UNIT-II

Software engineers use product metrics to help them assess the quality of the design and construction of the software product being built. Product metrics provide software engineers with a basis to conduct analysis, design, coding, and testing more objectively. Qualitative criteria for assessing software quality are not always sufficient by themselves. The process of using product metrics begins by deriving the software measures and metrics that are appropriate for the software representation under consideration. Then data are collected and metrics are computed. The metrics are computed and compared to pre-established guidelines and historical data. The results of these comparisons are used to guide modifications made to work products arising from analysis, design, coding, or testing.

Software engineers use product metrics to help them assess the quality of the design and construction of the software product being built. Product metrics provide software engineers with a basis to conduct analysis, design, coding, and testing more objectively. Qualitative criteria for assessing software quality are not always sufficient by themselves. The process of using product metrics begins by deriving the software measures and metrics that are appropriate for the software representation under consideration.

➢ System Engineering

Software engineering occurs as a consequence of a process called system engineering. Instead of concentrating solely on software, system engineering focuses on a variety of elements, analyzing, designing, and organizing those elements into a system that can be a product, a service, or a technology for the transformation of information or control.

The system engineering process takes on different forms depending on the application domain in which it is applied. Business process engineering is conducted when the context of the work focuses on a business enterprise. When a product is to be built, the process is called product Engineering.

Both business process engineering and product engineering attempt to bring order to the development of computer-based systems, work to allocate a role for computer software and, at the same time, to establish the links that tie software to other elements of a computer-based system.

➢ Computer-Based Systems

1.Set or arrangement of things so related as to form a unity or organic whole;2.a set of facts,principles,rules,etc.classified and arranged in an orderly form so as to show a logical plan linking the various parts3.a method or plan of classification or arrangement;4.an established way of doing something;method;procedure…….

We define a computer-based system as

A set or arrangement of elements that are organized to accomplish some predefined goal by processing information.

The goal may be to support some business function or to develop a product that can be sold to generate business revenue.To accomplish the goal,a computer-based system makes use of a variety of system elements:

Software.Computer programs, data structures, and related work products that serve to effect the logical method, procedure, or control that is required.

Hardware:Electronic devices that provide computing capability ,the interconnectivity devices(e.g. ., network switches, telecommunications devices)that enable the flow of data, and electromechanical devices(e.g., sensors,motors,pumps)that provide external world function.

People: users and operators of hardware and software.

Database: A Large, organized collection information (e.g., models, specifications, hard-copy manuals, on-line help files, web sites) that portays the use and/or operation of the system.

Procedures: The steps that define the specific use of each system element or the procedural context in which the system resides.

These elements combine in a variety of ways to transform information. For example, a marketing department transforms raw sales data into a profile of the typical purchaser of aproduct;a robot transforms a command file containing specific instructions into a set of control signals that cause some specific physical action Creating an information system to assist the marketing department and controls software to support the robot both require system engineering.

One complicated characteristic of computer-based systems is that the elements constituting one system may also represent one macro element of a still larger system .The macro element is a computer-based system that is one part of a larger computer-based systems. As an example, we consider a factory automation system that is essentially a hierarchy of systems. At the lowest level of the hierarchy we have a numerical control machine, robots, and data entry devices.

➢ THE SYSTEM ENGINEERING HIERARCHY

System Engineering encompasses a collection of top- down and methods to navigate the hierarchy illustrated in figure 3.1. The system engineering process usually begins with a “world view”. That is, the entire business or product domain is examined to ensure that the proper business or technology context can be established. The world view is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements is analysed. Finally, the analysis, design, and construction of a targeted system element are initiated. At the top of the hierarchy, a very broad context is established and, at the bottom, detailed activities, performed by the relevant engineering disciplines are conducted

The world view (WV) is composed of a set of domains (D1), which can each be a system or system of systems in its own right

WV = {D1 D2 D3………, Dn}

Each domain is composed of specific elements (Ej) each of which serves some role in accomplishing the objective and goals of the domain and component

Di = {E1 E2 E3 …… Em}

Finally, each element is implemented by specifying the technical components (Ck) that achieve the necessary function for an element

Ej = {C1, C2 C3 …….. C k}

In the software context, a component could be a computer program, a reusable program component, a module, a class or object, or even a programming language statement.

System Modeling System modeling is an important element of the system engineering process. Whether the focus is on the world view or the detailed view, the engineer creates models that:

• Define the process that serves the needs of the view under consideration.

• Represent the behavior of the processes and the assumption on which the behavior is based.

To construct a system model, the engineer should consider a number of restraining factors:

1. Assumption that reduce the number of possible permutations and variations, thus enabling a model to reflect the problem in a reasonable manner
2. Simplifications that the model has to create in a timely manner
3. Limitations that help to bound the system
4. Constraints that will guide the manner in which the model is created and the approach taken when the model is implemented.

5. Preference that indicate the preferred architecture for all data, functions, and technology.

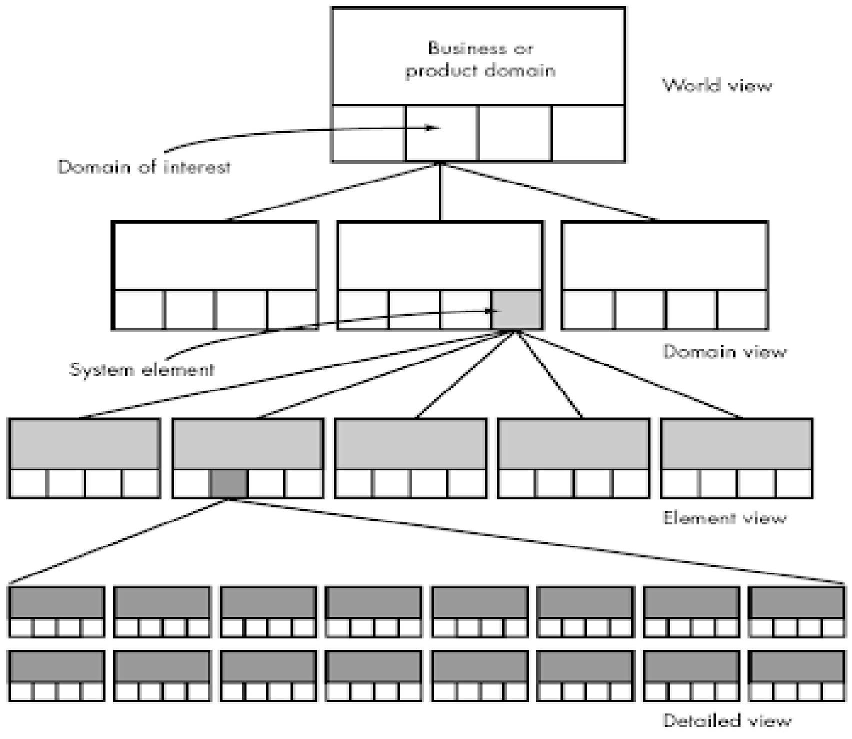
System Simulations

Many computer- based systems interact with the real world in a reactive fashion. That is, real-world events are monitored by the hardware and software that from the computer- based system, and based on these events; the system imposes control on the machine, process, and even people who cause the events to occur. Real- time and embedded systems often fall into the reactive systems category

There are many systems in the reactive category- control machine and/or processes (e.g., commercial aircraft or petroleum refineries) that must operate with an extremely high degree of reliability. If the system fails, significant economic or human loss could occur. For this reason system modeling and simulation tools are used to help eliminate surprises when reactive computer- based systems are built. These tools are applied during the system engineering process, while the role of hardware and software, databases, and people is being specified. Modeling and simulation tools enable a system engineer to "test drive" a specification of the system

System Simulation Tools:

Objective: System simulation tools provide the software engineer with the ability to predict the behavior of a real-time system prior to the time that it is built. In addition, these tools enable the software engineer to develop mock-ups of the real-time system, allowing the customer to gain insight into the function, operation, and response prior to actual implementation



Mechanics: Tools in this category allow a team to define the elements of a computer- based system and then execute a variety of simulations to better understand the operating characteristics and overall performance of the system. Two broad categories of system simulation tools exist: (1) general purpose tools that can model virtually any computer- based system, and (2) special purpose tools that are designed to address a specific application domain (e.g., aircraft avionics systems, manufacturing systems, electronic- systems)

Representative Tools: CSIM, developed by Lockheed Martin Advanced Technology Labs (www.atl.external.lmco.com), is a general purpose discrete- event simulation for block diagram- oriented systems.

