Unit – II SOFTWARE ENGINEERING PROCESS

Functional And Non-Functional - User - System - Requirement Engineering Process - Feasibility Studies - Requirements - Elicitation - Validation and managemen`t - Fundamental of requirement analysis - Analysis principles Software prototyping - Prototyping in the Software Process - Rapid Prototyping Techniques - User Interface Prototyping - Software Document Analysis and Modeling - Data - Functional and Behavioral Models - Structured Analysis and Data Dictionary.

2.1 Functional and Non-Functional Requirements

The analyst starts requirements gathering and analysis activity by collecting all information from the customer which could be used to develop the requirements of the system. He then analyzes the collected information to obtain a clear and thorough understanding of the product to be developed, with a view to removing all ambiguities and inconsistencies from the initial customer perception of the problem.

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?
- What exactly are the data input to the system and what exactly are the data output by the system?
- What are the likely complexities that might arise while solving the problem?
- What are types of data interchange format for interaction with external software or hardware?

The requirements are identified and prepared in a textual format called Software Requirement Specification (SRS) document.

Parts of SRS document are:

- Functional requirements of the system
- Non-functional requirements of the system, and
- Goals of implementation

Functional requirements:-

The functional requirements part discusses the functionalities required from the system. The system is considered to perform a set of high- level functions {*fi*}. The functional view of the system is shown in fig.2.1. Each function f_i of the system can be considered as a transformation of a set of input data (ii) to the corresponding set of output data (*oi*). The user can get some meaningful piece of work done using a high-level function.

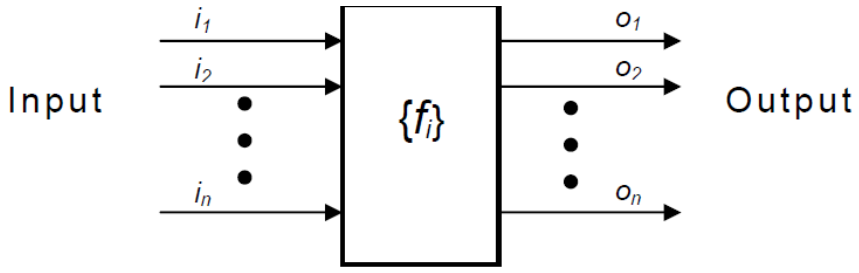


Fig. 2.1: View of a system performing a set of functions

Identifying functional Requirements

Functional requirements need to be identified either from informal problem description document or from a conceptual understanding of the problem. Identify different types of users who might use the system and try to identify the requirements from each user perspective.

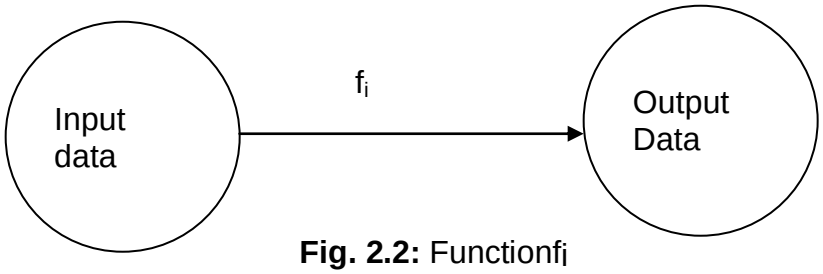


Fig. 2.2: Functionf_i

Figure 2.2 shows that each function f_i is considered as a transformation of a set of input data to some corresponding output data.

Example:-
Consider the case of the library system, where **-**

F1: Search Book function (fig. 2.3)

Input: an author's name

Output: details of the author's books and the location of these books in the library

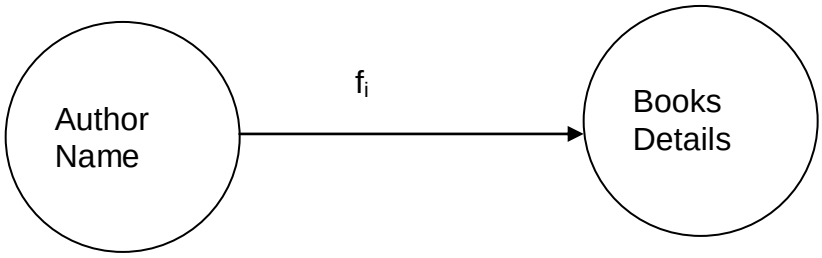


Fig. 2.3: Book Function

So the function Search Book (F1) takes the author's name and transforms it into book details. Functional requirements actually describe a set of high-level requirements, where each high-level requirement takes some data from the user and provides some data to the user as an output. Also each high-level requirement might consist of several other functions.

