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.
- Fig 5.11, based on data collected by Boehm and others, illustrates this phenomenon. Anecdotal data reported by Kaplan, Clark, and Tang reinforces earlier cost statistics and is based on work at IBM's Rochester development facility:

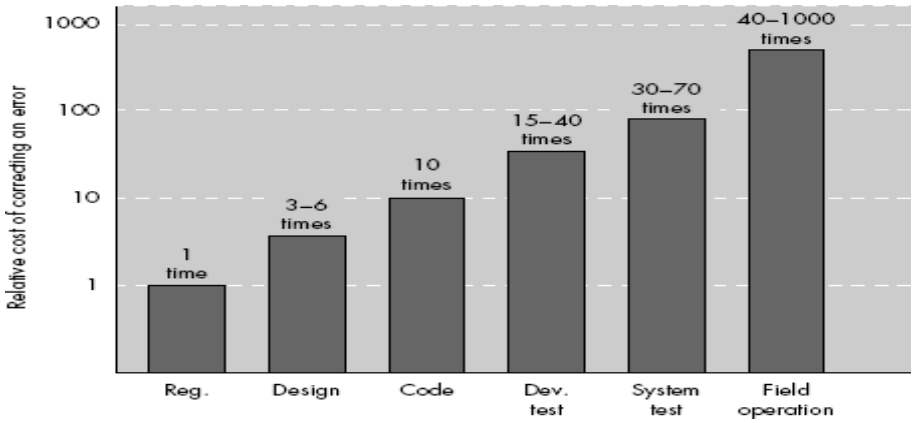


Fig 5.11 Relative cost of correcting an error

A total of 7053 hours was spent inspecting 200,000 lines of code with the result that 3112 potential defects were prevented. Assuming a programmer cost of $40.00 per hour, the total cost of preventing 3112 defects was $282,120, or roughly $91.00 per defect. Compare these numbers to the cost of defect removal once the product has been shipped to the customer. Suppose that there had been no inspections, but that programmers had been extra careful and only one defect per 1000 lines of code [significantly better than industry average] escaped into the shipped product. That would mean that 200 defects would still have to be fixed in the field. At an estimated cost of $25,000 per field fix, the cost would be $5 million, or approximately 18 times more expensive than the total cost of the defect prevention effort.

It is true that IBM produces software that is used by hundreds of thousands of customers and that their costs for field fixes may be higher than those for software organizations that build custom systems. This in no way negates the results just noted.

Even if the average software organization has field fix costs that are 25 percent of IBM's, the cost savings associated with quality control and assurance activities are compelling.

Testing is necessary, but it's also a very expensive way to find errors. Spend time finding errors early in the process and you may be able to significantly reduce testing and debugging costs.

POSSIBLE QUESTIONS

1.define software testing.
2.wnat are the testing fundamentals?
3.note on testing approach.
4.note on testing strategy.
5.define validation testing.
6.define system testing.
7.define whitebox testing.
8.define blackbox testing.
9.compare white box testing & blackbox testing.
10.what are the types of testing in software?

UNIT V

251	_____ is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation.	software specification	software generation	software coding	software testing
252	Software is tested from _____ different perspectives.	2	3	4	5
253	Software engineers are by their nature _____ people.	pessimistic	optimistic	constructive	destructive
254	_____ is a process of executing a program with the intent of finding an error.	coding	testing	debugging	designing
255	All tests should be _____ to customer requirements.	traceable	designed	tested	coded
256	Tests should be planned long before _____ begins.	testing	coding	specification	requirements
257	Testing should begin in the _____ and progress toward testing in the large.	design	beginning	small	big
258	The less there is to test, the more _____ we can test it.	quickly	shortly	automatically	hardly
259	_____ is a process of executing a program with the intend of finding an error.	testing	coding	planning	designing
260	A good _____ is one that has a high probability of finding an as-yet-undiscovered error	planning	test case	objective	goal
261	All _____ should be traceable to customer-requirements.	analysis	designs	tests	plans
262	_____ is simple how easily a computer program can be tested.	software operability	software simplicity	software decomposability	software testability
263	The better it works, the more efficiently it can be testing. This characteristic is called _____.	operability	observability	controllability	decomposability
264	There are _____ characteristics in testability	5	6	7	8
265	What you see is what you test. This characteristic is called _____.	controllability	observability	decomposability	stability
266	The better we can control the software, the more the testing can be automated and optimized. This characteristic is called _____.	operability	stability	understandability	controllability
267	By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting. This characteristic is called _____.	decomposability	simplicity	stability	understandability
268	. The less there is to test, the more quickly we can test it. This characteristic is called _____.	controllability	simplicity	operability	observability
269	The fewer the changes, the fewer the disruptions to testing. This characteristic is called _____.	controllability	decomposability	stability	understandability

