UNIT-III
SOFTWARE DESIGN

3.1 DESIGNPROCESS

- Software design consists of the set of principles, concepts, and practices which is used for developing the system orproduct
- Software design is an iterative process through which requirements are translated into a blueprint for constructing thesoftware

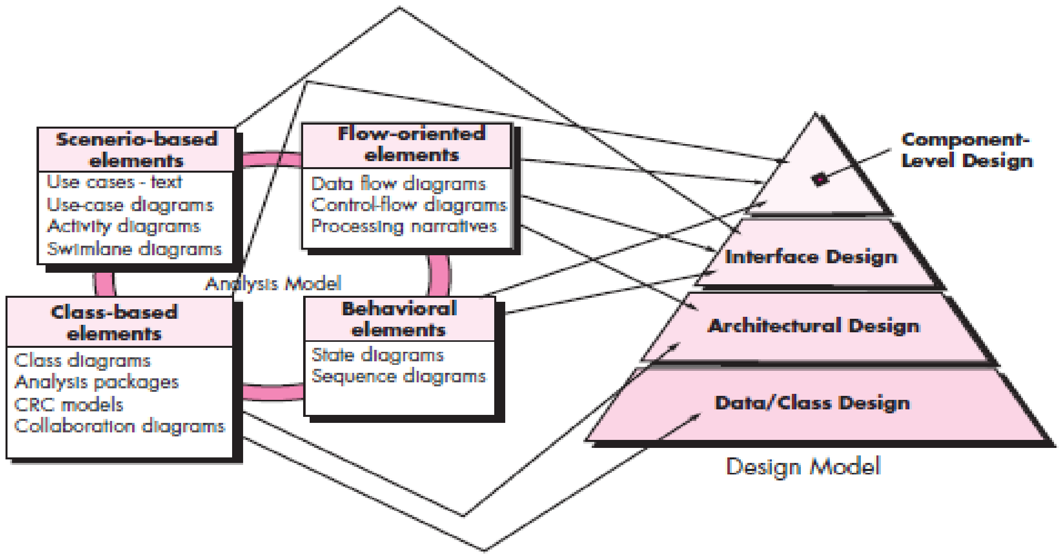


Fig: Translating the Requirements model into Design model

- Initially the design is represented at a high level of abstraction and as iteration occurs, the design shows lower levels ofabstraction.
- The design model provides detail about the software data structures, architecture, interfaces, andcomponents

Characteristics of a good design

- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by thecustomer.
- The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support thesoftware.
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementationperspective.

Quality Guidelines

- A design should exhibit an architecture that is created using recognizable architectural styles or patterns and composed of components that exhibit good design characteristics and can be implemented in an evolutionaryfashion
- A design should be modular
- A design should contain distinct representations of data, architecture, interfaces, and components.
- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable datapatterns.
- A design should lead to components that exhibit independent functionalcharacteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the externalenvironment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirementsanalysis.
- A design should be represented using a notation that effectively communicates its meaning.

Design Principles

- The design process should not suffer from 'tunnelvision.'
- The design should be traceable to the analysismodel.
- The design should exhibit uniformity andintegration.
- The design should be structured to accommodatechange.
- The design should be structured to degrade gently, even when aberrant data, events, or operating conditions areencountered.
- Design is not coding, coding is notdesign.
- The design should be assessed for quality as it is being created, not after thefact.
- The design should be reviewed to minimize semanticerrors.

Quality Attributes. [FURPS]

- **Functionality** is assessed by evaluating the feature set and capabilities of the program and the securityof the overall systemetc.
- **Usability** is assessed by considering human factors, consistency anddocumentation.

- **Reliability** is evaluated by measuring the frequency and severity of failure, the mean-time-to-failure (MTTF), the ability to recover from failureetc.
- **Performance** is measured by considering processing speed, response time, resource consumption, throughput, andefficiency.
- **Supportability** combines the ability to extend the program (extensibility), adaptability, serviceability

The Evolution of Software Design

- Early design work concentrated on criteria for the development of modular programs and methods for refining software structures in a top downmanner.
- Procedural aspects of design definition evolved into structuredprogramming
- Later work proposed methods for the translation of data flowinto a designdefinition.
- Newer design approaches proposed an object-oriented approach to designderivation.
- The design patterns can be used to implement software architectures and lower levels of designabstractions.
- Aspect-oriented methods, model-driven development, and test-driven development emphasize techniques for achieving more effective modularity and architectural structure in the designs.

Characteristics of design methods:

(1) A mechanism for the translation of the requirements model into a design representation,

(2) A notation for representing functional components and theirinterfaces,

(3) Heuristics for refinement and partitioning,and

(4) Guidelines for qualityassessment.

3.2 DESIGNCONCEPTS

Concepts that has to be considered while designing the software are:

Abstraction, Modularity, Architecture, pattern, Functional independence, refinement, information hiding, refactoring and design classes

3.2.1 Abstraction

- There are many levels ofabstraction
- At the highest level of abstraction, a solution is stated in broad terms and at lower levels of abstraction, a more detailed description of the solution isprovided.

Types:

- **Procedural abstraction** refers to a sequence of instructions that have a specific and limitedfunction
  - Example: open for a door. Open implies a long sequence of procedural steps (e.g., walk to the door, reach out and grasp knob, turn knob and pull door, step away from moving door,etc.).
- **Data abstraction** is a named collection of data that describes a data object
  - Eg: Data abstraction for door would encompass a set of attributes that describe the door (e.g., door type, swing direction, opening mechanism, weight,dimensions).

3.2.2 Architecture

- Software architecture is the overall structure of the software and the ways in which that structure provides conceptual integrity for asystem

Properties:

- **Structural properties.** This representation defines the components of a system and how they interact with oneanother.
- **Extra-functional properties.** The architectural design should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability.
- **Families of related systems.** The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similarsystems

Models to represent the Architectural design:

- Structural models represent architecture as an organized collection of program components.
- Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of applications.
- Dynamic models address the behavioural aspects of the program architecture, indicating how the structure changes when an external eventoccurs.
- Process models focus on the design of the business or technical process that the system mustaccommodate.
- Functional models can be used to represent the functional hierarchy of asystem.

- A number of different architectural description languages (ADLs) have been developed to represent thesemodels

3.2.3 Patterns

- Design pattern is a design structure that solves a particular design problem within a specific context
- It provides a description to determine whether the pattern is applicable, whether the pattern can be reused, and whether the pattern can serve as a guide for developing similar patterns

3.2.4 Separation ofConcerns

- Any complex problem can be more easily handled by subdividing it into pieces which can be solvedindependently
- A concern is a feature or behavior that is specified as part of the requirements model for the software
- By separating concerns into smaller manageable pieces, a problem takes less effort and time tosolve.

3.2.5 Modularity

- Modularity is the single attribute of software that allows a program to <u>be intellectually manageable</u>
- Module is a separately named and addressable components that are integrated to satisfy requirements (divide and conquerprinciple)

3.2.6 InformationHiding

- Information hiding is that the algorithms and local data contained within a module are inaccessible to othermodules

3.2.7 FunctionalIndependence

- Functional independence is achieved by developing modules that have a "single-minded" function having less interaction with othermodules
- Independent modules are easier to maintain and test and error propagation is also reduced.
- Independence is assessed using two qualitative criteria: **cohesion andcoupling**.
- *Cohesion* is an indication of the relative functional strength of amodule.
- *Coupling* is an indication of the relative interdependence amongmodules.

- A module should have **high cohesion and lowcoupling**
- High cohesion – a module performs only a singletask

- Low coupling – a module has the lowest amount of connection needed with other modules

3.2.8 Refinement

- Refinement is a process of*elaboration*
- It is the development of a program by successively refining levels of proceduredetail
- This complements abstraction, which enables a designer to specify procedure and data and suppress low-leveldetails
- Refinement helps to reveal low-level details as designprogresses.

3.2.9 Aspects

- An *aspect* is a representation of a cross-cuttingconcern
- These concerns include requirements, use cases, features, data structures, quality-of- service issues, variants, intellectual property boundaries, collaborations, patterns and contracts

3.2.10 Refactoring

- Refactoring is a reorganization technique that simplifies the design of a component without changing its function or externalbehaviour
- It removes redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other designfailures

3.2.11 Object-Oriented DesignConcepts

- The object-oriented (OO) paradigm is widely used in modern softwareengineering.
- Some of the OO design conceptsare
  - Designclasses
  - Inheritance
  - Message passing
  - Polymorphismetc

3.2.12 DesignClasses

- Design classes refines the analysis classes by providing design detail that will enable the classes to beimplemented
- Creates a new set of design classes that implement a software infrastructure to support the businesssolution

Types of Design Classes

- **User interface classes** – define all abstractions necessary for human-computer interaction
- **Business domain classes** – refined from analysis classes; identify attributes and methods

that are required to implement some element of the businessdomain

- **Process classes** – implement business abstractions required to fully manage the business domainclasses
- **Persistent classes** – represent data stores (e.g., a database) that will persist beyond the execution of thesoftware
- **System classes** – implement software management and control functions that enable the system to operate and communicate within its computing environment and the outside world