Documenting functional Requirements

Documenting functional requirements for an ATM system as follows:

- Withdraw cash is a high level requirements.
- It has several sub-requirements.

Example: - Withdraw Cash from ATM

R1: withdraw cash

Description: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash; otherwise it generates an error message.

R1.1 select withdraw amount option

Input: “withdraw amount” option

Output: user prompted to enter the account type

R1.2: select account type

Input: user option

Output: prompt to enter amount

R1.3: get required amount

Input: amount to be withdrawn in integer values greater than 100 and less than 10,000 in multiples of 100.

Output: The requested cash and printed transaction statement.

Processing: the amount is debited from the user’s account if sufficient balance is available, otherwise an error message displayed.

Nonfunctional requirements:-

Nonfunctional requirements deal with the characteristics of the system which can not be expressed as functions - such as the maintainability of the system, portability of the system, usability of the system, etc. Nonfunctional requirements may include:

- reliability issues,
- accuracy of results,
- human - computer interface issues,
- constraints on the system implementation, etc.

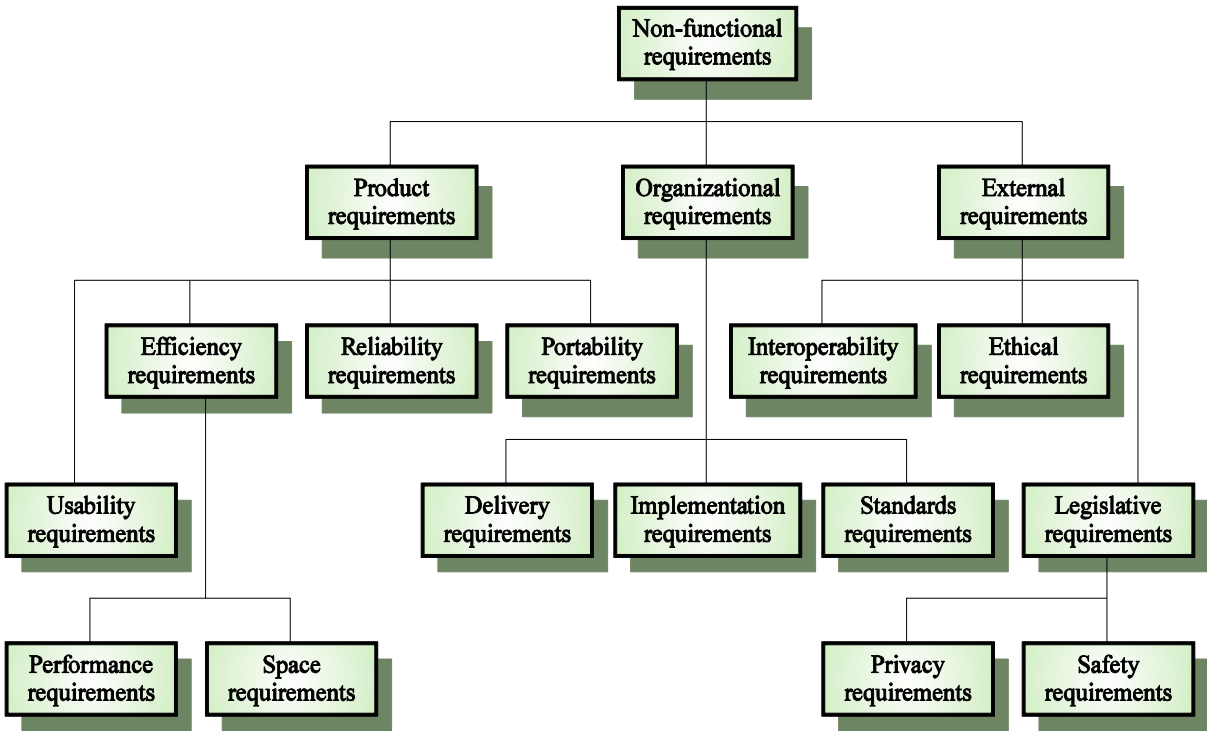


Figure 2.4 Non-functional requirments

Properties of a good SRS document

- The important properties of a good SRS document are the following:
  - **Concise.** The SRS document should be concise and at the same time unambiguous, consistent, and complete. Verbose and irrelevant descriptions reduce readability and also increase error possibilities.
  - **Structured.** It should be well-structured, easy to understand and modify.
  - **Black-box view** SRS document should specify the external behavior of the system and not discuss the implementation issues.
  - **Conceptual integrity.** It should show conceptual integrity so that the reader can easily understand it.
  - **Response to undesired events.** It should characterize acceptable responses to undesired events.
  - **Verifiable.** All requirements of the system as documented in the SRS document should be verifiable. This means that it should be possible to

determine whether or not requirements have been met in an implementation.