➢ BUSINESS PROCESS ENGINEERING

The goal of business process engineering is to define architectures that will enable a business to use information effectively. When taking a world view of a company's information technology needs, there is little doubt that system engineering is required. Not only is the specification of the appropriate computing architecture required. But the software architecture that populates the organization's unique configuration of computing resources must be developed. Business process engineering is one approach for creating an overall plan for implementing the computing architecture. The different architectures must be analyzed and designed within the context of business objectives and goals

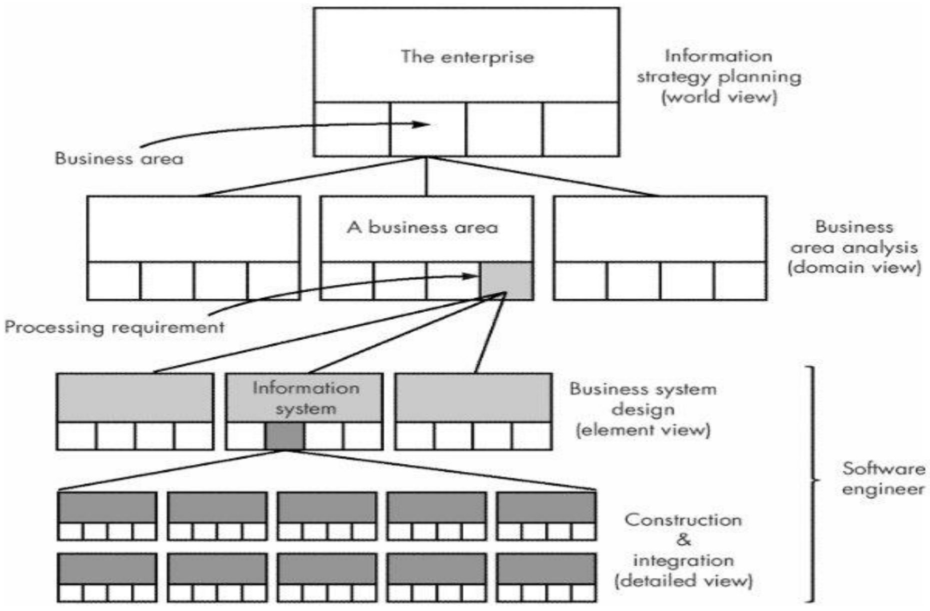
• Data architecture
• Applications architecture
• Technology infrastructure

The data architecture provides a framework for the information needs of a business or business function. The individual building blocks of the architecture are the data objects that are used the business. A data object contains a set of attributes that define some aspect. Quality, characteristic, or descriptor of the data that are being described

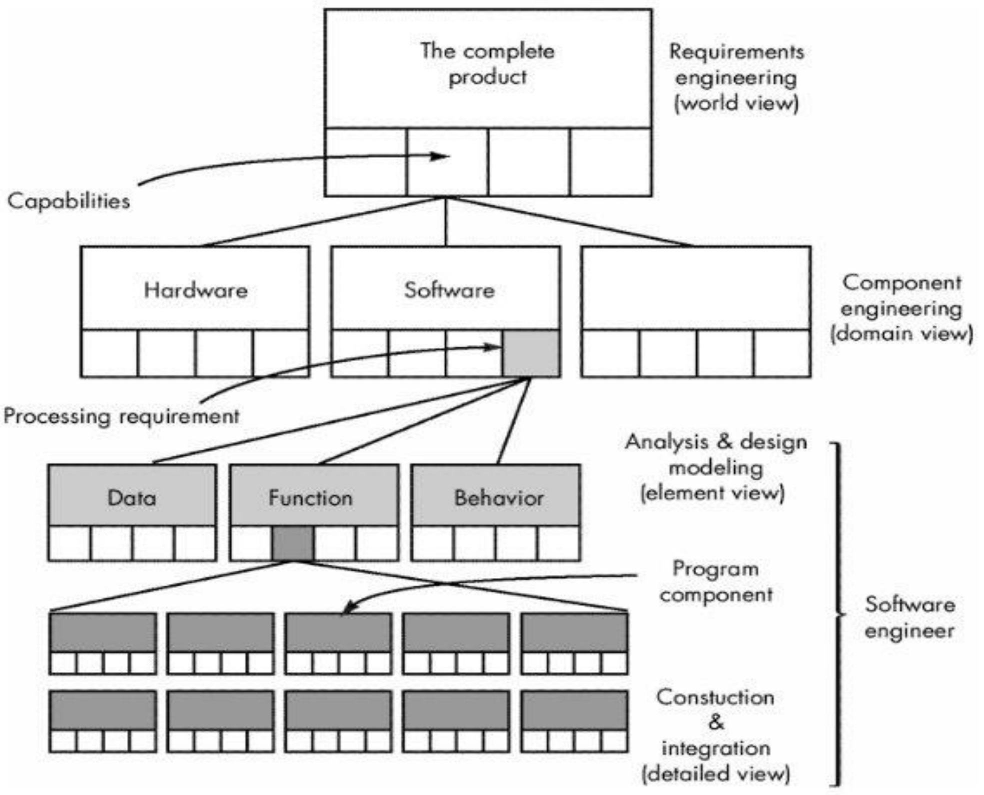
Once set of data objects is defined, their relationships are identified. A relationship indicates how objects are connected to one another. As an example, consider the objects: customer and product A. The two objects can be connected by the relationship purchases; that is, a customer purchases product A or product is purchased by customer. The data objects (there may be hundreds or even thousands for a major business activity) flow between business functions, are organized within a database, and are transformed to provide information that serves the needs of the business

The application architecture encompasses those elements of a system that transform objects within the data architecture for some business purpose. In the context of this book, we consider the application architecture to be the system of programs (software) that performs this transformation. However, in a broader context, the application architecture might incorporate the role of people (who are information transformers and users) and business procedures that have not been automated

The technology infrastructure provides the foundation for the data and application architectures. The infrastructure encompasses the hardware and software that are used to support the applications and data. This includes computers, operating systems, networks, telecommunication links, storage technologies, and the architecture (e.g., client/ server) that has been designed to important these technologies.


The enterprise
Information strategy planning (world view)
Business area
A business area
Business area analysis (domain view)
Processing requirement
Information system
Business system design (element view)
Software engineer
Construction & integration (detailed view)

Product Engineering:


The complete product
Requirements engineering (world view)
Capabilities
Hardware
Software
Component engineering (domain view)
Processing requirement
Analysis & design modeling (element view)
Data
Function
Behavior
Program component
Software engineer
Constuction & integration (detailed view)

The goal of product engineering is to translate the customer's desire for a set of defined capabilities into a working product. To achieve this goal, product engineering—like business process engineering—must derive architecture and infrastructure. The architecture encompasses four distinct system components: software, hardware, data (and databases), and people. A support infrastructure is established and includes the technology required to tie the components together and the information (e.g., documents, CD-ROM, video) that is used to support the components.

The world view is achieved through requirements engineering. The overall requirements of the product are elicited from the customer. These requirements encompass information and control needs, product function and behavior, overall product performance, design and interfacing constraints, and other special needs. Once these requirements are known, the job of requirements engineering is to allocate function and behavior to each of the four components noted earlier.

Once allocation has occurred, system component engineering commences. System component engineering is actually a set of concurrent activities that address each of the system components separately: software engineering, hardware engineering, human engineering, and database engineering. Each of these engineering disciplines takes a domain-specific view, but it is important to note that the engineering disciplines must establish and maintain active communication with one another. Part of the role of requirements engineering is to establish the interfacing mechanisms that will enable this to happen.