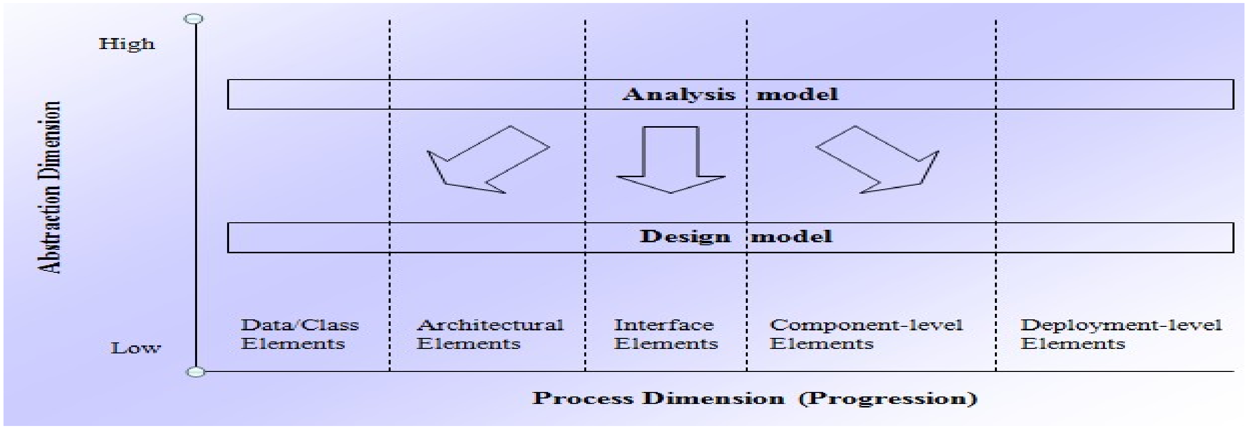
Characteristics of a Well-Formed Design Class

- Complete andsufficient
- Primitiveness - Each method of a class focuses on accomplishing one service for theclass
- Highcohesion
- Low coupling

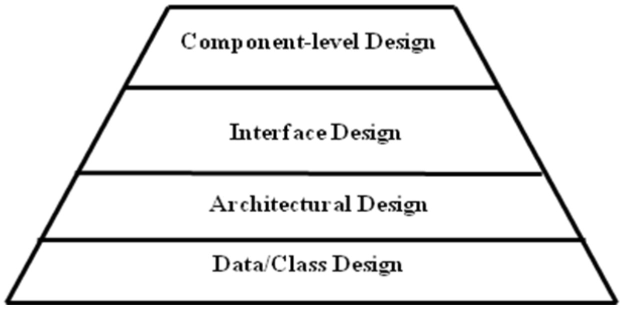
3.3 DESIGNMODEL

- The design model can be viewed in two differentdimensions
  - (Horizontally) The process dimension indicates the evolution of the parts of the design model as each design task isexecuted
  - (Vertically) The abstraction dimension represents the level of detail as each element of the analysis model is transformed into the design model and then iterativelyrefined
- Elements of the design model use many of the same UML diagrams used in the analysis model
  - The diagrams are refined and elaborated as part of thedesign
  - More implementation-specific detail isprovided

Fig: Dimensions of the DesignModel



- In Design model the emphasis is placedon
  - Architectural structure andstyle
  - Interfaces between components and the outsideworld
  - Components that reside within thearchitecture
- The design model has the following layeredelements
    - Data/classdesign
    - Architecturaldesign
    - Interface design
    - Component-leveldesign
- A fifth element that follows all of the others is <u>deployment-leveldesign</u>



3.3.1DESIGN ELEMENTS

- Data design creates a model of data and objects that is represented at a high level of abstraction
- This data model is then refined into progressively more implementation-specific representations
- At the <u>program component level</u>, the <u>design of data structures and the associated algorithms</u> are used for the creating the high-qualityapplications.
- At the <u>application level</u>, the <u>translation of a data model into a database</u>is essential to achieve the business objectives of asystem.
- At the <u>business level</u>, the collection of information stored in different databases and reorganized into a <u>data warehouse enables data mining</u> or knowledgediscovery.

Architectural design elements

- Architectural design depicts the overall layout of thesoftware
- The architectural model is derived from threesources:

  (1) Information about the applicationdomain

(2) Specific requirements model elements such as data flow diagrams and their relationships

(3) The availability of architectural styles andpatterns.

- The architectural design element is usually represented as a set of interconnected subsystems, each have its ownarchitecture

Interface design elements

- Interface design tells how information flows into and out of the system and how it is communicated among the various components of the architecture
- Elements of interfacedesign:
    - userinterface,
    - external interfaces, and
    - internalinterfaces.
- *UI design* (called as usability design) incorporates aesthetic elements (e.g., layout, color, graphics, interaction mechanisms), ergonomic elements (e.g., information layout and placement, metaphors, UI navigation), and technical elements (e.g., UI patterns, reusable components).
- *External interface design* requires information about the entity to which the information is communicated. The design should incorporate error checking and securityfeatures.

- *Internal interface design* is related with component-level design. Design realizations represent all operations and the messaging schemes required to enable communication and collaboration between operations in variousclasses.

Component-level design elements

- Component level design describes the internal detail of each software componentlike data structure definitions, algorithms, and interfacespecifications.
- A UML activity diagram can be used to represent processinglogic.
- Pseudocode or some other diagrammatic form (e.g., flowchart or box diagram) is used to represent the detailed procedural flow between thecomponent.
- Algorithmic structure follows the rules established for structured programming
- E.g.

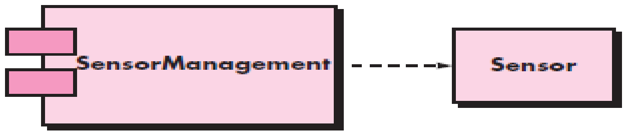
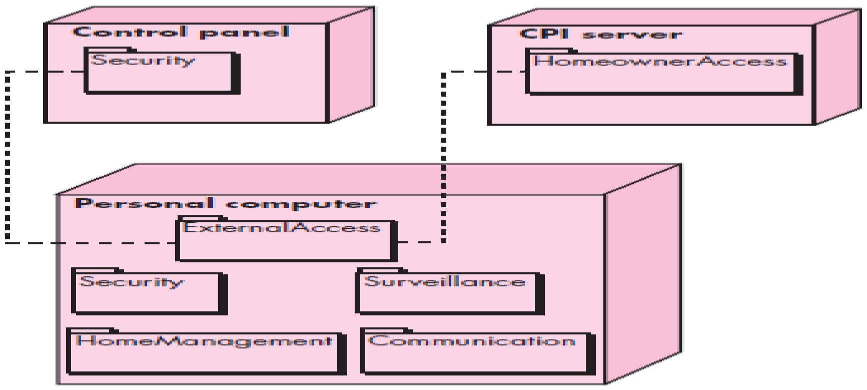


Fig: Component diagram

Deployment-level design elements

➢ This design indicates how software functionality and subsystems will be allocated within the physical computing environment that will support thesoftware

Fig: Deployment Diagram



3.4

DESIGNHUERISTIC

1. **Evaluate the first iteration of the program structure to reduce the couplingand improve cohesion**
   - The module independency can be achieved by exploding or imploding the modules
   - Exploding means obtaining more than one modules in the finalstage
   - Imploding means combining the results of differentmodules
2. **Attempt to minimize the structures with high fan-out and strive for fan-in as depthincreases**
   - At the higher level of program structure the distribution of control should be made.
   - Fan in means number of immediate ancestors the moduleshave
   - Fan –out means number of immediate subordinates to themodule.
3. **Keep scope of the effect of a module within the scope of control of thatmodule**
   - The decisions made in one module should not affect the other module which is

not inside itsscope

4. **Evaluate the module interfaces to reduce complexity and redundancy and improve consistency**
    - The major cause of errors is module interfaces. They should simply pass the information and should be consistent with themodule
5. **Define module whose function is predictable but avoid modules that are too restrictive**
    - The modules should be designed with simple processing logics and should not restrict the size of the local data structure, control flow or modes of external interfaces
6. **Strive for controlled entry modules by avoiding pathologicalconnections**
    - Interfaces should be constrained and controlled. Pathological connection means many references or branches into the middle of themodule

3.5 ARCHITECTURALDESIGN

- The software architectureis the overall structure of the system which is includes the software components , their properties and the relationships among thecomponents
- Software architectural design represents the structure of the data and program components that are required to build a computer-basedsystem
- An architectural design model istransferable
    - It can be applied to the design of othersystems
    - It represents a set of abstractions that enable software engineers to describe architecture in predictableways
- For deriving the architecture, horizontal and vertical partitioning arerequired

Horizontal Partitioning

- Horizontal Partitioning define separate branches of the module hierarchy for each major function
- This use control modules to coordinate communication betweenfunctions
- Eg:

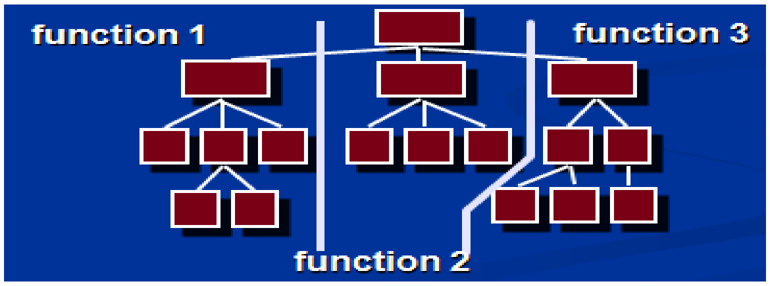
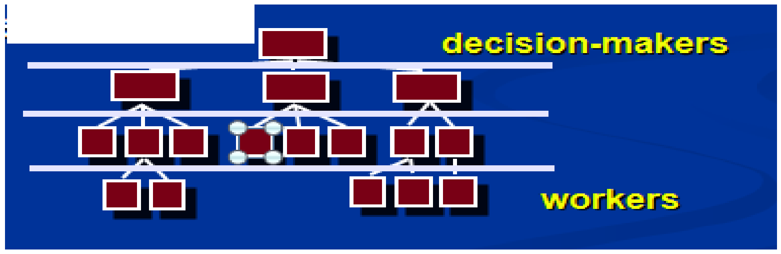