Problems without a SRS document

- The system would not be implemented according to customer needs.
- Software developers would not know whether what they are developing is what exactly required by the customer.
- Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
- It will be very much difficult for user document writers to write the users' manuals properly.

Problems with an unstructured specification

- It would be very much difficult to understand that document.
- It would be very much difficult to modify that document.
- Conceptual integrity in that document would not be shown.
- The SRS document might be unambiguous and inconsistent.

2.2 User requirements:

- Statements of what services the system is expected to provide to system users and the constraints under which it must operate, in a natural language plus diagrams

2.3 System requirements:

- More detailed descriptions of the software system's functions, services and operational constraints.
- The system requirements document / functional specification defines exactly what is to be implemented. It may be part of a contract between the system buyer and the software developers.

2.4 Requirement Engineering Process

- Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system
- The Requirement engineering process is accomplished through the execution of seven distinct functions
    - Inception
    - Elicitation
    - Elaboration
    - Negotiation
    - Specification
    - Validation and management

2.4.1 Feasibility Study

A feasibility study decides whether or not the proposed system is worthwhile

- A short focused study that checks
    - If the system contributes to organisational objectives;
    - If the system can be engineered using current technology and within budget;
    - If the system can be integrated with other systems that are used.
- In a feasibility study we need to concentrate our attention on four primary areas of interest:
    1. **Economic feasibility**. An evaluation of development cost weighed against the ultimate income or benefit derived from the developed system or product.
    2. **Technical feasibility**. A study of function, performance, and constraints that may affect the ability to achieve an acceptable system.
    3. **Legal feasibility**. A determination of any infringements, violation, or liability that could result from development of the system.

4. **Alternatives**. An evaluation of alternative approaches to the development of the system or product.

The feasibility study results in a written document called the Feasibility Report.

1. Executive Summary - Management Summary and Recommendations
   A summary of important findings and recommendations for further system development. Should be maximum of one page and self contained, i. e., no references to tables or figures.
2. Introduction
   A brief statement of the problem, the computing environment in which the system is to be implemented and constraints that affect the project. The scope of the problem should be defined.
3. Background
   Discuss what others have done in relation to the problem. Define terms and other information which will be needed as background in order for the reader to understand the report.
4. Alternatives
   A presentation of alternative system configurations; state the criteria that were used in selecting the final approach.
5. System Description
   An abbreviated statement of scope of the system. Feasibility of allocated elements.
6. Cost-Benefit Analysis
   An economic justification for the system.
7. Evaluation of Technical Risk
   A presentation of technical feasibility.
8. Legal Ramifications

2.4.2 Inception

- How does a software project get started?
- At project inception, you establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

2.4.3 Elicitation.

Ask the customer, the users, and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis.

- **Problems of scope**. The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives
- **Problems of understanding**. The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs to the system engineer.
- **Problems of volatility**. The requirements change over time.

2.4.3 Elaboration

- The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.
- This task focuses on developing a refined requirements model that identifies various aspects of software function, behavior, and information.

2.4.5 Negotiation

- It isn't unusual for customers and users to ask for more than can be achieved, given limited business resources.
- It's also relatively common for different customers or users to propose conflicting requirements, arguing that their version is "essential for our special needs."
- You have to reconcile these conflicts through a process of negotiation.
- Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction

2.4.6 Specification

- A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.
- The specification is the final work product produced by the reuirement engineer.

2.4.7 Validation

- Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.
- The primary requirements validation mechanism is the formal technical review

Requirements management

- Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds
- Once requirements have been identified, traceability tables are developed Among many possible traceability tables are the following:
    - **Features traceability table**. Shows how requirements relate to important customer observable system/product features.
    - **Source traceability table -** Identifies the source of each requirement.
    - **Dependency traceability table -** Indicates how requirements are related to one another.
    - **Subsystem traceability table** - Categorizes requirements by the subsystem(s) that they govern.
    - **Interface traceability table -** Shows how requirements relate to both internaland external system interfaces.