270	. The more information we have, the smarter we will test. This characteristic is called _____.	controllability	decomposability	stability	understandability
271	A good test has a high _____ of finding an error.	probability	simplicity	understandability	stability
272	A good test is not _____.	stable	redundant	simple	complex
273	White-box testing sometimes called _____.	control structure testing	condition testing	glass-box testing	black-box testing
274	Logic errors and incorrect assumptions are inversely proportional to the _____ that a program path will be executed	simplicity	probability	understandability	stability
275	Typographical errors are _____.	redundant	simple	random	complex
276	One often believes that a _____ path is not likely to be executed when, in fact, it may be executed on a regular basis.	control	structural	physical	logical
277	Basic path testing is a _____.	black-box testing	white-box testing	control structure testing	control path testing
278	_____ is a software metric that provides a quantitative measure of the logical complexity of a program.	cyclomatic complexity	flow graph	deriving test cases	graph matrices
279	An _____ is any path through the program that introduces atleast one new set of processing statements or a new condition.	dependent path	independent path	basic path	control path
280	There are _____ steps to be applied to derive the basis set.	2	3	4	5
281	There are _____ test cases that satisfy the basis set.	3	4	5	6
282	. A _____ is a square matrix whose size is equal to the number of nodes on the flow graph.	graph matrix	matrix	flow graph	cyclomatic complexity
283	To develop a software tool that assists in basis path testing, a data structure called a _____ is useful.	matrix	flow graph	graph matrix	cyclomatic omplexity
284	_____ requires three or four tests to be derived for a relational expression.	branch testing	data flow testing	data control testing	domain testing
285	_____ is probably the simplest condition testing strategy.	branch testing	data flow testing	condition testing	domain testing
286	The _____ method selects test paths of a program according to the locations of definitions and uses of variables in the program	data flow testing	condition testing	loop testing	black box testing
287	_____ is a white box testing technique that focuses exclusively on the validity of loop constructions	data flow testing	loop testing	condition testing	control path testing
288	_____ is a test case design method that exercises the logical conditions contained in a program module	black box testing	loop testing	data flow testing	condition testing
289	_____ is called behavioral testing.	black box testing	loop testing	data flow testing	condition testing
290	The first step in _____ is to understand the objects that are modeled in software and the relationships that connect these objects	black box testing	loop testing	data flow testing	condition testing
291	Equivalence partitioning is a _____ method that divides the input domain of a program into classes of data.	black box testing	loop testing	data flow testing	condition testing

292	Comparison testing is also called _____.	black box testing	loop testing	behavioral testing	back-to-back testing
293	_____ testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.	orthogonal array	loop	behavioral	back-to-back
294	_____ focuses verification effort on the smallest unit of software design – the software component or module.	module testing	unit testing	structure testing	system testing
295	A driver is nothing more than a _____.	subprogram	main program	stub	subroutine
296	_____ serve to replace modules that are subordinate called by the component to be tested.	subprograms	main programs	stubs	subroutines
297	Drivers and _____ represent overhead.	subprograms	main programs	stubs	subroutines
298	_____ of execution paths is an essential task during the unit test.	unit testing	module testing	selective testing	white box testing
299	Good _____ dictates that error conditions be anticipated and error-handling paths set up to reroute or cleanly terminate processing when an error does occur	design	testing	code	module
300	_____ is completely assembled as a package, interfacing errors have been uncovered and corrected.	software	program	code	all of the above