Fig: Horizontal Partitioning

Vertical Partitioning: Factoring

- Vertical partitioning suggests the control and work should bedistributed top-down in the programstructure.
- The decision making modules should reside at the top of thearchitecture

Fig: Vertical Partitioning



3.5.1 ARCHITECTURALSTYLES

- The architectural styles is used to describes a system which has:
  - A set of components that perform a function required by thesystem
  - A set of connectors enabling communication amongcomponents
  - Semantic constraints that define how components can be integrated to form the system
  - A topological layout of the components indicating their runtimeinterrelationships

Taxonomy of Architecturalstyles

Data-Centered Architecture

- A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or modify data within the store.
- The main goalis to integrate thedata
- The shared data may be a passive repository or an activeblackboard
- Passive repository means the client software accesses the data independent of any changes to the data or the actions of other clientsoftware.
- A variation on this approach transforms the repository into a“blackboard”
  - A blackboard notifies subscriber clients when changes occur in data ofinterest
- Clients are relatively independent of each other so they can be added, removed, or changed infunctionality

- The data store is independent of theclients

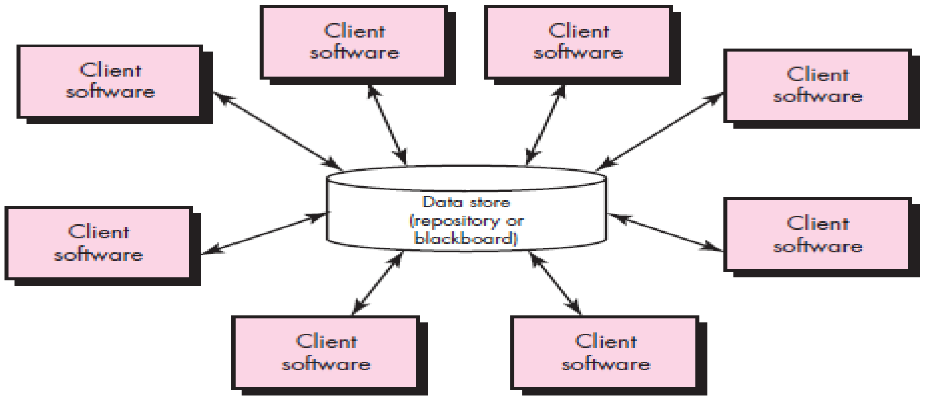


Fig: Data centered architectural style

Advantages:

- Data-centered architectures promote*integrability*
- Data can be passed among clients using the blackboardmechanism.
- Client components can execute the processesindependently.

Data Flow Architectures

- This architecture is used when input has to be transformed into output through a series of computationalcomponents.
- The main goalismodifiability
- **Pipe-and-filterstyle**
    - A pipe-and-filter pattern has a set of components, called *filters*, connectedby *pipes* that transmit data from one component to the next.
    - The filters incrementally transform the data
    - The filter does not require knowledge of the workings of its neighboringfilters.

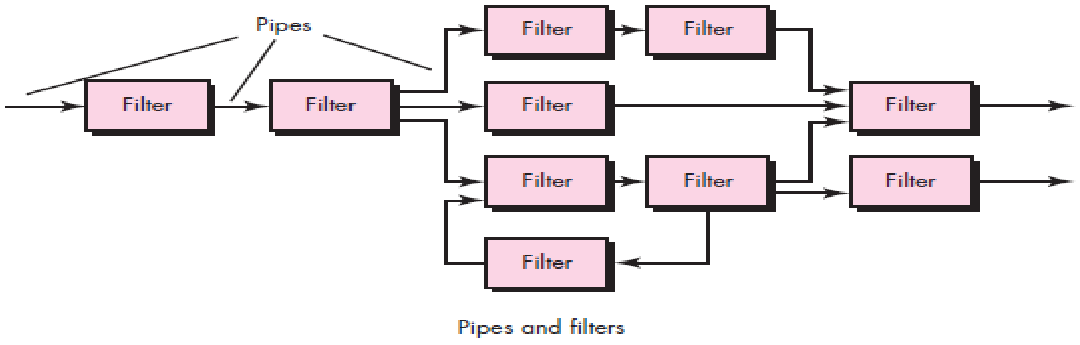
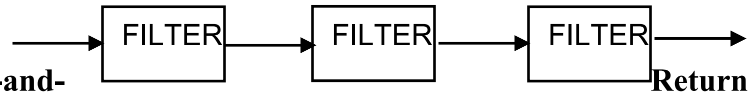


Fig: Pipe and filter style

- **Batch sequentialstyle**
  - If the data flow degenerates into a single line of transforms, it is termed batch sequential.
  - This structure accepts a batch of data and then applies a series of sequential components (filters) to transform it.



Call-and-Return Architectural Style

- This architectural style enables to achieve a program structure that is relatively easy to modify andscale.

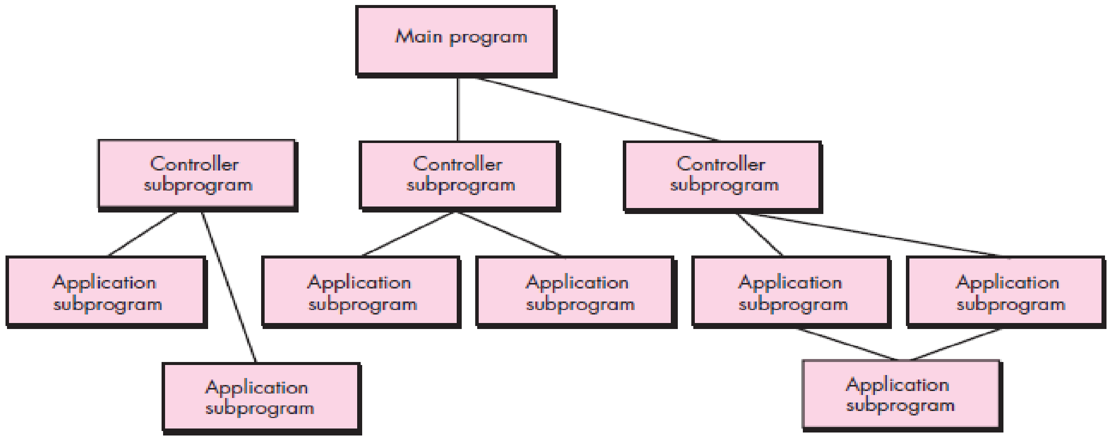


Fig: Call and return architectural style

- Substyles of call and returnarchitecture:
  - *Main program/subprogramarchitectures.*

    This style decomposes the function into a control hierarchy where a main program invokes a number of program components which in turn may invoke other components.
  - *Remote procedure callarchitectures.*

    The components of a main program/subprogram architecture are distributed across multiple computers on anetwork.

Object-oriented architectures.

- The components of a system encapsulate data and the operations that are applied to manipulate thedata.
- Communication between components are accomplished through messagepassing.

Layered architectures.

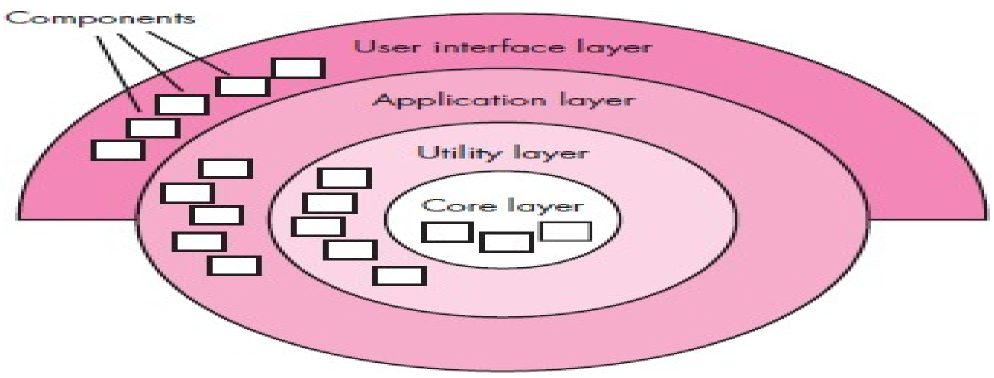


Fig: Layered architecture

- A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instructionset.
- At the outer layer, components service user interfaceoperations.
- At the inner layer, components perform operating systeminterfacing.
- Intermediate layers provide utility services and application softwarefunctions.

Note:

- Once requirements engineering uncovers the characteristics and constraints of the system, then the appropriate architectural style can bechosen.
- For example, a layered style can be combined with a data-centered architecture in many database applications.