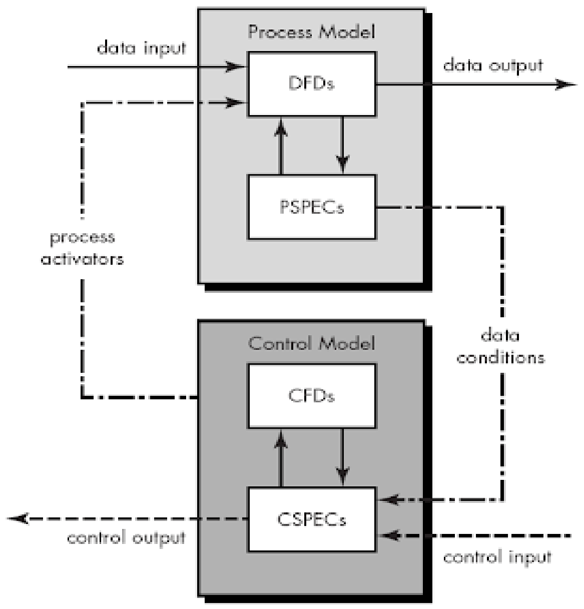
The element view for product engineering is the engineering discipline itself applied to the allocated component. For software engineering, this means analysis and design modeling activities and construction and integration activities that encompass code generation, testing, and support steps. The analysis step models allocated requirements into representations of data, function, and behavior. Design maps the analysis model into data, architectural, interface, and software component-level designs.

➢ System Modeling

System models are hierarchical or layered as a system is represented at different levels of abstraction. The top level of hierarchy presents the complete system. The data objects, functions, behaviors are represented. As the hierarchy is refined or layered, component level detail is modeled and finally system models evolve into engineering models. Hatley-Pirbhay modeling is an extension of the concept that every computer system can be modeled through the usage of an input-processing-output model by including the two additional features of user interface process and maintenance/self testing. These five components are added to a system model template to allow for modeling of the system which allows for proper assignment to the processing regions.

The Hatley-Pirbhai Modeling

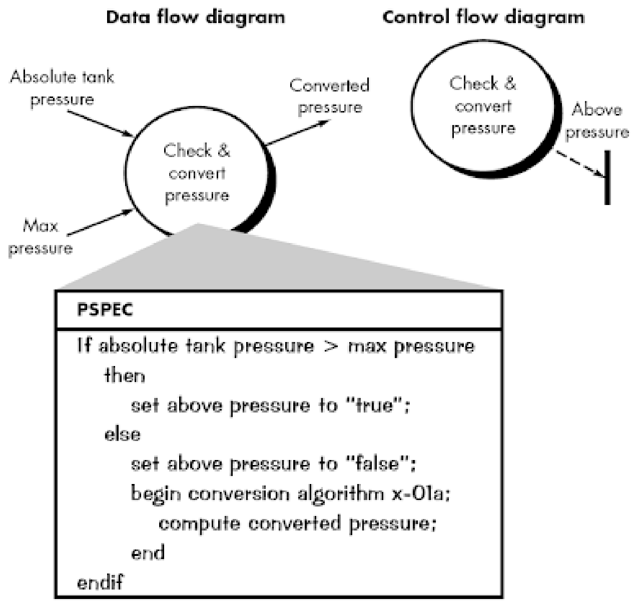
The Hatley and Pirbhai extensions to basic structured analysis notation focus less on the creation of additional graphical symbols and more on the representation and specification of the control-oriented aspects of the software. The dashed arrow is once again used to represent control or event flow. Unlike Ward and Mellor, Hatley and Pirbhai suggest that dashed and solid notation be represented separately. Therefore, a control flow diagram is defined. The CFD contains the same processes as the DFD, but shows control flow, rather than data flow. Instead of representing control processes directly within the flow model, a notational reference (a solid bar) to a control specification (CSPEC) is used. In essence, the solid bar can be viewed as a "window" into an "executive" (the CSPEC) that controls the processes (functions) represented in the DFD based on the event that is passed through the window. The CSPEC is used to indicate (1) how the software behaves when an event or control signal is sensed and (2) which processes are invoked as a consequence of the occurrence of the event. A process specification is used to describe the inner workings of a process represented in a flow diagram.



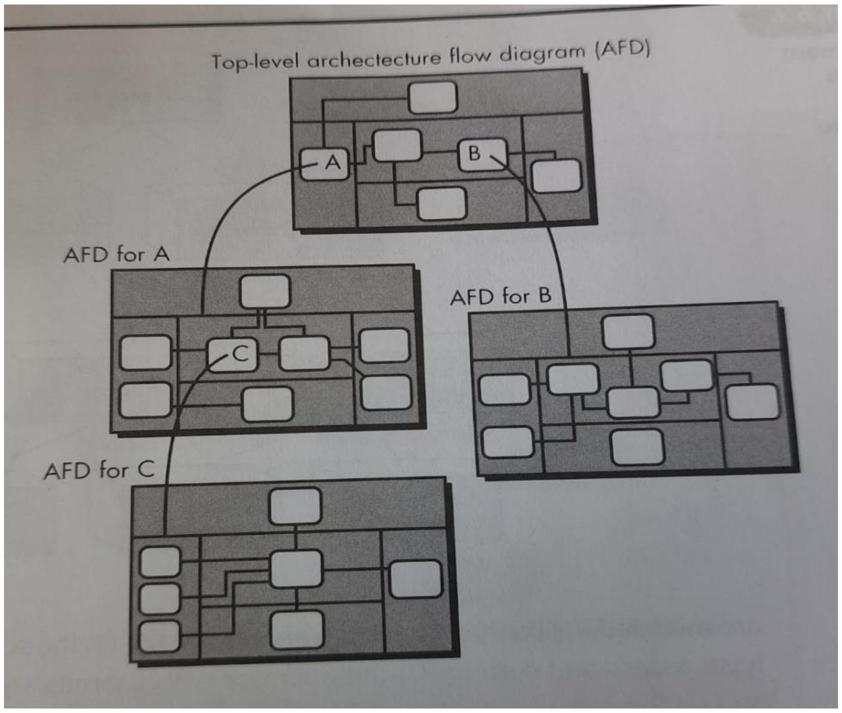
Using the notation described in earlier figures, along with additional information contained in PSPECs and CSPECs, Hatley and Pirbhai create a model of a real-time system. Data flow diagrams are used to represent data and the processes that manipulate it. Control flow diagrams show how events flow among processes and illustrate those external events that cause various processes to be activated. The interrelationship between the process and control models is shown schematically in the figure below. The process model is "connected" to the control model through data conditions. The control model is "connected" to the process model through process activation information contained in the CSPEC.

A data condition occurs whenever data input to a process result in control output. This situation is illustrated in figure below, part of a flow model for an automated monitoring and control system for pressure vessels in an oil refinery. The process check and convert pressure implements the algorithm described in the PSPEC pseudocode shown. When the absolute tank pressure is greater than an allowable maximum, an above pressure event is generated. Note that when Hatley and Pirbhai notation is used, the data flow is shown as part of a DFD, while the control flow is noted separately as part of a control flow diagram. As we noted earlier, the vertical solid bar into which the above pressure event flows is a pointer to the CSPEC. Therefore, to determine what happens when this event occurs, we must check the CSPEC.

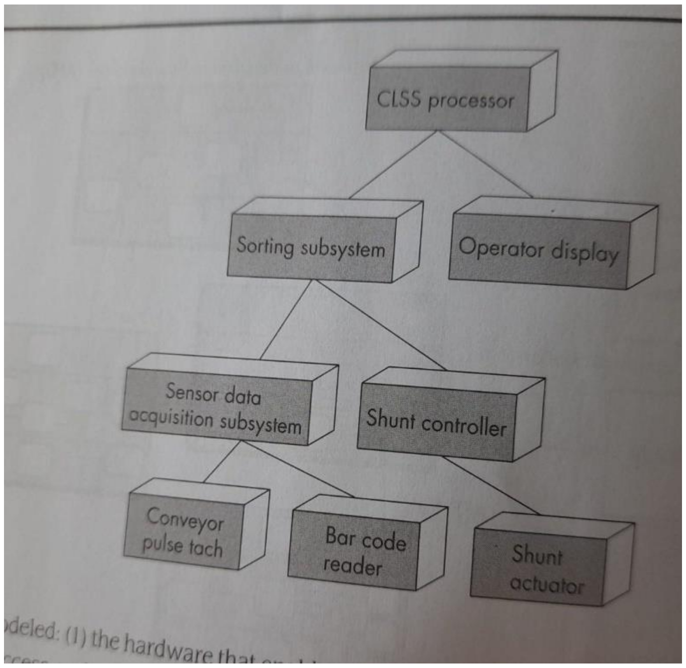
The control specification (CSPEC) contains a number of important modeling tools. A process activation table is used to indicate which processes are activated by a given event. For example, a process activation table (PAT) for the above figure might indicate that the above pressure event would cause a process reduce tank pressure (not shown) to be invoked. In addition to the PAT, the CSPEC may contain a state transition diagram. The STD is a behavioral model that relies on the definition of a set of system states and is described in the following section.