software testing
2
constructive

testing
traceable
testing

small
quickly

testing

test case
tests
software
testability

operability
7

observability

controllability

decomposability

simplicity

stability

understandability
probability
redundant

glass-box testing

probability
random

logical

white-box testing
cyclomatic
complexity

independent path
4
6

graph matrix

graph matrix

domain testing
branch testing

data flow testing

loop testing

condition testing
black box testing

black box testing

black box testing

back-to-back
testing

orthogonal array

unit testing
main program

stubs
stubs

selective testing

design

software

Unit - 2

1	The _____ as a bridge between the systm decription and the design model	design	analysis model
2	The role of the software engineer in the requirement analysis is called____	designer	analyst
3	analysis modeling often begins with _____	data modeling	function modeling
4	A _____ can be an external entity	function object	structural object
5	_____ defines the properities of a data object	relationship	cardinality
6	Data objects are connected to one another in different ways is called _____	modality	relationships
7	_____ is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object	modality	relationships
8	_____ defines the maximum number of objects that can participate in a relationship	modality	relationships
9	_____ provide an indication of whether or not a particular data object must participate in the relationship	Modality	relationships
10	The _____ diagram takes an an input-process-output view of sysytem	use-case diagram	activity diagram
11	The level 0 DFD is called as _____ diagram	contract level diagram	context level diagram
12	The _____ describes the behavior of the system but not the inner working of the processes	PSPEC	ASPEC
13	The _____ is used to describe all flow model processes the appear in the final level of refinement	CSPEC	ASPEC
14	The _____ model indicates how software will respond to external events	data	behavior
15	The _____ represents a sequence of activities that involves actor and the system	csase tool	activity
16	The _____ diagram indicates how events cause transitions from object to object	sequence diagram	activity
17	Which one depict the software requirements from the user's point of view.	behavioral based	flow based model
18	Which model depicts how input is transformed into output as data objects move through a system	behavioral based	flow based model
19			
20			

Unit 3

1	_____is a iterativeprocess through which requirements are translated into a blueprint for constructing the software	requirements gathering	coding
2	Who developeda set of software quality attributefor the software design	Barry Boehm	R.Pattis
3	Which quality attribute measure the response time, throughput and effeciency of the sysytem	Functionality	Usability

4	The quality attribute, Usability is assessed by considering the overall _____ of the system	consistency	Functionality
5	A _____ refers to a sequence of instructionsthat have a specific and limited functions	procedural abstraction	data abstraction
6	_____ represent architecture as an organized collection of programs components	process models	structural models
7	_____ models address the behavioral aspects of the program architecture	process models	structural models
8	Software is divided into separately named and addressable components is called _____	process	behavior
9	the _____ is a process of changing a software by which doesnot alter the external behavior of the code	refinement	cohesion
10	_____ is an indication of the relative functional strength of the module	refinement	cohesion
11	_____ is an indication of the relativeinterdependency among modules	cohesion	patterns
12	Refinement is a top-down design strategy which is actually a process of _____	eloboration	abstraction
13	A _____ is a named collection of data that describes a data object.	procedural abstraction	data abstraction
14	_____ implies a program control mechanism without specifying internal detail.	procedural abstraction	data abstraction
15	software architecture consider ____ levels of the design pyramid	3	2
16	Which action translates data objects into data structures	data design	component design
17	In data centered arcjitecture _____ resides at the centre of the architecture which is accessed frequently by other components	client software	data store
18	_____ represents the structure of data and program components that are required to build a computer-based syste,	architectural design	data design
19	KDD stand for	Knowledge Discovery of data manipulatio	Knowledge Discovery in database
20	the _____ classes defines all abstraction that are necessary for human computer interaction	primitive class	user interface
21	The _____ classes implement lower level business abstraction required to manage the business domain class	primitive class	user interface
22	_____ suggest that a method should send or receive messages from friend class	Law of cohesion	Law of meter
23	_____ is achieve by developing modules with single minded function and aversion of excessive interaction	refinement	refactoring
24	_____ suggest that the information contained in one module is inaccesible to othe modules	refinement	refactoring
25	Refinement is a process of _____	abstraction	eloboration
26	_____ is a process of breaking up of complex problem into a manageable piecies	refinement	refactoring
27	_____ is evaluated by measuring the frequency and severity of failure.	Usability	performance