Requirement	Specific aspect of the system or its environment								
	A01	**A02**	**A03**	**A04**	**A05**				**Aii**
R01			✓		✓				
R02	✓		✓						
R03	✓			✓					✓
R04		✓			✓				
R05	✓	✓		✓					✓
Rnn	✓		✓						

Feasibility Study

A feasibility study decides whether or not the proposed system is worthwhile or not.

A short focused study that checks

- If the system contributes to organisational objectives;
- If the system can be engineered using current technology and within budget;
- If the system can be integrated with other systems that are used.

In a feasibility study we need to concentrate our attention on four primary areas of interest:

a) **Economic feasibility**. An evaluation of development cost weighed against the ultimate income or benefit derived from the developed system or product.

b) **Technical feasibility**. A study of function, performance, and constraints that may affect the ability to achieve an acceptable system.

c) **Legal feasibility**. A determination of any infringements, violation, or liability that could result from development of the system.

d) **Alternatives**. An evaluation of alternative approaches to the development of the system or product.

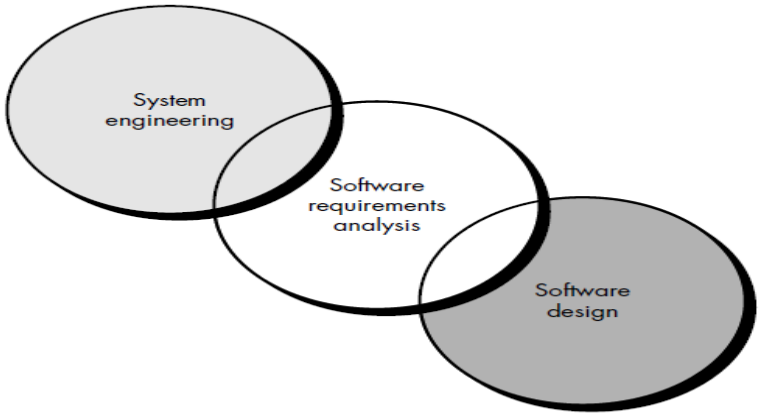
The feasibility study results in a written document called the Feasibility Report.

a) Executive Summary - Management Summary and Recommendations
   A summary of important findings and recommendations for further system development. Should be maximum of one page and self contained, i. e., no references to tables or figures.
b) Introduction
   A brief statement of the problem, the computing environment in which the system is to be implemented and constraints that affect the project. The scope of the problem should be defined.
c) Background
   Discuss what others have done in relation to the problem. Define terms and other information which will be needed as background in order for the reader to understand the report.
d) Alternatives
   A presentation of alternative system configurations; state the criteria that were used in selecting the final approach.
e) System Description
   An abbreviated statement of scope of the system. Feasibility of allocated elements.
f) Cost-Benefit Analysis
   An economic justification for the system.
g) Evaluation of Technical Risk
   A presentation of technical feasibility.
h) Legal Ramifications
i) Other Project-Specific Topics

2.5 Fundamental of requirement analysis

- Requirements analysis is a software engineering task that bridges the gap betweensystem level requirements engineering and software design
- Requirements engineering activities result in the specification of software's operational characteristics, indicate software's interface with other system elements, and establish constraints that software must meet.

- Requirements analysis allows the software engineer (sometimes called *analyst* in this role) to refine the software allocation and build models of the data, functional, and behavioral domains that will be treated by software.
- Requirements analysis provides the software designer with a representation of information, function, and behavior that can be translated to data, architectural, interface, and component-level designs.
- Finally,the requirements specification provides the developer and the customer with the means to assess quality once software is built.



REQUIREMENTS ELICITATION FOR SOFTWARE

- Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation process.

Initiating the Process

The most commonly used requirements elicitation technique is to conduct a meeting or interview. The analyst starts by asking context-free questions. For example, the analyst might ask:

a) Who is behind the request for this work?
b) Who will use the solution?
c) What will be the economic benefit of a successful solution?
d) Is there another source for the solution that you need?

The next set of questions enables the analyst to gain a better understanding of the problem and the customer to voice his or her perceptions about a solution:

a) How would you characterize "good" output that would be generated by a successful solution?

b) What problem(s) will this solution address?
c) Can you show me (or describe) the environment in which the solution will be used?
d) Will special performance issues or constraints affect the way the solution is approached?