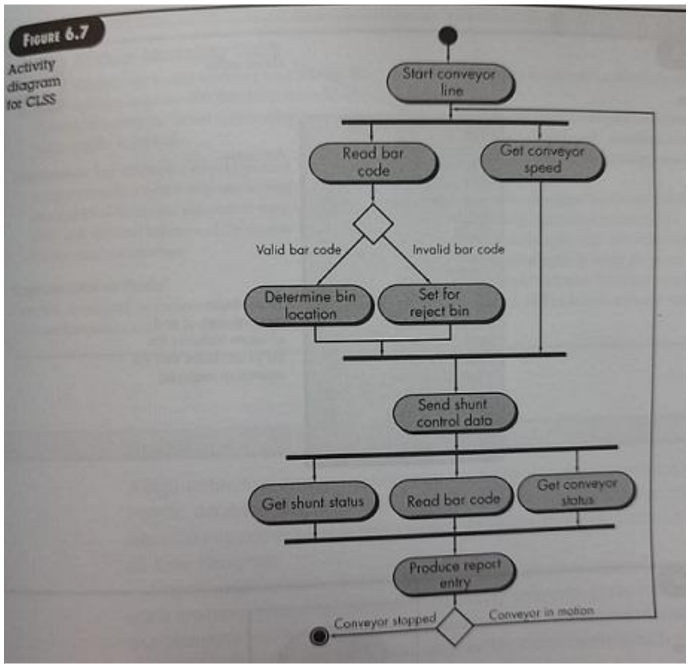


```
If absolute tank pressure > max pressure
    then
        set above pressure to "true";
    else
        set above pressure to "false";
        begin conversion algorithm x-01a;
            compute converted pressure;
        end
endif
```

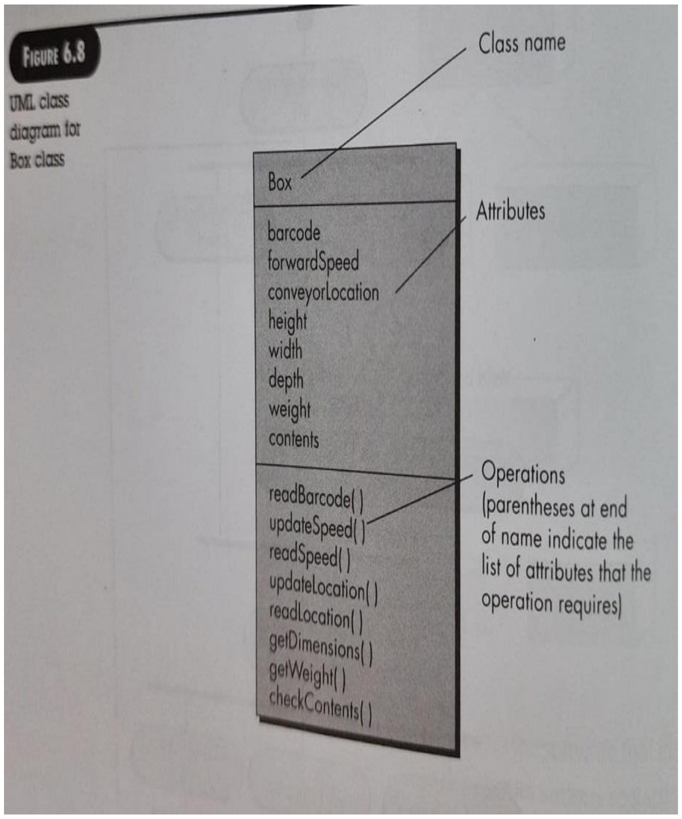


System Modeling with UML: UML provides a wide array of diagrams that can be used for analysis and design at both the system and the software level.for the CLSS system, four important system elements are modeled:1) the hardware that enables CLSS;2)the software that implements data-base acess and sorting;3)the operator who submits various requests to the system;and4)the database that contains relevant bar code and destination information.





CLSS hardware can be modeled at the system level using a UML deployment diagram in the figure



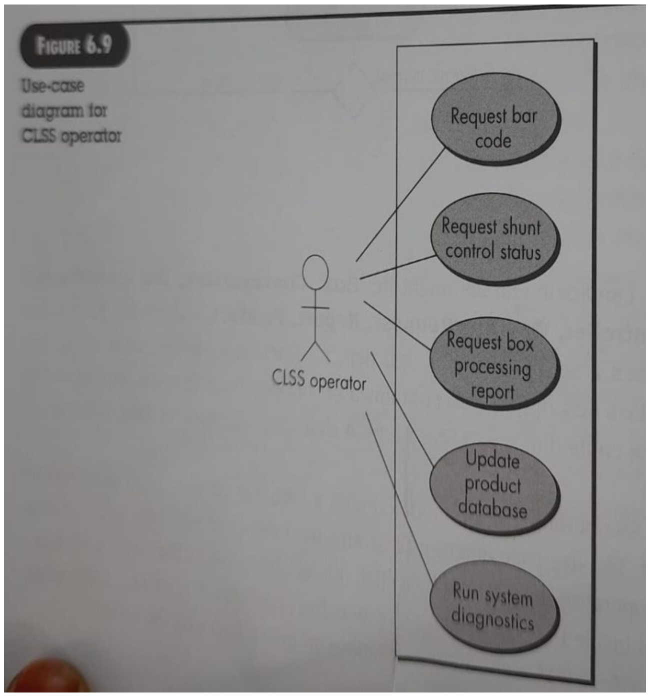


FIGURE 6.9
Use-case diagram for CLSS operator
CLSS operator
Request bar code
Request shunt control status
Request box processing report
Update product database
Run system diagnostics

- **Activity Diagram:** It models the flow of control from one activity to the other. With the help of an activity diagram, we can model sequential and concurrent activities. It visually depicts the workflow as well as what causes an event to occur.

- **Use Case Diagram:** It represents the functionality of a system by utilizing actors and use cases. It encapsulates the functional requirement of a system and its association with actors. It portrays the use case view of a system
- **Deployment Diagram:** It presents the system's software and its hardware by telling what the existing physical components are and what software components are running on them. It produces information about system software. It is incorporated whenever software is used, distributed, or deployed across multiple machines with dissimilar configurations.
- **Class Diagram:** Class diagrams are one of the most widely used diagrams. It is the backbone of all the object-oriented software systems. It depicts the static structure of the system. It displays the system's class, attributes, and methods. It is helpful in recognizing the relation between different objects as well as classes.

➢ Building the analysis model

The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system. The model changes dynamically as software engineers learn more about the system to be built, and stakeholders understand more about what they really require.For that reason, the analysis model is a snapshot of requirements at any given time.

- Analysis model operates as a link between the 'system description' and the 'design model'.
- In the analysis model, information, functions and the behaviour of the system is defined and these are translated into the architecture, interface and component level design in the 'design modeling'.

Elements of the analysis model

1. Scenario based element

- This type of element represents the system user point of view.
- Scenario based elements are use case diagram, user stories.