28	iIn transform flow the information must entered and exit in _____ form	external world	internal world
29	Information flow is characterized by an single data item is called	context flow	transaction flow
30	A _____ diagram is mapped into a program structure using transform or transaction mapping	data flow	use case
31	_____ language provides a semantic and syntax for describing a software architecture	architectural description	architectural design
32	Design begins with the _____ model.	data	requirements
33	_____ focus on the design of the business or technical process that the system must accommodate.	framework models	dynamic models
34	_____ can be used to represent the functional hierarchy of a system.	framework models	dynamic models
35	_____ represent architecture as an organized collection of program components.	dynamic models	functional models
36	_____ increases the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.	framework models	dynamic models
37	_____ address the behavioural aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.	framework models	dynamic models
38	_____ is the place where quality is fostered in software engineering	model	data
39	_____ provides us with representations of software that can be assessed for quality.	design	specification
40	_____ describes a program should not have any bugs that inhibit its function	firmness	commodity
41	A program should be suitable for the purposes for which it is intended is called	firmness	commodity
42	The experience of using the program should be a pleasurable one is called	firmness	commodity
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			

planning	construction	analysis model
programmer	tester	analyst
behavior modling	structure modeling	
Data object	flow object	Data object
Data attributes	modality	Data attributes
cardinality	Data attributes	relationships
cardinality	Data attributes	Cardinality
Data attributes	cardinality	Cardinality
Data attributes	cardinality	Modality
data flow diagram	ERD	data flow diagram
text level diagram	zero level diagram	context level diagram
LSPEC	CSPEC	CSPEC
LSPEC	PSPEC	PSPEC
function	structural	behavior
use-case	swimlane	use-case
use-case	swimlane	sequence diagram
class based	scenario based model	scenario based model
class based	scenario based model	flow based model
software design	deployment	software design
M.C.Escher	Hewlett-Packard	Hewlett-Packard
Performance	Supportability	Performance

Supportability	Performance	consistency
behavior abstraction	structural abstraction	procedural abstraction
dynamic models	framework models	structural models
dynamic models	framework models	dynamic models
modules	data	Modules
patterns	refactoring	refactoring
patterns	refactoring	cohesion
coupling	functional dependency	coupling
refactoring	information hiding	eloboration
control abstraction	behavior abstraction	data abstraction
control abstraction	behavior abstraction	control abstraction

1	4	2
behavior design	functional design	data design
filter	pipes	data store
software design	behavioural design	architectural design
Knowing of database discovery process	Knowing discovery of database business	Knowledge Discovery in database
classes	domain	user interface
process classes	System class	process classes
Low of completeness	law of primitiveness	Law of meter
functional independence	information hiding	functional independence
functional independence	information hiding	information hiding
architecture	modularity	eloboration
modularity	arichiteture	modularity
supportability	reliability	reliability

top down	bottom up	external world
transform flow	contract flow	transaction flow
state diagram	activity diagram	data flow
architectural pattern	architecturaldefinition	architectural description
specification	code	requirements
process models	functional models	process models
process models	functional models	functional models
framework models	structural models	structural models
process models	functional models	framework models
process models	functional models	dynamic models
design	specification	design
data	prototype	design
delight	roman	firmness
delight	roman	commodity
delight	roman	delight

Register Number_____

[18CAU402]

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
BCA DEGREE EXAMINATION
(For the candidates admitted from 2018 onwards)
Fourth Semester
SOFTWARE ENGINEERING
First Internal Examination December 2019

Date & Session : 18.12.19 & AN
Duration: 2 Hours

Class II BCA (A & B)
Maximum : 50 Marks

PART-A (20 * 1 = 20Marks)
Answer ALL the questions

1.Software takes on a_____role.

a. Single b. Dual c. Triple d.Tetra

2.Software is a_____.

a. Virtual b. System c. Modifier d.Framework

3.Instructions that when executed provide desired function and performance is called_____.

a. Software b. Hardware c. Firmware d. Humanware

4.High quality of software is achieved through_.

a.Testing b.Good Design c.Construction d.Manufacture

5.Software doesn't

a. Tearout b.Wearout c.Degrade d.Deteriorate

6. Software is not susceptible to_____.

a.Hardware b.Defects c. Environmental Melodies d. Deterioration

7. Software will undergo_____.

a.Database b.Testing c.Enhancement d.Manufacture

8._____refers to the meaning and form of incoming and outgoing information.

a. Content b.Software c.Hardware d.Data

9._____ refers to the predictability of the order and timing of information.

a.System Software b.Network Software c.Information Determinacy d.Database

10._____is not a system software.

a.MS Office b.Compiler c.Editor d.FileManagement Utility

11._____is a process of discovery, refinement, modeling, and specification.

a.Software Engineering b.Software Requirement c.Software Analysis d.Software Design

12. _____is the systematic use of proven principles, techniques, languages, and tools.

a.Software Engine b.Software Analysis c. Software Design d.Requirements Engineering