The final set of questions focuses on the effectiveness of the meeting.

a) Are you the right person to answer these questions? Are your answers "official"?
b) Are my questions relevant to the problem that you have?
c) Am I asking too many questions?
d) Can anyone else provide additional information?
e) Should I be asking you anything else?

Facilitated Application Specification Techniques (FAST)

Facilitated application specification techniques (FAST), this approach encourages the creation of a joint team of customers and developers who work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements . The basic guidelines:

- A meeting is conducted at a neutral site and attended by both software engineers and customers.
- Rules for preparation and participation are established.
- An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting.
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used.
- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal.

Quality Function Deployment (QFD)

- Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software.
- QFD “concentrates on maximizing customer satisfaction from the software engineering process”
- QFD identifies three types of requirements

I. Normal requirements.

- The objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied.
- Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance.

II. Expected requirements.

- These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction.
- Examples of expected requirements are: ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation.

III. Exciting requirements.

- These features go beyond the customer’s expectations and prove to be very satisfying when present.
- For example, software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multitouch screen, visual voice mail) that delight every user of the product

Use-Cases

- It is difficult to move into more technical software engineering activities until you understand how these functions and features will be used by different classes of end users.
- To accomplish this, developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed. The scenarios,

often called **use cases** ,provide a description of how the system will be used.

- To create a use-case, the analyst must first identify the different types of people (or devices) that use the system or product. These actors actually represent roles that people (or devices) play as the system operates.
- An actor is anything that communicates with the system or product and that is external to the system itself.

2.6 Analysis Principles

All analysis methods are related by a set of operational principles:

a) The information domain of a problem must be represented and understood.
b) The functions that the software is to perform must be defined.
c) The behavior of the software (as a consequence of external events) must be represented.
d) The models that depict information function and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.
e) The analysis process should move from essential information toward implementation detail.

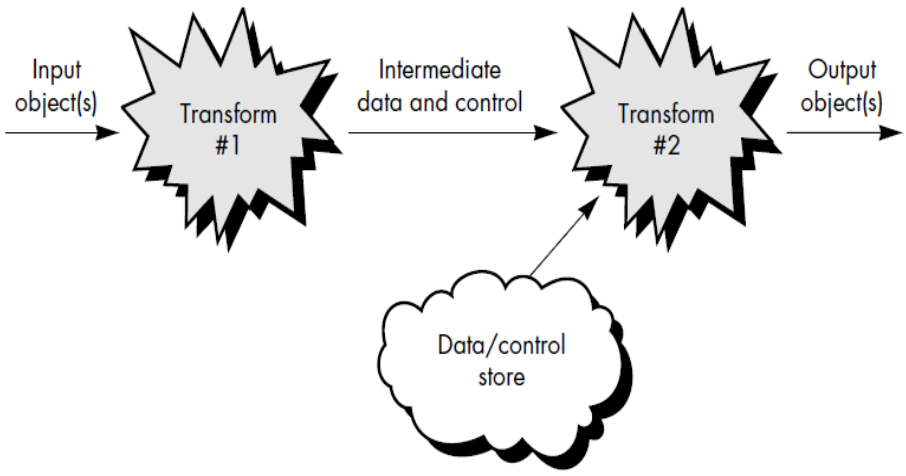
Davis suggests a set of guiding principles for requirements engineering:

a) Understand the problem before you begin to create the analysis model.
b) Develop prototypes that enable a user to understand how human/machine interaction will occur..
c) Record the origin of and the reason for every requirement.
d) Use multiple views of requirements. Building data, functional, and behavioral models provide the software engineer with three different views.
e) Rank requirements. If an incremental process model is applied, those requirements to be delivered in the first increment must be identified.
f) Work to eliminate ambiguity. Because most requirements are described in a natural language. The use of formal technical reviews is one way to uncover and eliminate ambiguity

The Information Domain

- Software is built to process data, to transform data from one form to another; that is, to accept input,manipulate it in some way, and produce output.
- Software also processes events.
- An event represents some aspect of system control and is really nothing more than Boolean data—it is either on or off, true or false, there or not there.
- For example, a pressure sensor detects that pressure exceeds a safe value and sends an alarm signal to monitoring software.
- The information domain contains three differentviews of the data and control as each is processed by a computer program:
    1. Information content and relationships
    2. Information flow
    3. Information structure

- **Information content** represents the individual data and control objects that constitute some larger collection of information transformed by the software.
- For example, the data object, **paycheck,** is a composite of a number of important pieces ofdata: the payee's name, the net amount to be paid, the gross pay, deductions, and so forth.
- Therefore, the content of **paycheck** is defined by the attributes that are needed to create it.
- **Information flow**represents the manner in which data and control change as eachmoves through a system
- **Information structure**represents the internal organization of various data and controlitems.