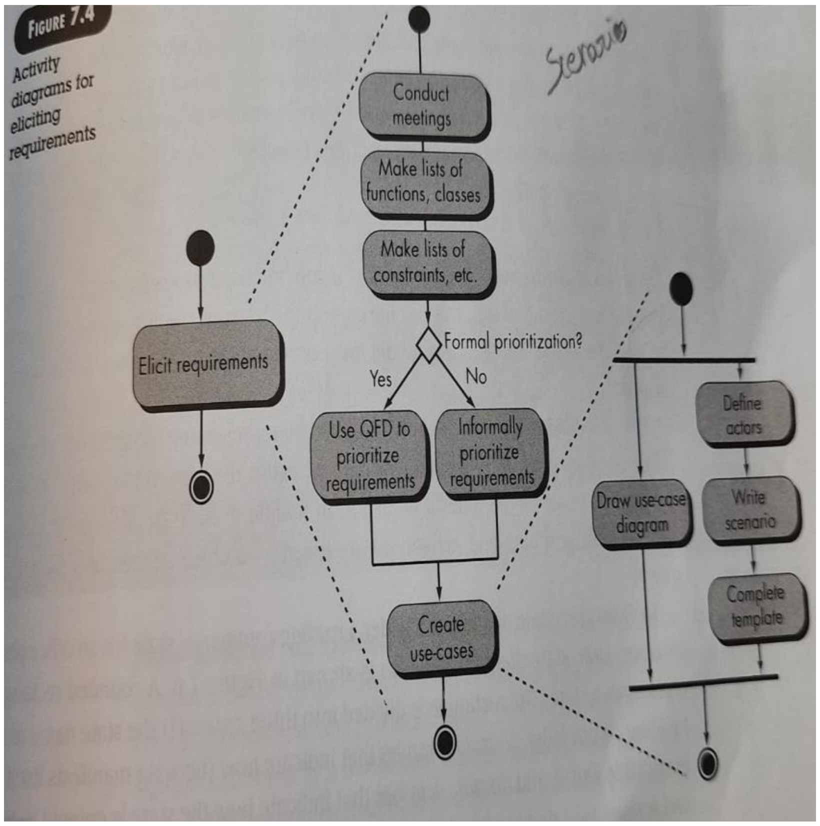
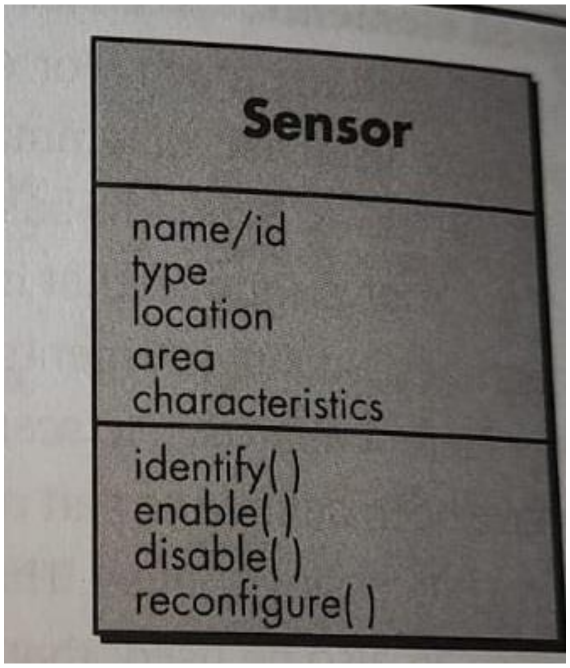


FIGURE 7.4
Activity diagrams for eliciting requirements
Scenario
Elicit requirements
Conduct meetings
Make lists of functions, classes
Make lists of constraints, etc.
Formal prioritization?
Yes
No
Use QFD to prioritize requirements
Informally prioritize requirements
Create use-cases
Define actors
Draw use-case diagram
Write scenario
Complete template

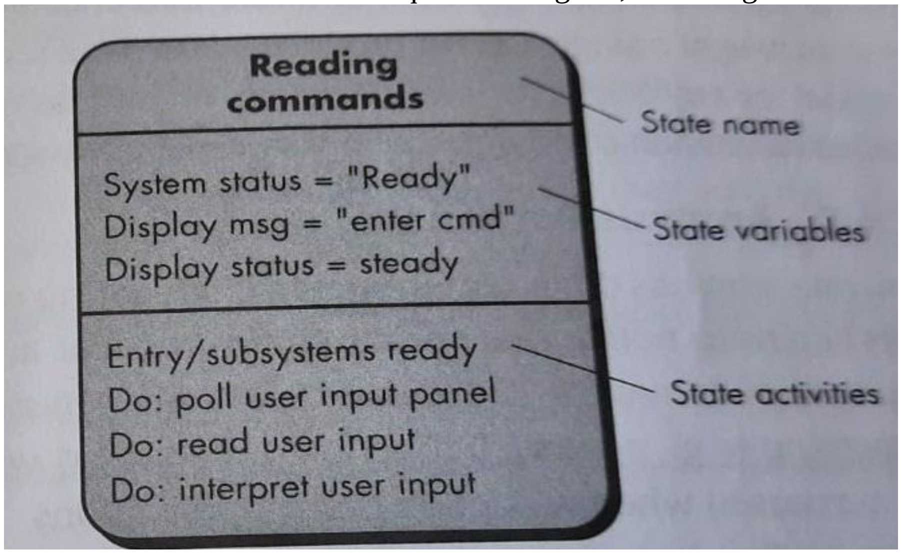


2. Class based elements

- The object of this type of element manipulated by the system.
- It defines the object, attributes and relationship.
- The collaboration is occurring between the classes.
- Class based elements are the class diagram, collaboration diagram.

3. Behavioral elements

- Behavioral elements represent state of the system and how it is changed by the external events.
- The behavioral elements are sequenced diagram, state diagram.



4. Flow oriented elements

- An information flows through a computer-based system it gets transformed.
- It shows how the data objects are transformed while they flow between the various system functions.

- The flow elements are data flow diagram, control flow diagram.

Analysis Rules of Thumb

The rules of thumb that must be followed while creating the analysis model

The rules are as follows:

- The model focuses on the requirements in the business domain. The level of abstraction must be high i.e there is no need to give details.
- Every element in the model helps in understanding the software requirement and focus on the information, function and behaviour of the system.
- The consideration of infrastructure and nonfunctional model delayed in the design. **For example,** the database is required for a system, but the classes, functions and behavior of the database are not initially required. If these are initially considered then there is a delay in the designing.
- Throughout the system minimum coupling is required. The interconnection between the modules is known as 'coupling'.
- The analysis model gives value to all the people related to model.
- The model should be simple as possible. Because simple model always helps in easy understanding of the requirement.

Analysis Patterns

It has been observed that the software engineer ‘reuses’ certain functions, classes, and/ or behavior across all the projects, which may or may not form a part of the specific application domain. For example, features and functions described by a user interface are almost common, regardless of the application domain chosen. Analysis patterns refer to classes, functions, and other features that can be reused when developing applications within a specific application domain. The objectives of analysis patterns are listed below.

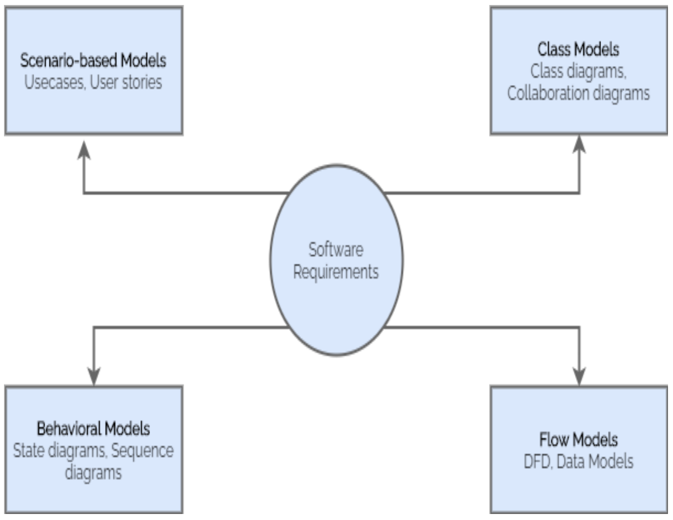
- To quicken the requirements analysis phase by providing reusable analysis models with the description of both advantages and limitations.
- To suggest several design patterns and feasible solutions to common problems in order to help the software designer translate an analysis model into a design model.

Analysis patterns have a unique pattern name, which allows the development team to refer to them with their pattern names. These patterns are stored in a repository so that the software engineer can refer to these patterns and reuse them while developing new software.
Information is stored in the analysis pattern, which can be viewed in the form of a template. The analysis pattern template, which is comprises the following sections.

- **Pattern name:** Consists of important information (in the form of descriptor) about the analysis pattern. This descriptor is used within the analysis model when reference is made to a pattern.
- **Intent:** Describes the problem that is addressed in an analysis pattern, which facilitates the software engineer to use analysis patterns in the specified application domain.
- **Motivation:** Represents how an analysis pattern can be used to address a particular problem in the application domain with the help of a scenario.
- **External issues and contexts:** Lists the external issues that affect the manner in which an analysis pattern works. The external issues include business related subjects, external technical constraints, and so on. In addition, external issues and contexts specify the external issues that are to be resolved by an analysis pattern.
- **Solution:** Describes the procedure by which an analysis pattern is implemented in the analysis model.
- **Consequences:** Describes the implementation effects of an analysis pattern into the application domain. It also provides a description of limitations that arises during its implementation.
- **Design:** Describes the procedure of achieving analysis patterns by using the known design patterns.
- **Known uses:** Provides information about the uses of an analysis pattern within the system.
- **Related patterns:** Lists the analysis patterns that are related to the 'chosen' pattern.