13. Requirement engineering is conducted in a_.

a.Sporadic Way b.Random Way c.Haphazard Way d.Systematic Approaches

14. Software requirements analysis work products must be reviewed for_____.

a.Modeling b.Completeness c.Information Processing d.Functional Requirement

15.___bridges the gap between system level requirement engineering and software design.

a.System Engineering b.Modeling c.Requirements Analysis d.Software Engineering

16.A ____can be an external entity.

a.Function Object b.Structural Object c.Data Object d.Flow Object

17.The_____as a bridge between the systmdecription and the design model.

a.Design b.Analysis Model c.Planning d.Construction

18.Software applications can be collectively called as_____.

a. Data Gathering b. Information Gathering c. Data Processing d. Information Processing

19. _____ represents the individual data and control objects that constitute some larger collection of information transformed by the software

a. Information content b.Data content c. Data model d. Information

20.___represents the manner in which data and control change as each moves through a system.

a.Information content b.Information flow c.Information structure d.Data structure

Part – B(3X 2= 6 Marks)
Answer ALL Questions

21. Define Software.
22. Define Software Engineering.
23. What is meant by Requirement Analysis?

Part – C (3X 8= 24 Marks)
Answer ALL Questions

24. a.Write a note on Applications of Software Engineering.

(Or)

b.Write briefly about the Nature of Software.

25. a. Brief about Waterfall method.

(Or)

b. Write note on Spiral Model.

26. Explain about the Types of Requirement Analysis.

(Or)

b. Explain about the Software Layered Technology.

Reg.No ----------

[18CAU402]

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956)

COIMBATORE – 641 021

BCA DEGREE EXAMINATION

(For the candidates admitted in 2018 onwards)

Fourth Semester

First Internal Examination – December 2019

SOFTWARE ENGINEERING

Time : 2 Hours **Maximum : 50Marks**

DATE & SESSION: **18-12-2019 & AN** **Class : II BCA A & B**

PART-A (20 * 1 = 20Marks)

Answer ALL the questions

1.Software takes on a_____role.

a.Single **b.Dual** c.Triple d.Tetra

2.Software is a_____.

a.Virtual b.System **c.Modifier** d.Framework

3.Instructions that when executed provide desired function and performance is called_____.

a.Software b.Hardware c.Firmware d.Humanware

4.High quality of software is achieved through.

a.Testing **b.Good Design** c.Construction d.Manufacture

5.Software doesn't .

a.Tearout **b.Wearout** c.Degrade d.Deteriorate

6.Software is not susceptible to_____.

a.Hardware b.Defects c.Environmental Melodies d.Deterioration

7.Software will undergo_____.

a.Database b.Testing **c.Enhancement** d.Manufacture

8. _____refers to the meaning and form of incoming and outgoing information.

a.**Content** b.Software c.Hardware d.Data

9. _____refers to the predictability of the order and timing of information.

a.System Software b.Network Software **c.Information Determinacy** d.Database

10._is not a system software.

a.MS Office b.Compiler c.Editor d.FileManagement Utility

11. is a process of discovery, refinement, modeling, and specification.

a.Software Engineering b.Software Requirement c.Software Analysis d.Software Design

12._____is the systematic use of proven principles, techniques, languages, and tools.

a.Software Engineering	b.Software Analysis	c.Software Design	**d.Requirements Engineering**

13.Requirement engineering is conducted in a_____.

a.Sporadic Way b.Random Way c.Haphazard Way **d.Systematic Approaches**

14.Software requirements analysis work products must be reviewed for_____.

a.Modeling **b.Completeness** c.Information Processing d.Functional Requirement

15._bridges the gap between system level requirement engineering and software design.

a.System Engineering b.Modeling c.Requirements Analysis **d.Software Engineering**

16.A _____can be an external entity.

a.Function Object b.Structural Object **c.Data Object** d.Flow Object

17.The_____as a bridge between the systmdecription and the design model.

a.Design **b.Analysis Model** c.Planning d.Construction

18.Software applications can be collectively called as_____.

a.Data Gathering	b.Information Gathering	**c.Data Processing**	d.Information Processing

19._____represents the individual data and control objects that constitute some larger collection of information transformed by the software.

a.Information content b.Data content c. Data model d. Information model

20._represents the manner in which data and control change as each moves through a system.

a.Information content **b.Information flow** c.Information structure d.Data structure

Part – B(3X 2= 6 Marks)
Answer ALL Questions

21.Define Software.