Modeling

- We create functional models to gain a better understanding of the actual entity to be built.
- It must be capable of representing the information that software transforms, the functions that enable the transformation to occur and the behavior of the system as the transformation is taking place.

Functional models.

- Software transforms information, and in order toaccomplish this, it must perform at least three generic functions: input, processing,and output.
- The functional model begins with a single context level model. Over a series of iterations, more and more functional detail is provided, until a thorough delineation of all system functionality is represented.

Behavioral models.

- Most software responds to events from the outside world.
- This stimulus/response characteristic forms the basis of the behavioral model.
- A computer program always exists in some state—an externally observable mode of behavior that is changed only when some event occurs

Models created during requirements analysis serve a number of important roles:

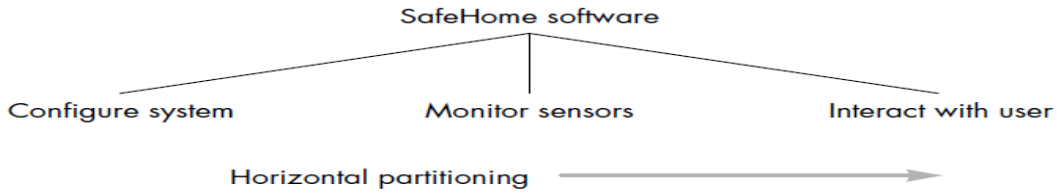
- The model aids the analyst in understanding the information, function, and behavior of a system, thereby making the requirements analysis task easier and more systematic.
- The model becomes the focal point for review and, therefore, the key to a determination of completeness, consistency, and accuracy of the specifications.
- The model becomes the foundation for design, providing the designer with an essential representation of software that can be "mapped" into an implementation context.

Partitioning

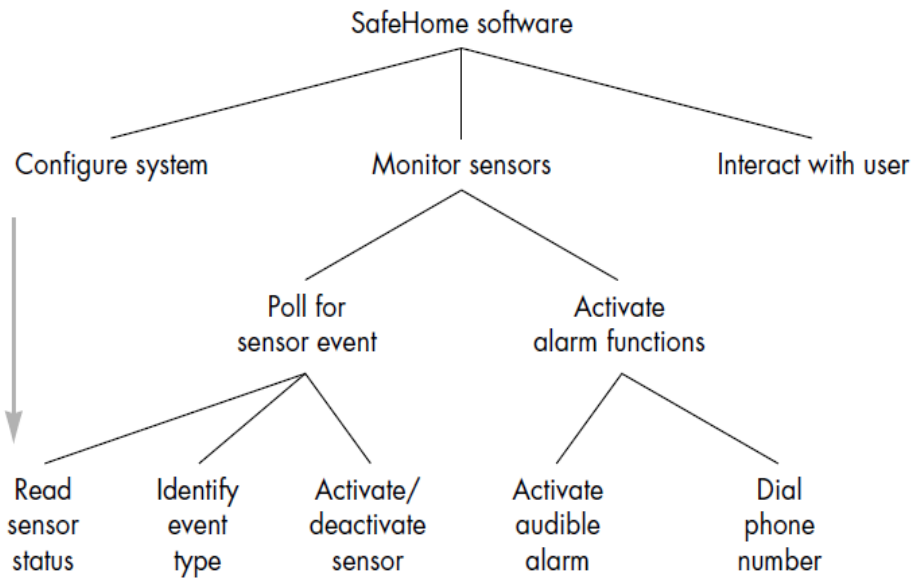
- Problems are often too large and complex to be understood as a whole.
- For this reason, we tend to partition such problems into parts that can be easily understood and establish interfaces between the parts so that overall function can be accomplished.
- We establish a hierarchical representation of function or information and then partition the uppermost element by

  (1) Exposing increasing detail by moving verticallyin the hierarchy or

  (2) Functionally decomposing the problem by moving horizontallyin the hierarchy.