➢ Requirements Analysis:

Requirements analysis leads to the specification of software's operating characteristics, the identification of software's interface with other system elements, and the establishment of constraints that software must meet. Requirements analysis helps you elaborate on basic requirements generated through inception, elicitation, and negotiation requirements engineering tasks.



The action of requirements modeling produces one or more of the following types of models:

- Scenario-based models of requirements from the perspective of various system “actors” Scenario-based models of requirements from the perspective of various system “actors” .
- Data models that depict the problem’s information domain.
- Class-oriented models that represent object-oriented classes (attributes and operations) and how classes collaborate to achieve system requirements.
- Flow-oriented models that represent the system’s functional elements and how they hey transform data as it moves through the system.

Behavioral models that describe how software behaves in response to external “events” that describe how software behaves in response to external “events”

These models provide information to a software designer that can be turned into architectural, interface, and component-level designs.

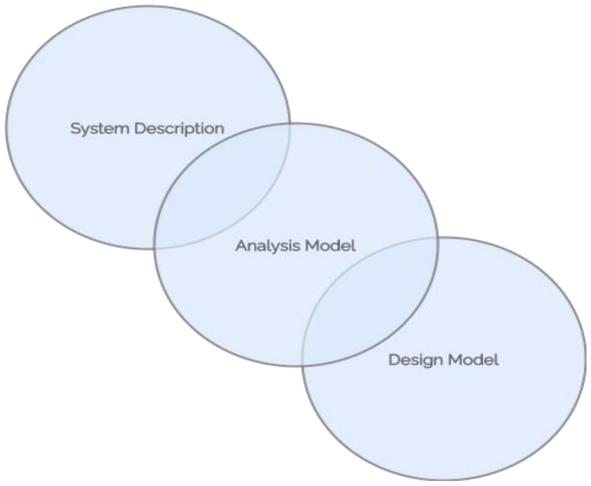
Overall Objectives and Philosophy

Throughout the requirements modeling process, your primary focus should be on what, not how.

The requirements model must accomplish three basic goals:

1. Describing what the client requires.
2. Establishing a foundation for the production of a software design.
3. Defining a set of criteria that can be validated once the software is constructed.

The analysis model bridges the gap between a system-level description of the overall system or business functioning as it is delivered through the use of software, hardware, data, human, and other system aspects and a software design. This relationship is shown in the figure below



Analysis Rules of Thumb

Arlow and Neustadt suggest a few useful rules of thumb to follow when developing the analytical model.

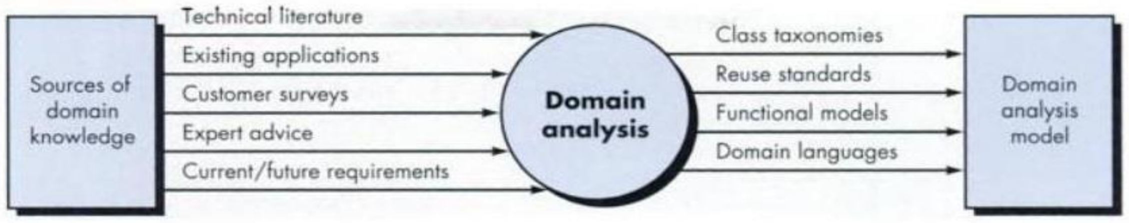
- **The model should concentrate on requirements that are obvious within the problem or business domain.** The level of abstraction should be moderate. “Don’t get bogged down in details” that attempt to explain how the system will function.
- Each element of the requirements **model should contribute to a broader understanding of software needs and provide insight into the system’s information domain, function, and behaviour**.
- **Put off thinking about infrastructure and other nonfunctional models until design**. That is, while a database may be required, the classes required to implement it, the functions required to access it, and the behaviour that will be displayed as it is used should be considered only after the problem domain analysis is complete.
- **Keep coupling to a minimum** across the system. Relationships between classes and functions must be represented. If the level of “interconnectedness” is excessively high, however, efforts should be made to reduce it.
- **Check to see if the requirements model gives value to all stakeholders.** Each constituency has a different application for the model. Business stakeholders, for example, should use the model to validate requirements; designers should use the model as a foundation for design; and QA personnel should use the model to help prepare acceptance tests.

- **Keep the model as simple as possible.** Don't make any new diagrams if they don't contribute any new information. When a basic list will enough, avoid using complex notational forms.

Domain Analysis

Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain…the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks.

FIGURE 8.2 Input and output for domain analysis



Modeling Approaches

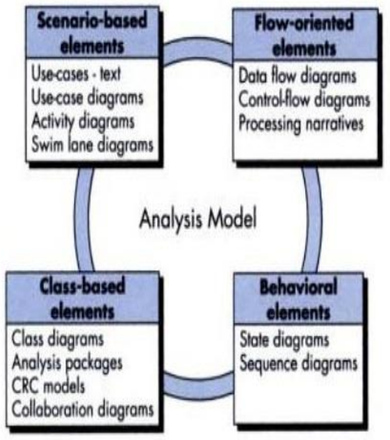
Structured Analysis

Structured analysis is one approach to requirements modeling that treats data and the processes that transform it as separate entities. Data objects are modeled in such a way that their attributes and relationships are defined. Processes that change data items are designed in such a way that it is clear how they transform data as it flows through the system.

Object-Oriented Analysis

The object-oriented analysis focuses on the definition of classes and how they interact with one another. The Unified Process and UML are both mostly object-oriented.

FIGURE 8.3 Elements of the analysis model



- **Data Modeling Concepts**

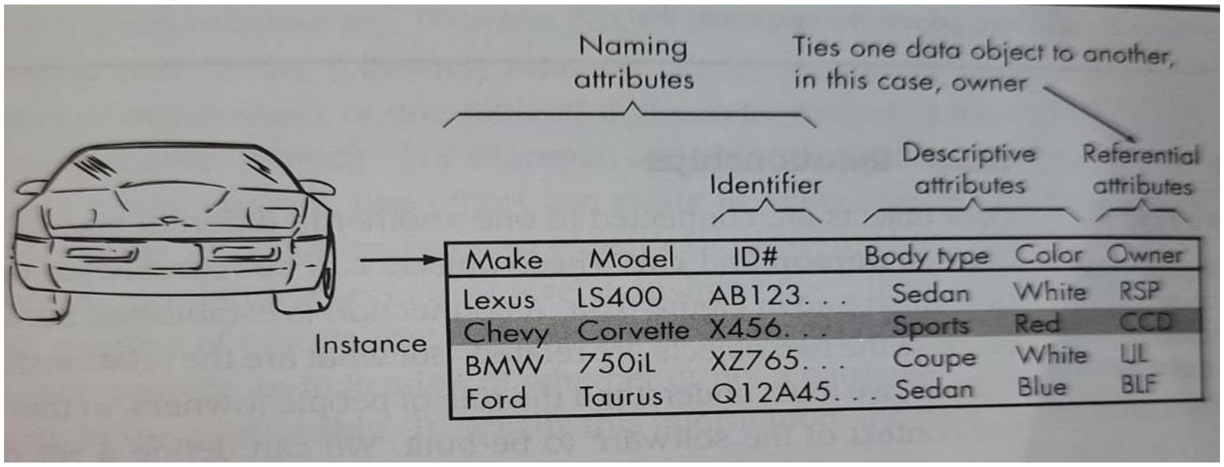
Analysis modeling often begins with *data modeling*. The software engineer or analyst defines all data objects that are processed within the system, the relationships between the data objects, and other information that is pertinent to the relationships.

8.3.1 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), an occurrence (e.g., a telephone call) 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). For example, a person or a car 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.

A data object encapsulates data only—there is no reference within a data object to operations that act on the data.[5] Therefore, the data object can be represented as a table as shown in Figure 8.4. 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

Data 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,or3)make reference to another instance in another table.