3.5.2 ARCHITECTURAL DESIGNPROCESS

Architectural DesignSteps

1) Represent the system incontext
2) Define archetypes
3) Refine the architecture intocomponents
4) Describe instantiations of the system

1. Represent the System inContext

- To represent the system in context an architectural context diagram (ACD) is used that shows
  - The identification and flow of all information into and out of asystem

    - The specification of allinterfaces
    - Any relevant support processing from othersystems
- An ACD models in such a way that the software interacts with entities externalto its boundaries
- An ACD identifies systems that interoperate with the targetsystem
    - Super-ordinate systems are systems that use the target system as part of some higher level processingscheme
    - Sub-ordinate systems are systems that are used by target system and provide necessary data orprocessing
    - Peer-level systems are those systems that interact on a peer-to-peer basis with target system to produce or consumedata
    - Actors are the people or devices that interact with target system to produce or consume data

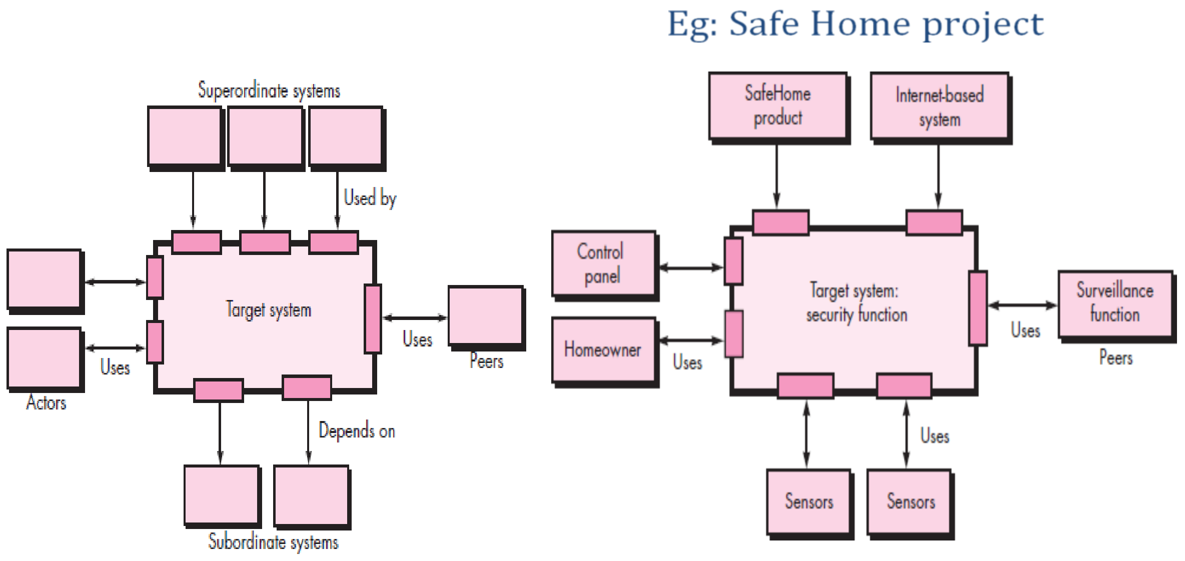


Fig: Architectural context diagram

2. Define Archetypes

- Archetypes indicate the abstractions within the problemdomain.
- An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the targetsystem
- It includes a common structure and class-specific design strategies and a collection of example program designs andimplementations

- Only a relatively small set of archetypes is required in order to design even relatively complexsystems
- The target system architecture is composed of thesearchetypes
    - They represent stable elements of thearchitecture
    - They may be instantiated in different ways based on the behavior of thesystem
    - They can be derived from the analysis classmodel
- The archetypes and their relationships can be **illustrated in the figbelow**

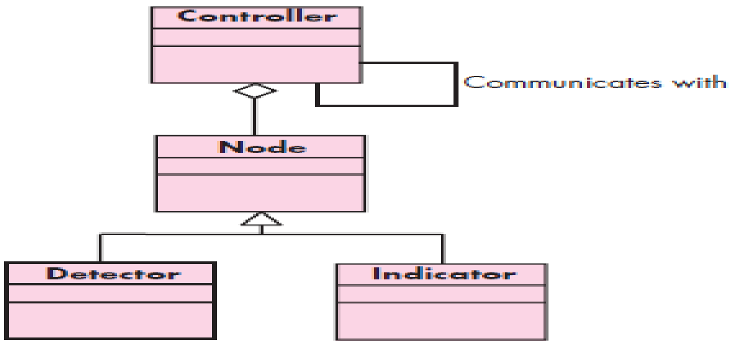
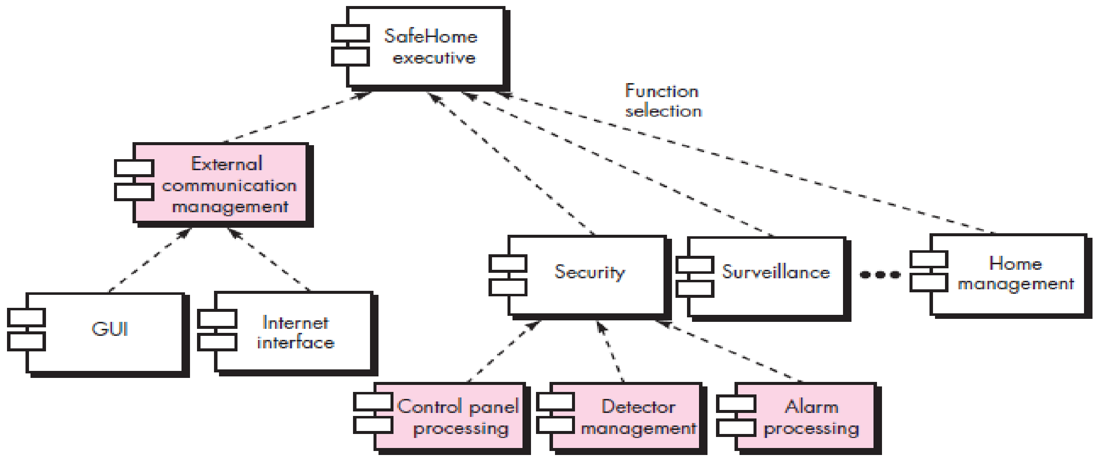


Fig: UML Class diagram

3. Refine the Architecture intoComponents

- The architectural designer refines the software architecture into components to describe the overall structure and architectural style of thesystem
- These components are derived from varioussources
    - The application domain provides components, which are the domain classesin the analysis model that represent entities in the realworld
    - The infrastructure domainprovides design components that enable application components but have no businessconnection
        - Examples: memory management, communication, database, and task management
    - The interfacesin the ACD imply one or more specialized components that process the data that flow across theinterface



➢ A UML class diagram can represent the classes of the refined architecture and their relationships

4. Describe Instantiations of the System

- An actual instantiation of the architecture is developed by applyingit to a specific problem
- This demonstrates that the architectural structure, style and components areappropriate
- A UML component diagram can be used to represent the instantiation of thesystem.

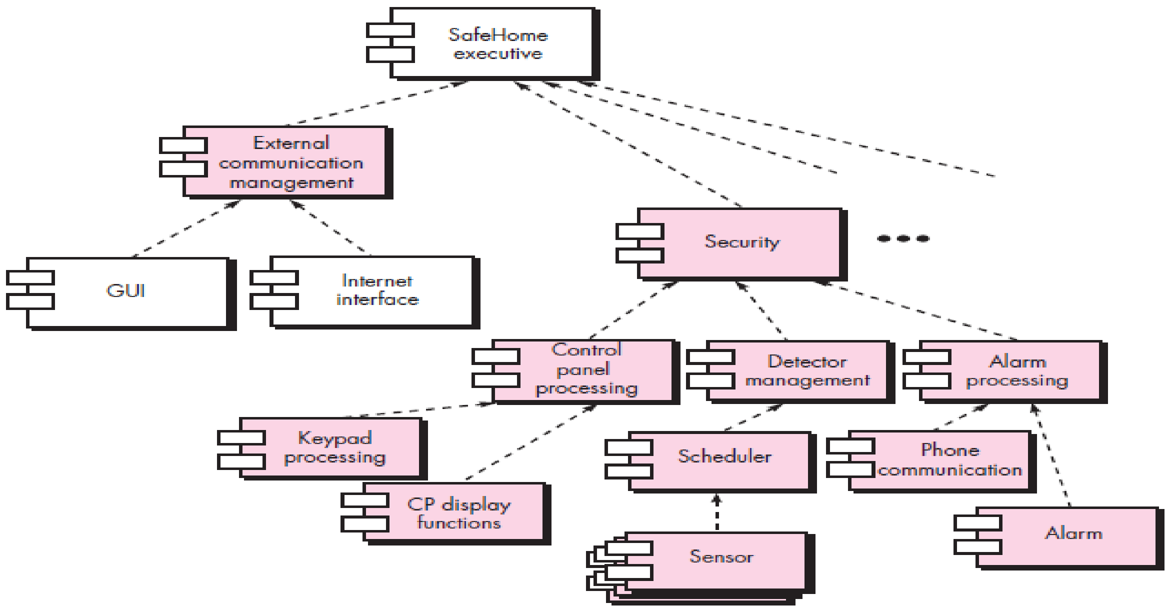


Fig: Refined Component diagram

3.6 ARCHITECTURALMAPPING

Structured design provides a convenient transition from a data flow diagram to software Architecture

Types of information flow

The 2 different types of information flows:

1. **transaction flow -** a single data item triggers information flow along one of manypaths
2. **transformflow**
   - overall data flow is sequential and flows along a small number of straight line paths
   - ***Incoming Flow:*** The paths that transform the external data into an internalform
   - ***Transform Center:*** The incoming data are passed through a transform center and begin to move along paths that lead it out of thesoftware
   - ***Outgoing Flow:*** The paths that move the data out of thesoftware