Horizontal partitioning of SafeHome function



Vertical partitioning of Safe Home function

2.7 Software Prototyping

- Rapid software development to validate requirements
- Prototyping is the process of quickly putting together a working model in order to test various aspects of a design, illustrate ideas or features and gather early user feedback

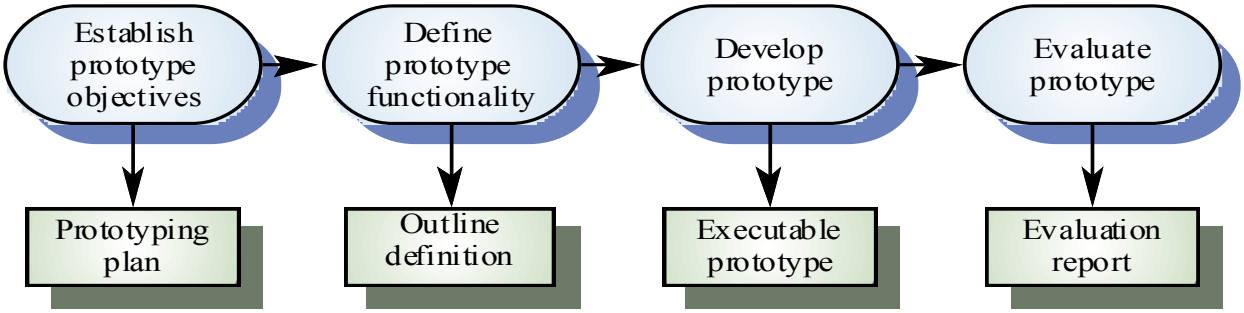
Uses of prototypes

- The principal use is to help customers and developers understand the requirements for the system
- Prototyping can be considered as a risk reduction activity which reduces requirements risks

Prototyping benefits

- Misunderstandings between software users and developers are exposed
- Missing services may be detected and confusing services may be identified
- A working system is available early in the process
- The prototype may serve as a basis for deriving a system specification
- The system can support user training and system testing

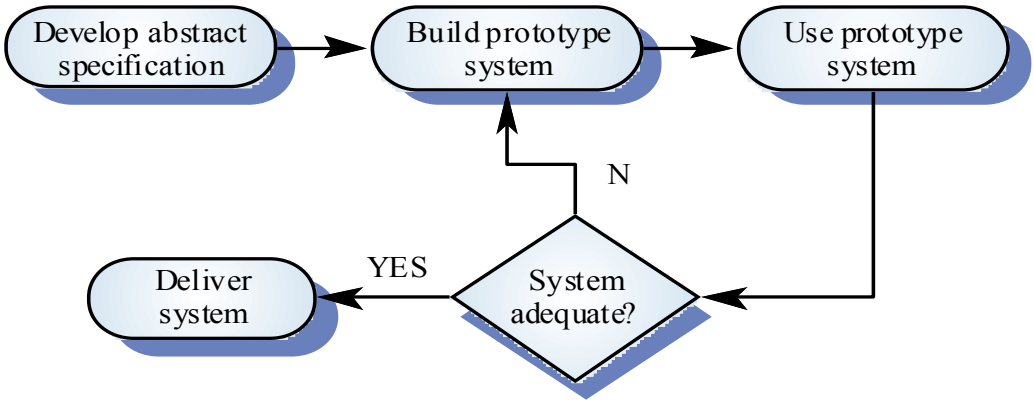
Prototyping process



Types of prototyping

Evolutionary prototyping

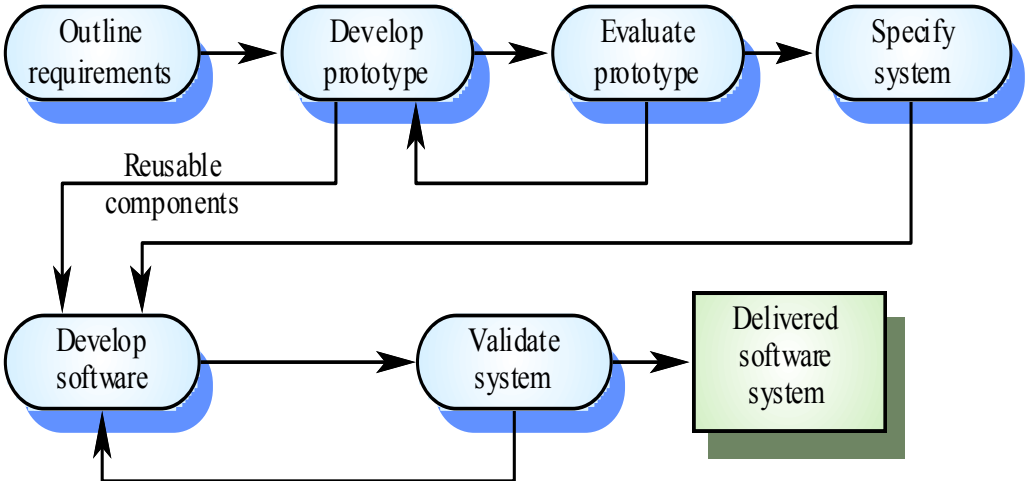
- An open-ended approach, called evolutionary prototyping, uses the prototype as the first part of an analysis activity that will be continued into design and construction.
- The prototype of the software is the first evolution of the finished system.