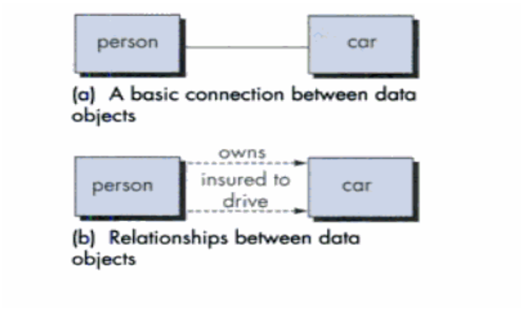
One or more of the attributes must be defined as an identifier-that is the identifier attribute becomes a "key" when we want to find an instance of the data object.In some cases; values for the identifier(s) are unique, although this is not a requirement.

The set of attributes that is appropriate for a given data object is determined through an understanding of the problem context.The attributes for car might serve well for an application that would be used by a Department of Motor vehicles, but these attributes would be used by a Department of Motor vehicles, but these attributes would be useless for an automobile company that needs manufacturing control software.

Relationships

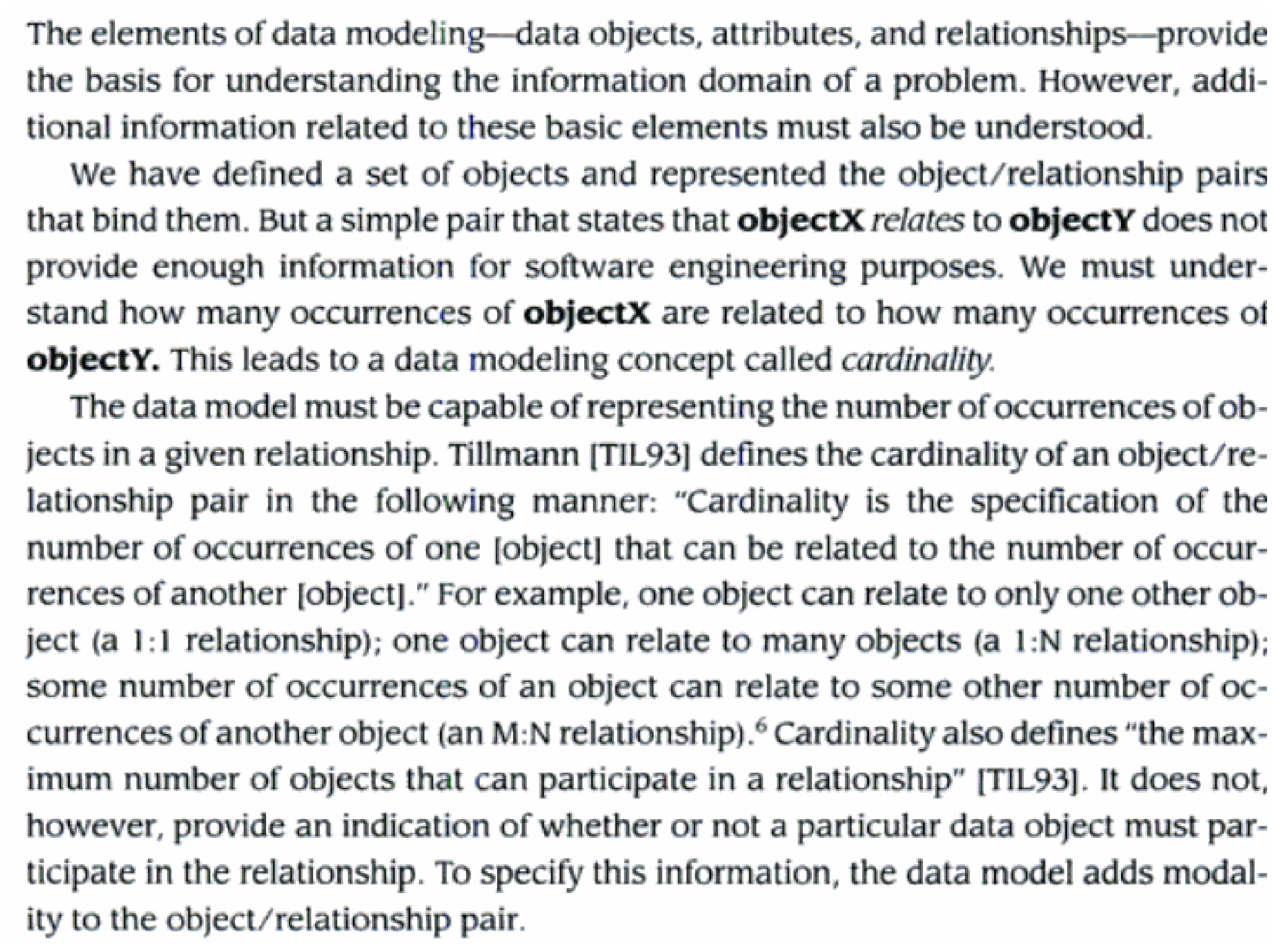
Data objects are connected to one another in different ways.two data objects, person and car.These objects can be represented using the simple notation in figure.a connection is established between person and car because the two objects are related. We can define a set of object/relationship pairs that define the relevant relationships. For example,

- A person owns a car.
- A person is insured to drive a car.


person
car
(a) A basic connection between data objects
person
owns
insured to drive
car
(b) Relationships between data objects

The relationship owns and insured to drive define the relevant connections between person and car. The above fig illustrates these object/relationship pairs graphically. The arrows noted in diagram these provide important information about the directionality of the relationship and often reduce ambiguity or misinterpretations.

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 that **objectX** *relates* to **objectY** does not provide enough information for software engineering purposes. We must understand how many occurrences of **objectX** are related to how many occurrences of **objectY.** This leads to a data modeling concept called *cardinality.*

The data model must be capable of representing the number of occurrences of objects in a given relationship. Tillmann [TIL93] 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]." For example, one object can relate to only one other object (a 1:1 relationship); one object can relate to many objects (a 1:N relationship); some number of occurrences of an object can relate to some other number of occurrences of another object (an M:N relationship).[6] Cardinality also defines "the maximum number of objects that can participate in a relationship" [TIL93]. 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.

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.

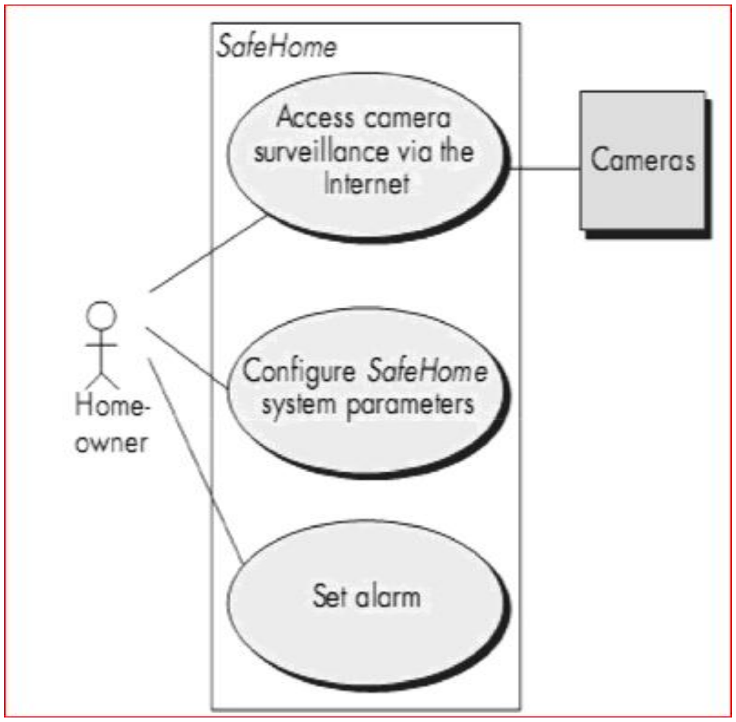
➢ Behavioral model

The system's behaviour can be represented as a function of certain events and time. The behavioural model describes how the software will respond to outside events or stimuli. Go through the following steps, to make the model.

1. Evaluate all use cases to properly understand the system's interaction sequence.
2. Recognize the events that drive the interaction sequence and understand how these events are related to specific objects.
3. For each use case, create a sequence.
4. Create a system state diagram.
5. Go over the behavioural model again to ensure its accuracy and consistency.