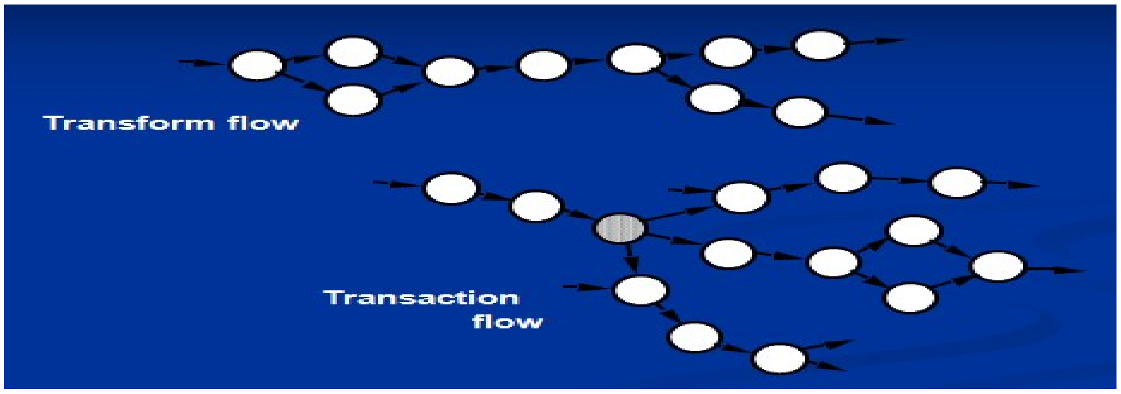


Fig: Types of Information flow

3.6.1 Transformmapping

- Mapping the DFD with transform flow into an Architectural design is referred as Transformmapping

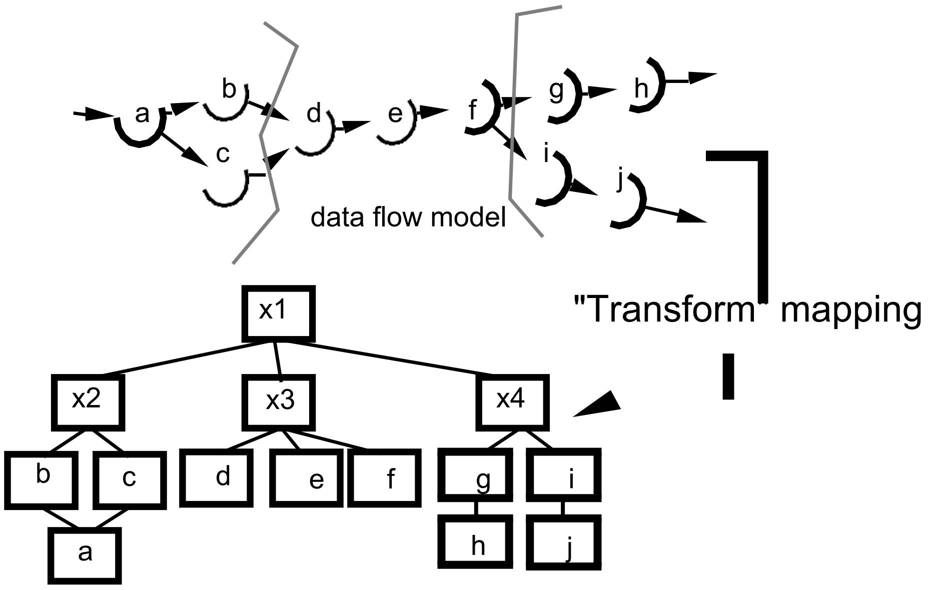


Fig: Transform mapping

STEPS INVOLVED IN ARCHTECTURAL MAPPING:

- **Step 1. Review the fundamental systemmodel.**
  - Eg: SAFE HOME PROJECT

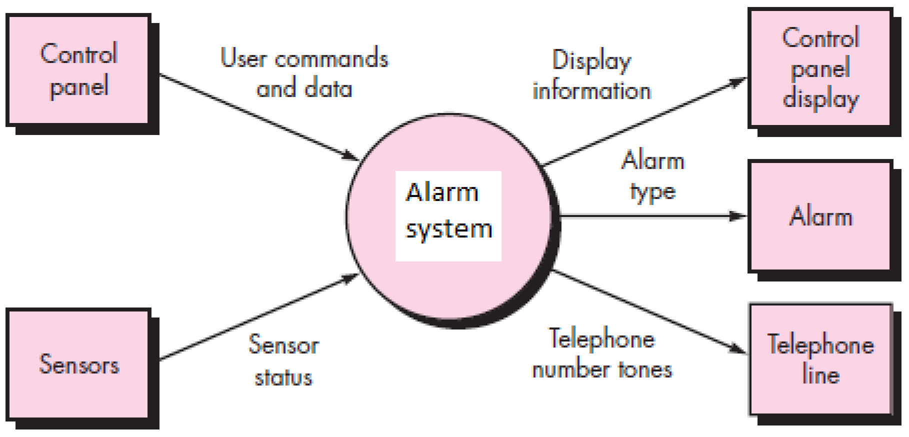


Fig: Context level diagram for Alarm system

➢ **Step 2. Review and refine data flow diagrams for thesoftware.**

Eg:

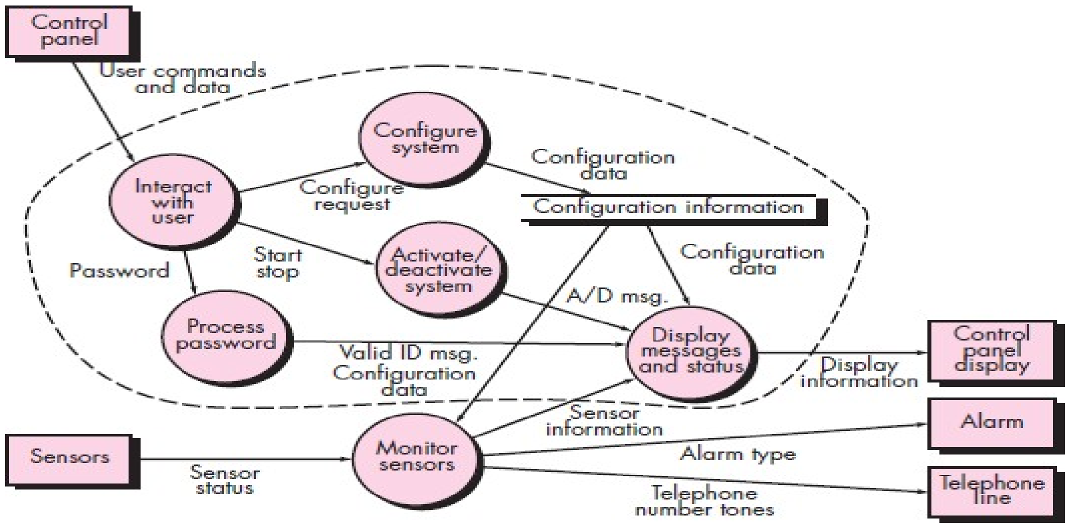


Fig: Level-n DFD

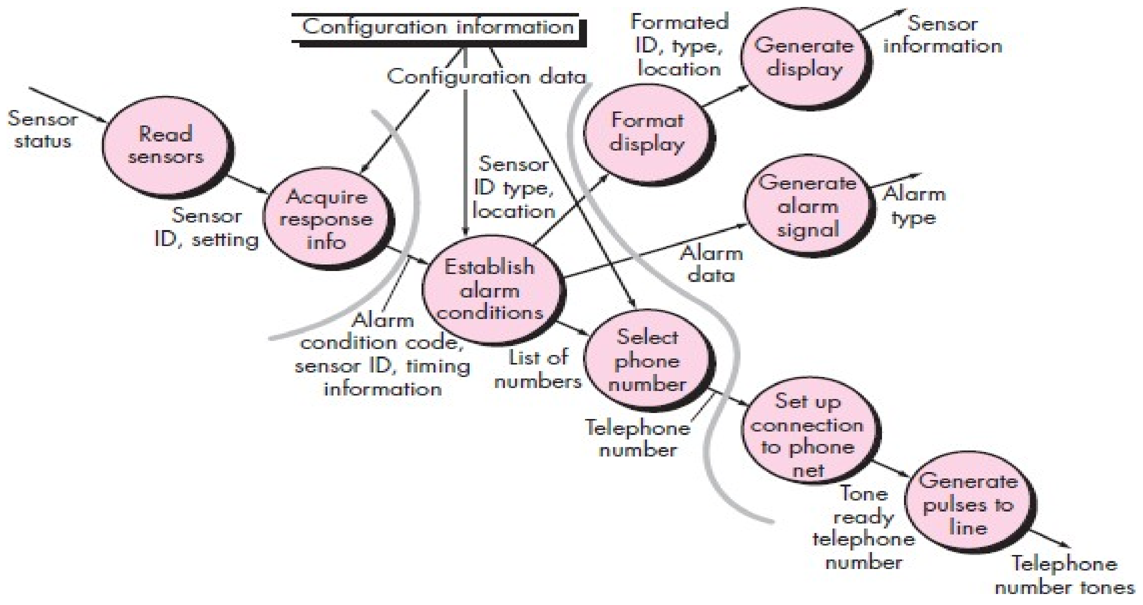
➢ **Step 3. Determine whether DFD has transform or transaction flowcharacteristics.**

- **in general---transformflow**
- **special case---transaction flow**

➢ **Step 4. Isolate the transform center by specifying incoming and outgoing flow boundaries**

- different designers may select slightlydifferently
- transform center can contain more than onebubble.

Eg:



- **Step 5. Perform “first-levelfactoring”**
    - Program structure represents a top-down distributioncontrol.
    - factoring results in a program structure(top-level, middle-level,low-level)
    - Number of modules limited tominimum.