Software, in its most general sense, is a set of instructions or programs instructing a computer to do specific tasks. **Software** is a generic term used to describe computer programs. Scripts, applications, programs and a set of instructions are all terms often used to describe **software**.

22.Define Software Engineering.

The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software

23.What is meant by Requirement Analysis?

Requirements analysis, also called **requirements** engineering, is the process of determining user expectations for a new or modified product. ... In software engineering, such **requirements** are often called functional specifications.**Requirements analysis** is an important aspect of project management.

Part – C (3X 8= 24 Marks)
Answer ALL Questions

24. a. Write note on Applications of Software Engineering.

Different types of application software include:

- Application Suite: Has multiple applications bundled together. Related functions, features and user interfaces interact with each other.
- Enterprise Software: Addresses an organization's needs and data flow in a huge distributed environment
- Enterprise Infrastructure Software: Provides capabilities required to support enterprise software systems
- Information Worker Software: Addresses individual needs required to manage and create information for individual projects within departments
- Content Access Software: Used to access content and addresses a desire for published digital content and entertainment
- Educational Software: Provides content intended for use by students
- Media Development Software: Addresses individual needs to generate and print electronic media for others to consume

b.Write briefly about the Nature of Software.

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

(a) Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

(b) Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

(c) Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability

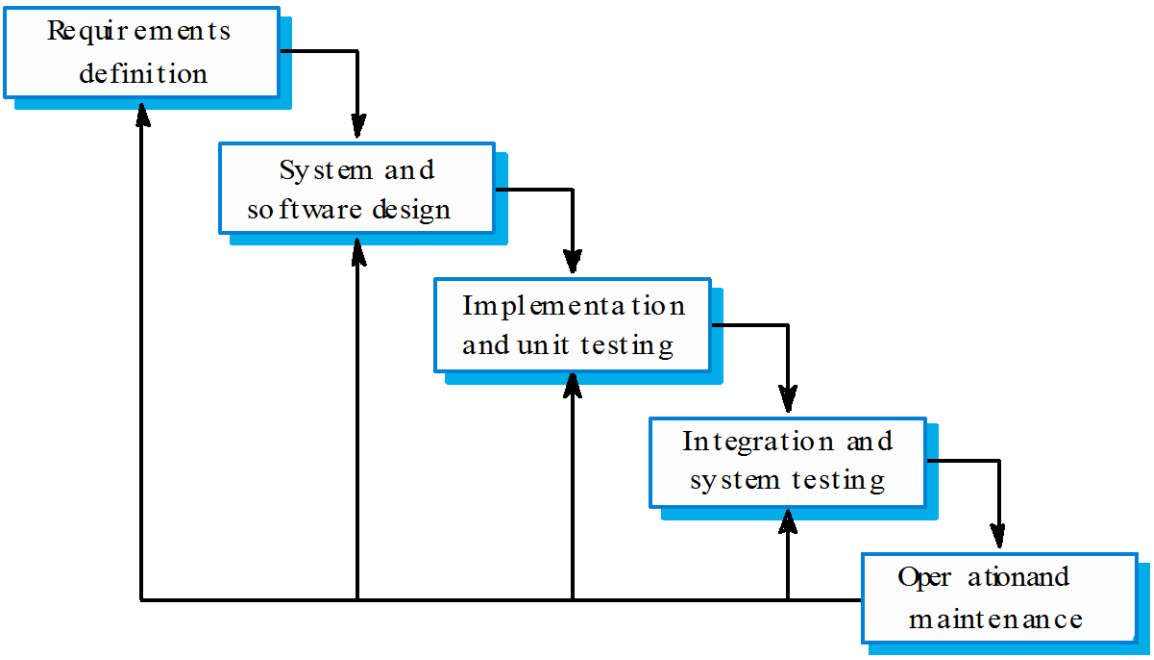
- Flexibility
- Scalability

25. a.Brief about Waterfall Method.

It is a Generic software process models

- The waterfall model separate and distinct phases of specification and development.

Waterfall model



Waterfall Model

The classic software life cycle is often represented as a simple prescriptive waterfall software phase model, where software evolution proceeds through an orderly sequence of transitions from one phase to the next in order (Royce 1970). Such models resemble finite state machine descriptions of software evolution.