Identifying Events with the Use Case

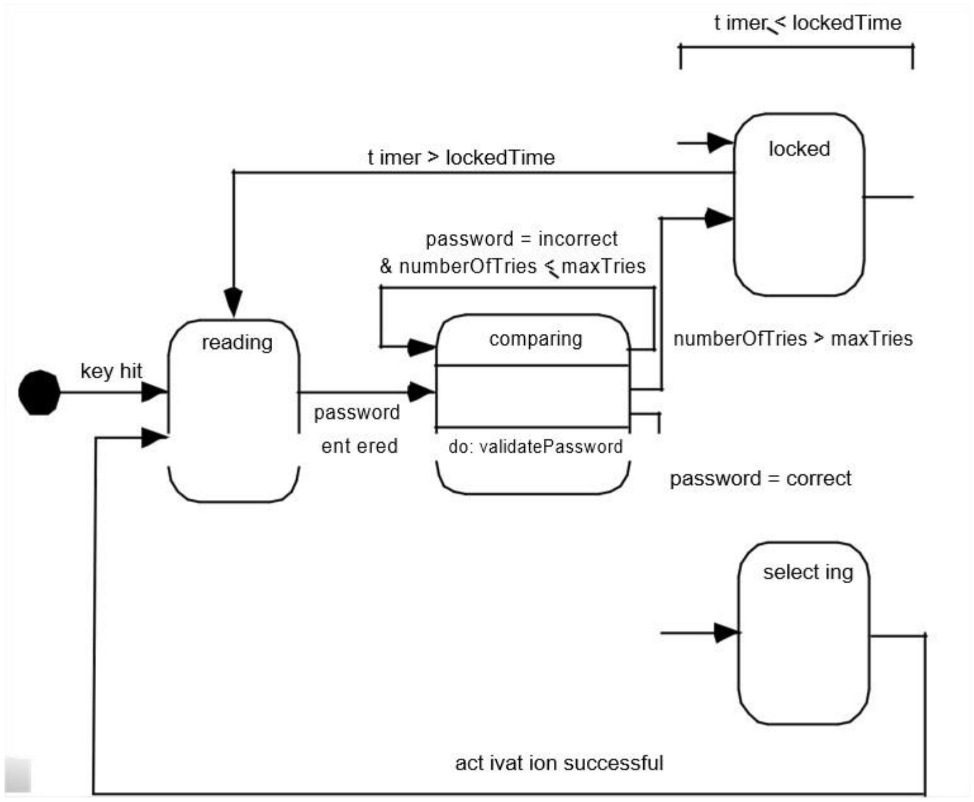
- The use case represents a sequence of activities that involves actors and the system.
- In general, an event occurs whenever the system and an actor exchange information.
- It has been indicated that an event is not the information that has been exchanged, but rather the fact that information has been exchanged.
- For example: The use case for a portion of the Safe Home security function.
- The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further act.
- The underlined portions of the use case scenario indicate events. An actor should be identified for each event; the information that is exchanged should be noted,
- Here the object, Homeowner, transmits an event to the object Control Panel. The event might be called password entered.


SafeHome
Access camera surveillance via the Internet
Cameras
Configure SafeHome system parameters
Home-owner
Set alarm

State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
- The state of each class as the system performs its function and
- The state of the system as observed from the outside as the system performs its function.
- The state of a class takes on both passive and active characteristics.

- A passive state is simply the current status of all of an object's attributes.
- The active state of an object indicates the current status of the object as it undergoes a continuing processing.
- For example, the passive state of the class Player (in the video game application) would include the current position and orientation attributes of Player as well as other features of Player that are relevant to the game .
- The class Player might have the following active states:
- Moving, at rest, injured, being cured;, lost etc.
- An event (sometimes called a trigger) must occur to force an object to make a transition from one active state to another.



State diagram

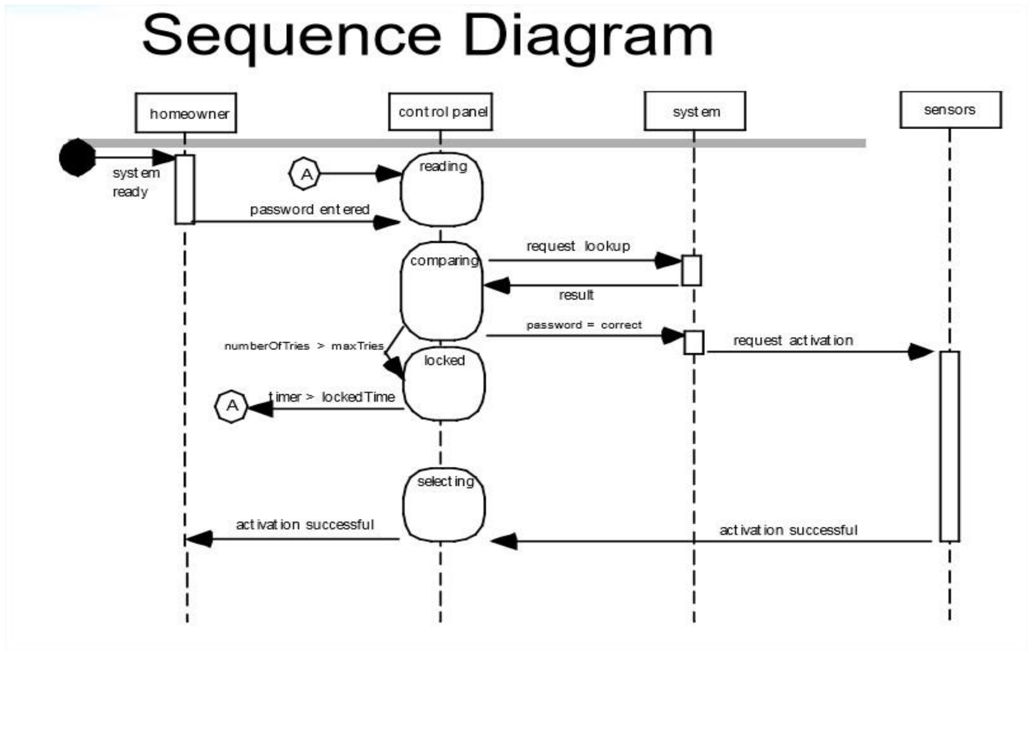
- One component of a behavioral model is a UML state diagram that represents active states for each class and the events (triggers) that cause changes between these active states.
- Figure (In Previous Slide) illustrates a state diagram for the Control Panel object in the Safe Home security function.
- Each arrow shown in Figure represents a transition from one active state of an object to another. The labels shown for each arrow represent the event that triggers the transition.
- An action occurs concurrently with the state transition, generally involves one or more operations (responsibilities) of the object.
- For example, the action connected to the password entered event (ing Figure) is an operation named validate Password () that accesses a password object and performs a digit-by-digit comparison to validate the entered password.

The States of a System

- state—a set of observable circum-stances that characterizes the behavior of a system at a given time.
- state transition—the movement from one state to another.
- event—an occurrence that causes the system to exhibit some predictable form of behavior.
- action—process that occurs as a consequence of making a transition.
- state—a set of observable circum-stances that characterizes the behavior of a system at a given time.
- state transition—the movement from one state to another.

Sequence diagram

- The second type of behavioral representation, called a sequence diagram in UML, indicates how events cause transitions from object to object.
- Once events have been identified by examining a use case, the modeler creates a sequence diagram.
- It is a representation of how events cause flow from one object to another as a function of time.
- The sequence diagram is a shorthand version of the use case. It represents key classes and the events that cause behavior to flow from class to class.
    - For example: Figure (In Next Slide) illustrates a partial sequence diagram for the Safe Home security function.
- Each of the arrows represents an event (derived from a use case)
- It indicates how the event channels behavior between Safe Home objects.
- Time is measured vertically (downward),
- Narrow vertical rectangles represent time spent in processing an activity.



Sequence diagram

- Explanation of Figure: The first event, system ready, is derived from the external environment and channels behavior to the Homeowner object.
- The homeowner enters a password. A request lookup event is passed to System, which looks up the password in a simple database and returns a result (found or not found) to ControlPanel (now in the comparing state).
- A valid password results in a password=correct event to System, which activates Sensors with a request activation event. Ultimately, control is passed back to the homeowner with the activation successful event.