Eg:

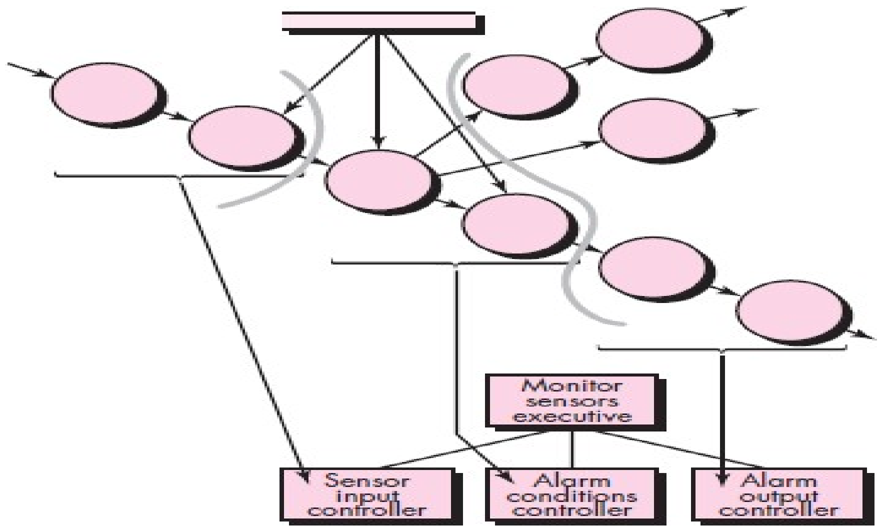
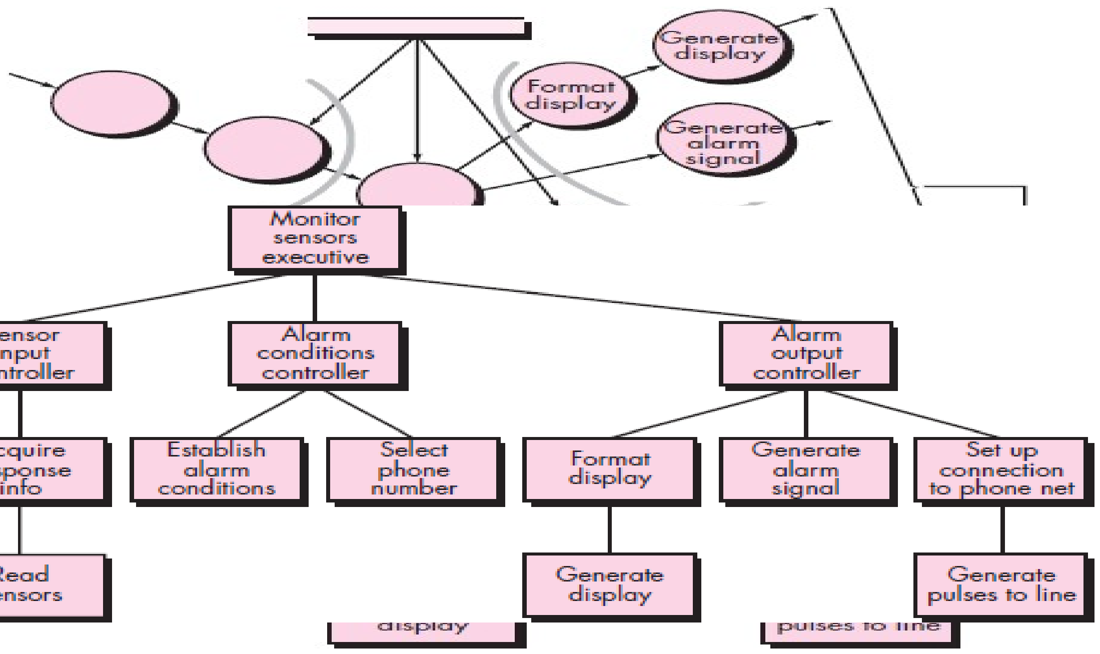


Fig: First level factoring

- **Step 6. Perform “second-levelfactoring”**
  - Mapping individual transforms (bubbles) to appropriatemodules.
  - Factoring accomplished by moving outwards from transform centerboundary.

Fig: Second level factoring

Eg: First iteration architecture



- **Step 7. Refine the first iteration program structure using design heuristics** for improved softwarequality.

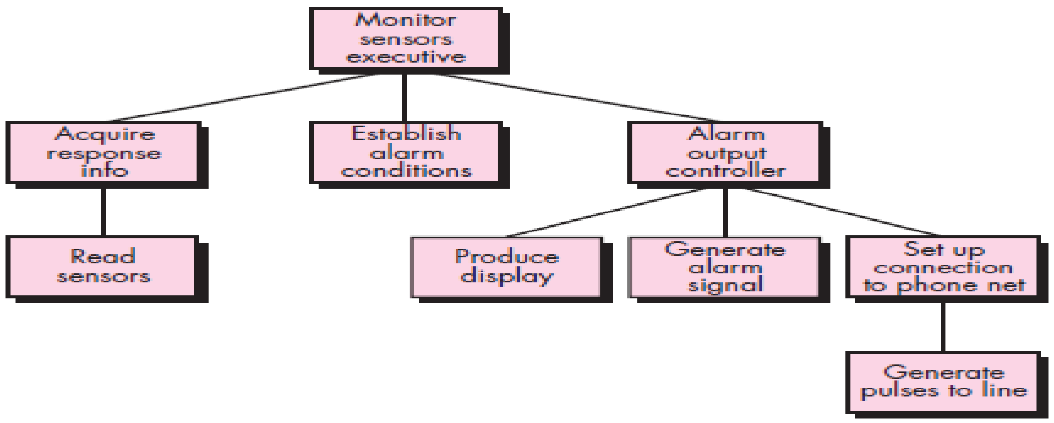


Fig: final Architecture

3.7 USER INTERFACE ANALYSIS ANDDESIGN

3.7.1 Introduction

- **GoldenRules**

1. Place the ***user*** incontrol
    - Define interaction modes in such a waythat does not force a user into ***unnecessary or undesired actions***.
    - Allow user interaction to be ***interruptible andundoable***.
    - Streamline interaction as *skill levels* advance and allow the interaction tobe ***customized***.
    - Design for ***direct*** interaction with objects that appear on thescreen.
2. Reduce the ***user***'s memoryload
    - Reduce demand on ***short-termmemory***.
    - Establish meaningful***defaults***.
    - Define ***shortcuts*** that areintuitive.
    - The visual layout of the interface should be based on a ***real worldmetaphor.***
    - Disclose information in a ***progressive***fashion
3. Make the interface consistent for the***user***
    - Allow the user to put the current ***task*** into a meaningful***context***.
    - Maintain consistency ***across*** a family ofapplications.
    - If ***past*** interactive models have created user expectations, do not make changes unless there is a compelling reason to doso.

3.7.2 Interface Designmodels

- There are Four different models for interfacedesign:
    - **User profilemodel**
        - Establish the profile of end users of thesystem
        - considers the syntactic and semantic knowledge of theuser
    - **Designmodel**
        - This model is derived from the analysis model of therequirements
        - This incorporates data, architectural, interface, and procedural representations of thesoftware
    - **Implementationmodel**

    - This model consists of the look and feel of the interface combined with all supporting information that describe system syntax andsemantics
  - **User's mentalmodel**
    - This model is also called the user's systemperception
    - This model consists of the image of the system that users carry in their heads

3.7.3 INTERFACE DESIGNPROCESS

- User interface development follows a **<u>spiralprocess</u>**

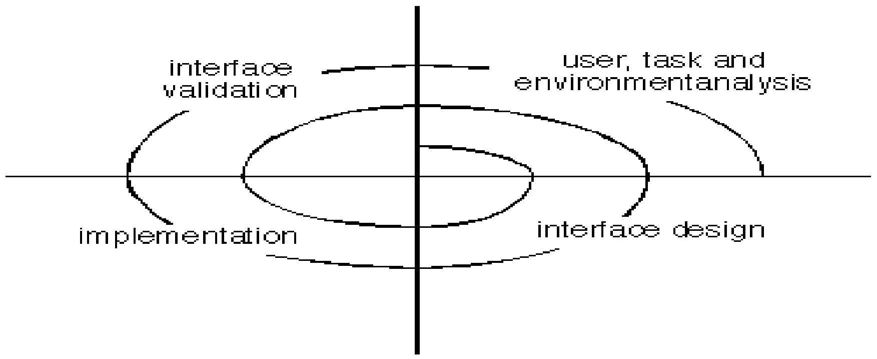


Fig: Interface Design steps

- Steps involved in the interface designprocess:
  1. Interface analysis
     - This phase focuses on the profile of the users who will interact with the system
     - Skill level, business understanding, and general ideas to the new system are recorded; and different user categories aredefined

     - Concentrates on users, tasks, content and workenvironment
  2. Interface design
     - The set of interface <u>objects</u> and <u>actions</u> that is needed to perform all defined tasks in a manner that meets every usability goal defined for the system are defined in this phase.
  3. Interface construction
     - construction begins with a prototype that enables usage scenarios to be evaluated
     - Continues with development tools to complete theconstruction
  4. Interface validation

- This focus on the ability of the interface to implement every user task, to accommodate all task variations, and to achieve all userrequirements
- The degree to which the interface is easy to use and easy tolearn
-

3.7.4 INTERFACEANALYSIS

- Interface analysis meansunderstanding

  (1) The *users* who will interact with the system through theinterface;

  (2) The *tasks* that end-users must perform to do theirwork,

  (3) The *content* that is presented as part of theinterface

  (4) The *environment* in which these tasks will beconducted.

3.7.4.1 UserAnalysis

- The analyst uses both the user's mental model and the design model to understand the and how they use thesystem
- Information are collectedfrom
  - User interviews with the endusers
  - Sales input from the sales people who interact with customers and users on a regularbasis
  - Marketing input based on a market analysis to understand how different population segments might use thesoftware
  - Support input from the support staff who are aware of what works and what doesn't, what users like and dislike, what features are easy to useetc.
- sample questionnaire used for useranalysis

  - Are users trained professionals, technician, clerical, or manufacturingworkers?
  - Are the users capable of learning from written materials or have they expressed a desire for classroomtraining?
  - Are users expert typists or keyboardphobic?
  - Are users experts in the subject matter that is addressed by thesystem?