However, these models have been perhaps most useful in helping to structure, staff, and manage large software development projects in complex organizational settings, which was one of the primary purposes (Royce 1970, Boehm 1976).Alternatively, these classic models have been widely characterized as both poor descriptive and prescriptive models of how software development "in-the-small" or "in-the-large" can or should occur. Figure 1 provides a common view of the waterfall model for software development attributed to Royce (1970).

Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.

Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

b. Write note on Spiral Model.

The Spiral Model

Spiral Model: The spiral model is an evolutionary software process model that combines the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions. Given figure is of a spiral model that contains five task regions.

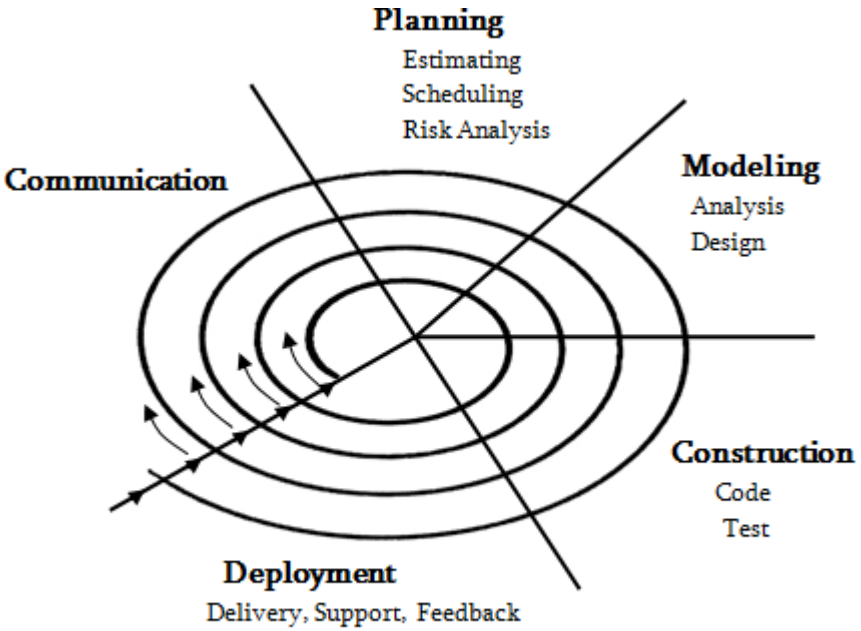


Figure : Spiral Model

Model that contains five task regions.

(i) Customer communication — Tasks required to establish effective communication

between developer and customer.

(ii) Planning — Tasks required to define resources, timelines, and other project related information.

(iii) Modeling — Tasks required in building one or more representations of the application.

(iv) Construction and release — Tasks required to construct, test, install.

(v) Deployment — Tasks required to deliver the software, getting feedbacks etc.

Software engineering team moves around the spiral in a clockwise direction, beginning at the center. The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from customer evaluation.

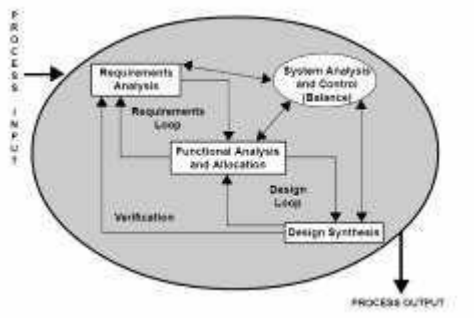
In addition, the project manager adjusts the planned number of iterations required to complete the software.

The spiral model is a realistic approach to the development of large- scale systems and software. The spiral model enables the developer to apply the prototyping approach at any stage in the evolution of the product. The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic. It demands considerable risk assessment expertise and relies on this expertise for success.

26.a. Explain about the Types of Requirement Analysis.

In systems engineering and software engineering, **requirements analysis** encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, *analyzing, documenting, validating and managing*software or system requirements.[2]

Requirements analysis is critical to the success or failure of a systems or software project.[3] The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.



Customer requirements

Statements of fact and assumptions that define the expectations of the system in terms of mission objectives, environment, constraints, and measures of effectiveness and suitability

(MOE/MOS). The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer. Operational requirements will define the basic need and, at a minimum, answer the questions posed in the following listing:[1]

- *Operational distribution or deployment*: Where will the system be used?
- *Mission profile or scenario*: How will the system accomplish its mission objective?
- *Performance and related parameters*: What are the critical system parameters to accomplish the mission?
- *Utilization environments*: How are the various system components to be used?
- *Effectiveness requirements*: How effective or efficient must the system be in performing its mission?
- *Operational life cycle*: How long will the system be in use by the user?
- *Environment*: What environments will the system be expected to operate in an effective manner?