3.7.4.2 Task Analysis andModeling

- Task analysis helps to know andunderstand
  - The work the user performs in specificcases
  - The tasks and subtasks that are performed by theuser

- The specific problem domain objects that the user manipulates as work is performed
- The sequence of worktasks
- The hierarchy oftasks

Use cases

- Use cases are used to define the interaction between thecomponents.
- Use case show how an end user performs some specific work-related task
- This helps to extract tasks, objects, and overall workflow of theinteraction
- This helps the software engineer to identify additional helpfulfeatures

Task elaboration

- Task elaboration refines interactivetasks
- This gives the step wise elaboration of thefunction
- Eg: withdraw from anATM
  - Insert thecard
  - enterpin
  - select withdrawoption
  - enteramount
  - dispense cashetc

Object elaboration

- Object elaboration identifies interfaceobjects
- Attributes of each class are defined, and an evaluation of the actions applied to each object provides a list ofoperations.

- Eg: class Withdraw can include the attributes like account no, name, balance and functions like withdraw() and update()etc

Workflow analysis

- Workflow analysis defines how a work process is completed when several people are involved
- **For work flow analysis swimlane diagram isused**
- Eg:

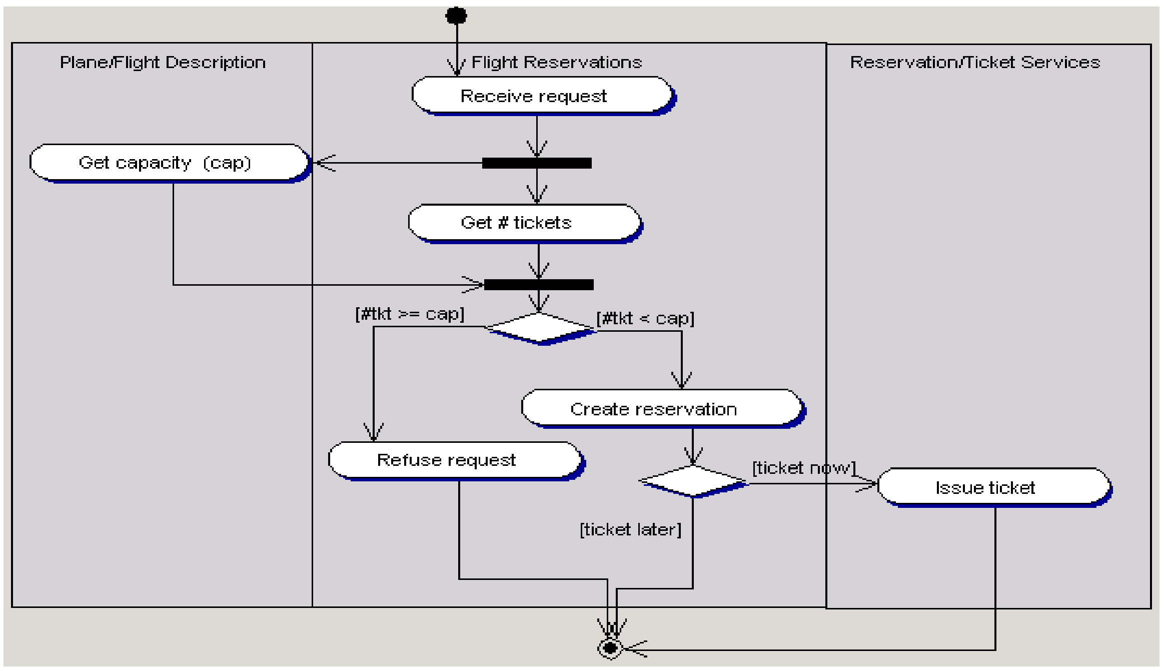


Fig: Swimlane diagram for ticket reservation

Hierarchical representation

- Once workflow has been established, a task hierarchy can be defined for each usertype.
- The hierarchy is derived by a stepwise elaboration of each task identified for theuser

3.7.4.3 Analysis of Display Content

- This phase is about analyzing how to present the variety ofcontents
- The display content may range from character-based reports (e.g Spreadsheet), to graphical displays(e.g : histogram,3D model etc), to multimedia information(e.g Audio or video)
- Display content maybe
    - Generated by components in other parts of theapplication
    - Acquired from data stored in a database that is accessible from theapplication
    - Transmitted from systems external to the application inquestion
- Sample questionnaire for contentanalysis
    - Are different types of data assigned to consistent geographic locations on the screen?
    - Can the user customize the screen location forcontent?
    - Will graphical output be scaled to fit within the bounds of the display device that is used?

   - How will color to be used to enhanceunderstanding?
   - How will error messages and warning be presented to theuser?

3.7.4.4 Work Environment Analysis

- Software products need to be designed to fit into the work environment, otherwise they may be difficult or frustrating touse
- Factors to considerinclude
   - Type oflighting
   - Display size andheight
   - Keyboard size, height and ease ofuse
   - Mouse type and ease ofuse
   - Surroundingnoise
   - Space limitations for computer and/oruser
   - Weather or other atmosphericconditions
   - Temperature or pressurerestrictions
   - Time restrictions (when, how fast, and for howlong)

3.7.5 INTERFACEDESIGN

- User interface design is an iterative process, where each iteration elaborate and refines the information developed in the precedingstep
- Steps involved in user interfacedesign
   1. Define user interface <u>objects</u> and <u>actions</u> from the information collected in the analysisphase
   2. Define events that will cause the state of the user interface to change and model the behavior
   3. Depict each interface state as it will actually look to the enduser
   4. Indicate how the user interprets the state of the system from information provided through theinterface

Applying Interface Design steps

- Interface objects and actions are obtained from a <u>grammatical parse</u> of the use cases and the software problemstatement
- Interface objects are categorized <u>into 3 types</u>: source, target, andapplication

    - A **source object** is dragged and dropped into a **target**object such as to create a hardcopy of areport
    - An **application object** represents application-specific data that are not directly manipulated as part of screen interaction such as alist
- After identifying objects and their actions, an interface designer performs screen layoutwhichinvolves
    - Graphical design and placement oficons
    - Definition of descriptive screentext
    - Specification and titling forwindows
    - Definition of major and minor menuitems
    - Specification of a real-world metaphor tofollow

User Interface Design Patterns

- A design pattern is an abstraction that prescribes a design solution to a specific, well-bounded designproblem.
- Eg:
    - **CalendarStrip** that produces a continuous, scrollable calendar in which the current date is highlighted and future dates may be selected by picking them from the calendar.

Design Issues

- Four common design issues in any userinterface
    - **System response time** (both length andvariability)
        - **Length** is the amount of time taken by the system torespond.
        - **Variability** is the deviation from average responsetime
    - **User helpfacilities**
        - Help facilities gives information about when is it available, how is it accessed, how is it represented to the user, how is it structured, what happens when help isexited
        - Eg: user manuals or online helpfacilities
    - **Error informationhandling**
        - Error messages and warnings should describe theproblem
        - It should provide constructive advice for recovering from theerror
        - It should indicate negative consequences of the erroretc
        - This messages helps to improve the quality of an interactive system and will

significantly reduce user frustration when problems dooccur

- **Menu and commandlabeling**
  - Menu and command labeling should be consistent, easy tolearn
  - Questions for menulabeling
    1. Will every menu option have a correspondingcommand?
    2. What can be done if a command isforgotten?
    3. Can commands be customized or abbreviated by theuser?
- **Applicationaccessibility**
  - Software engineers must ensure that interface design encompasses mechanisms that enable easy access for those with specialneeds.
- **Internationalization**
  - The challenge for interface designers is to create "globalized"software.
  - That is, user interfaces should be designed to accommodate a generic core of functionality that can be delivered to all who use thesoftware.
  - Localization features enable the interface to be customized for a specific market.

3.8 COMPONENT LEVELDESIGN

- A component is a modular building block for computersoftware
- A component-level design can be represented using some intermediate representation (e.g. graphical, tabular, or text-based) that can be translated into sourcecode

3.8.1 DESIGNING CLASS-BASEDCOMPONENTS

Component-level DesignPrinciples

- **Open-closedprinciple**
  - A module or component should be open for extension but closed formodification
  - The designer should specify the component in a way that allows it to be extended without the need to make internal code or design modifications to the existing parts of the component
- **Liskov substitutionprinciple**
  - The Subclasses should be substitutable for their baseclasses
  - A component that uses a base class should continue to function properlyif a subclass of the base class is passed to the componentinstead
- **Dependency inversionprinciple**

- This principle depend on abstractions ,do not depend onconcretions
- The more a component depends on other concrete components, the more difficult it will be toextend

- **Interface segregationprinciple**
  - Many client-specific interfaces are better than one general purposeinterface
  - For a server class, specialized interfaces should be created to serve major categories ofclients
  - Only those operations that are relevantto a particular category of clients should be specifiedin theinterface

Component Packaging Principles

- **Release reuse equivalencyprinciple**
  - The granularity of reuse is the granularity ofrelease
  - Group the reusable classes into packages that can be managed, upgraded, and controlled as newer versions arecreated
- **Common closureprinciple**
  - Classes that change together belongtogether
  - Classes should be packaged cohesively; they should address the same functional or behavioral area on the assumption that if one class experiences a change then they all will experience achange
- **Common reuseprinciple**
  - Classes that aren't reused together should not be groupedtogether
  - Classes that are grouped together may go through unnecessary integration and testing when they have experienced no changes but when other classes in the package have beenupgraded

3.8.2 Component-Level DesignGuidelines

- Components
  - Naming conventions should be established for components that are specified as part of the architectural model and then refined and elaborated as part of the component-levelmodel
  - architectural component names must be obtained from the problem domain and ensure that they have meaning to all stakeholders who view the architectural model

(e.g.,Calculator)

– infrastructure component names must that reflect their implementation-specific meaning (e.g.,Stack)

- Dependencies and inheritance inUML
  – Dependencies should be modelled from left to right and inheritance from top(base class) to bottom (derivedclasses)
- Interfaces
  – Interfaces provide important information about communication andcollaboration
  – lollipop representation of an interface should be used in UMLapproach
  – For consistency, interfaces should flow from the left-hand side of the component box;
  – only those interfaces that are relevant to the component under consideration should beshown

3.8.3 Cohesion

- Cohesion is the "single-mindedness' of acomponent
- It implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself
- The objective is to keep cohesion as high aspossible
- The kinds of cohesion can be ranked in order from highest (best) to lowest(worst)
  - **Functional**
    - Exhibited primarily byoperations
    - This level of cohesion occurs when a component performs a targeted computation and then returns aresult.
  - **Layer**
    - Exhibited by packages, components, andclasses,

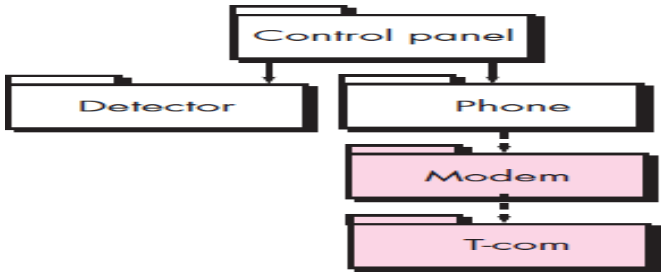


Fig: Layer Cohesion

        - This type of cohesion occurs when a higher layer accesses the services of a lower layer, but lower layers do not access higherlayers.
    - **Communicational**
        - All operations that access the same data are defined within oneclass.

3.8.4 Coupling

- Coupling is a qualitative measure of the degree to which operations and classes are connected to one another
- The objective is to keep coupling as low aspossible
- Different types of coupling are asfollows:
    - **Contentcoupling**
        - Occurs when one component surreptitiously modifies data that is internal to another component.
        - This violates informationhiding.
    - **Datacoupling**
        - Data coupling occurs when operations pass long strings of dataarguments.
        - Operation A() passes one or more atomic data operands to operation B(); the less the number of operands, the lower the level ofcoupling
        - Testing and maintenance are moredifficult
    - **Commoncoupling**
        - This coupling occurs when a number of components make use of a global variable, which can lead to uncontrolled error propagation and unforeseen side effects when changes aremade.
    - **Stampcoupling**
        - A whole data structure or class instantiation is passed as a parameter to an operation
    - **Controlcoupling**
        - Operation A() invokes operation B() and passes a control flag to B that directs logical flow withinB()
        - Consequently, a change in B() can require a change to be made to the meaning of the control flag passed by A(), otherwise an error mayresult
    - **Routine callcoupling**
        - Occurs when one operation invokesanother.

- **Type usecoupling**
  - Occurs when a Component A uses a data type defined in componentB
  - If the type definition changes, every component that declares a variable of that data type must alsochange
- **Inclusion or importcoupling**
  - Occurs when component **A** imports or includes a package or the content of component**B**.
- **Externalcoupling**
  - Occurs when a component communicates or collaborates with infrastructure components that are entities external to the software (e.g., operating system functions, database functions, networkingfunctions)

3.9 DESIGNING CONVENTIONALCOMPONENTS

- Conventional design constructs emphasize the maintainability of a functionaldomain
- The constructs include Sequence, condition, andrepetition
- Each construct has a predictable logical structure where control enters at the top and exits at the bottom, enabling a maintainer to easily follow the proceduralflow
- Various notations used for designing theseconstructs
  1. Graphical designnotation
     - Sequence, if-then-else, selection,repetition
  2. Tabular designnotation
  3. Program designlanguage

     - This is similar to a programming language but it uses narrative text embedded within the programstatements

Graphical design notation

- Graphical tools, such as the UML activity diagram or the flowchart, provide useful pictorial patterns that shows the proceduraldetail.
- The **activity diagram** allows to represent sequence, condition, andrepetition
- A **flowchart** like an activity diagram which is quite simplepictorially.
- Notations used inflowchart:
  - A boxis used to indicate a processingstep.
  - A diamond represents a logicalcondition

    - arrows show the flow of control.

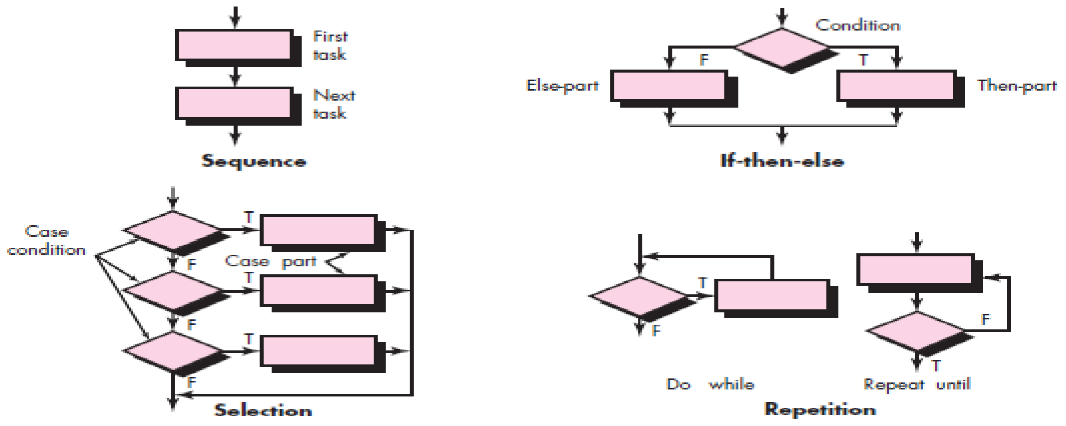
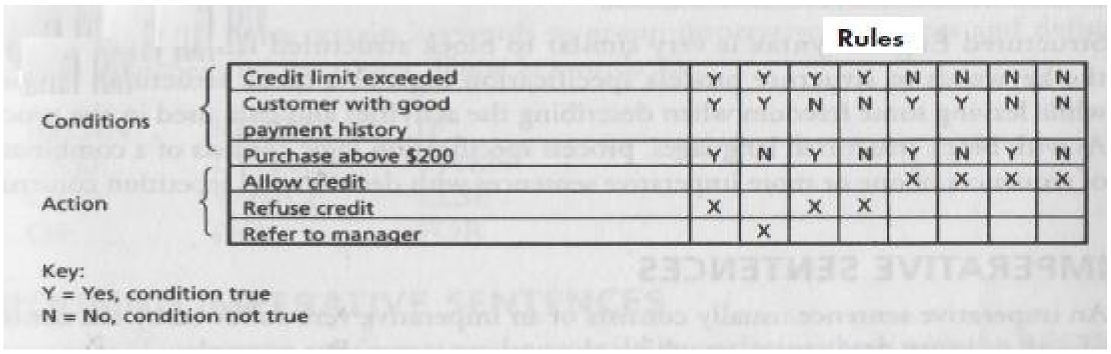


Fig: Flowchart constructs

Tabular Design Notation

- Decision tables provide a notation that translates actions and conditions into a tabular form.
- Tabular notation is difficult tomisinterpret.
- The table is divided into foursections.
    - The upper left-hand quadrant contains a **list of allconditions**.
    - The lower left-hand quadrant contains a **list of all actions** that are possible based on combinations ofconditions.
    - The right-hand quadrants form a matrix that indicates **condition combinations and the corresponding actions** that will occur for a specificcombination.
    - Therefore, each column of the matrix may be interpreted as a **processingrule**.

Eg:



		Rules							
Conditions	Credit limit exceeded	Y	Y	Y	Y	N	N	N	N
	Customer with good payment history	Y	Y	N	N	Y	Y	N	N
	Purchase above $200	Y	N	Y	N	Y	N	Y	N
Action	Allow credit					X	X	X	X
	Refuse credit	X		X	X				
	Refer to manager		X						

Key:
Y = Yes, condition true
N = No, condition not true

- **Steps involved in developing a decisiontable:**

1) List all actions associated with eachmodule
2) List all conditions during execution of theprocedure
3) Associate specific sets of conditions with specific actions, eliminatingimpossible

combinations ofconditions

4) Define <u>rules</u> by indicating what actions occurs for a set ofconditions

Program Design Language

- Program design language (PDL), also called structured English orpseudocode,
- PDL combines the logical structure of a programming language with the free-form naturallanguage.
- Automated tools can be used to enhance theapplication.
- A PDL syntax should include constructs for component definition, interface description, data declaration, block structuring, condition constructs, repetition constructs, and I/O constructs
- PDL can be extended to include keywords for multitasking and concurrent processing, interrupt handling, interprocess synchronizationetc.
- Sample PDL:

```
IF credit level exceeded
THEN (credit level exceeded)
IF customer has bad payment history THEN refuse credit
ELSE ( customer has good payment history) IF purchase is above $200
THEN refuse credit ELSE (purchase is below $200) Refer to
manager
ELSE (credit level not exceeded) allow credit
```