Architectural requirements

Architectural requirements explain what has to be done by identifying the necessary systems architecture of a system.

Structural requirements

Structural requirements explain what has to be done by identifying the necessary structure of a system.

Behavioral requirements

Behavioral requirements explain what has to be done by identifying the necessary behavior of a system.

Functional requirements

Functional requirements explain what has to be done by identifying the necessary task, action or activity that must be accomplished. Functional requirements analysis will be used as the toplevel functions for functional analysis.[1]

Non-functional requirements

Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviors.

Performance requirements

The extent to which a mission or function must be executed; generally measured in terms of quantity, quality, coverage, timeliness or readiness. During requirements analysis, performance (how well does it have to be done) requirements will be interactively developed across all identified functions based on system life cycle factors; and characterized in terms of
the degree of certainty in their estimate, the degree of criticality to system success, and their relationship to other requirements.[1]

Design requirements

The "build to", "code to", and "buy to" requirements for products and "how to execute" requirements for processes expressed in technical data packages and technical manuals.[1]

Derived requirements

Requirements that are implied or transformed from higher-level requirement. For example, a requirement for long range or high speed may result in a design requirement for low weight.[1]

Allocated requirements

A requirement that is established by dividing or otherwise allocating a high-level requirement into multiple lower-level requirements. Example: A 100-pound item that consists of two subsystems might result in weight requirements of 70 pounds and 30

pounds for the two lower-level items

26.b.Write note on Software Layered Technology.

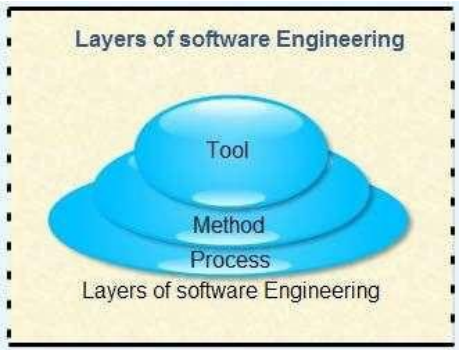
Software Engineering Layers

Software engineering can be viewed as a layered technology. Various layers are listed below.

The **process layer** allows the development of software on time. It defines an outline for a set of key process areas that must be acclaimed for effective delivery of software engineering technology.

The **method layer** provides technical knowledge for developing software. This layer covers a broad array of tasks that include requirements analysis, design, coding, testing, and maintenance phase of the software development.

The **tools layer** provides computerized or semi-computerized support for the process and the method layer. Sometimes tools are integrated in such a way that other tools can use information created by one tool. This multi-usage is commonly referred to as **Computer-Aided Software Engineering (CASE).**CASE combines software, hardware, and software engineering database to create software engineering analogous to **Computer-Aided Design (CAD) for** hardware. CASE helps in application development including analysis, design, code generation, and debugging and testing. This is possible by using CASE tools, which provide automated methods for designing and documenting traditional-structure programming techniques. For example, two prominent technologies using CASE tools are PC-based workstations and application generators that provide graphics-based interfaces to automate the development process.



Concepts of data modeling

- Analysis modeling starts with the data modeling.
- The software engineer defines all the data object that proceeds within the system and the relationship between data objects are identified.

Data objects

- The data object is the representation of composite information.
- The composite information means an object has a number of different properties or attribute.

For example, Height is a single value so it is not a valid data object, but dimensions contain the

height, the width and depth these are defined as an object.

Data Attributes

Each of the data object has a set of attributes.

Data object has the following characteristics:

- Name an instance of the data object.
- Describe the instance.
- Make reference to another instance in another table.

Relationship

Relationship shows the relationship between data objects and how they are related to each other.

Cardinality

Cardinality state the number of events of one object related to the number of events of another object.

The cardinality expressed as:

One to one (1:1)

One event of an object is related to one event of another object.
For example, one employee has only one ID.

One to many (1:N)

One event of an object is related to many events.
For example, One collage has many departments.

Many to many(M:N)

Many events of one object are related to many events of another object.
For example, many customer place order for many products.

Modality

- If an event relationship is an optional then the modality of relationship is zero.
- If an event of relationship is compulsory then modality of relationship is one.