Evolutionary prototyping

Throw-away prototyping

- The close-ended approach is often called throwaway prototyping.
- Using this approach, a prototype serves solely as a rough demonstration of requirements. It is then discarded, and the software is engineered using a different paradigm.



Throw-away prototyping

Question	Throwaway prototype	Evolutionary prototype	Additional preliminary work required
Is the application domain understood?	Yes	Yes	No
Can the problem be modeled?	Yes	Yes	No
Is the customer certain of basic system requirements?	Yes/No	Yes/No	No
Are requirements established and stable?	No	Yes	Yes
Are any requirements ambiguous?	Yes	No	Yes
Are there contradictions in the requirements?	Yes	No	Yes

Selecting the appropriate prototyping approach

Prototyping Methods and Tools

To conduct rapid prototyping, three generic classes of methods and tools are available:

Fourth generation techniques.

- Fourth generation techniques (4GT)encompass a broad array of database query and reporting languages, program end application generators, and other very high-level nonprocedural languages.

- Because 4GT enable the software engineer to generate executable code quickly, they are ideal for rapid prototyping.

Reusable software components.

- Another approach to rapid prototyping is to assemble, rather than build, the prototype by using a set of existing software components.
- It should be noted that an existing software product can be used as a prototype for a "new, improved" competitive product.

Formal specification and prototyping environments.

Developers of these formal languages are in the process of developing interactive environments that

1. Enable an analyst to interactively create language-based specifications of a system or software,
2. Invoke automated tools that translate the language-based specifications into executable code, and
3. Enable the customer to use the prototype executable code to refine formal requirements.

2.7 User interface prototyping

- It is impossible to pre-specify the look and feel of a user interface in an effective way. prototyping is essential
- UI development consumes an increasing part of overall system development costs
- User interface generators may be used to ‘draw’ the interface and simulate its functionality with components associated with interface entities
- Web interfaces may be prototyped using a web site editor

Techniques

a) **Work with the real users**. The best people to get involved in prototyping are the ones who will actually use the application when it is done. These are the people who have the most to gain from a successful implementation; these are the people who know their own needs best.

b) **Get your stakeholders to work with the prototype**. Just as if you want to take a car for a test drive before you buy it, your users should be able to take an application for a test drive before it is developed. Furthermore, by working with the prototype hands-on, they can quickly determine whether the system will meet their needs. A good approach is to ask them to work through some use case scenarios using the prototype as if it were the real system.

c) **Understand the underlying business**. You need to understand the underlying business before you can develop a prototype that will support it. The more you know about the business, the more likely it is you can build a prototype that supports it. Once again, active stakeholder participation is critical to your success.

d) **You should only prototype features that you can actually build**. If you cannot possibly deliver the functionality, do not prototype it.

e) **You cannot make everything simple**. Sometimes your software will be difficult to use because the problem it addresses is inherently difficult. Your goal is to make your user interface as easy as possible to use, not simplistic.

f) **It's about what you need**. Their point is a good user interface fulfills the needs of the people who work with it. It isn't loaded with a lot of interesting, but unnecessary, features.

g) **Get an interface expert to help you design it**. User interface experts understand how to develop easy-to-use interfaces, whereas you probably do not. A generalizing specialist with solid UI skills would very likely be an ideal member of your development team.

h) **Explain what a prototype is**. The biggest complaint developers have about UI prototyping is their users say "That's great. Install it this afternoon." This happens because users do not realize more work is left to do on the system. The reason this happens is simple: From your user's point-of-view, a fully functional application is a bunch of screens and reports tied together by a menu.

i) **Consistency is critical**. Inconsistent user interfaces lead to less usable software, more programming, and greater support and training costs.

j) **Avoid implementation decisions as long as possible**. Be careful about how you name these user interface items in your requirements documents. Strive to keep the names generic, so you do not imply too much about the implementation technology.

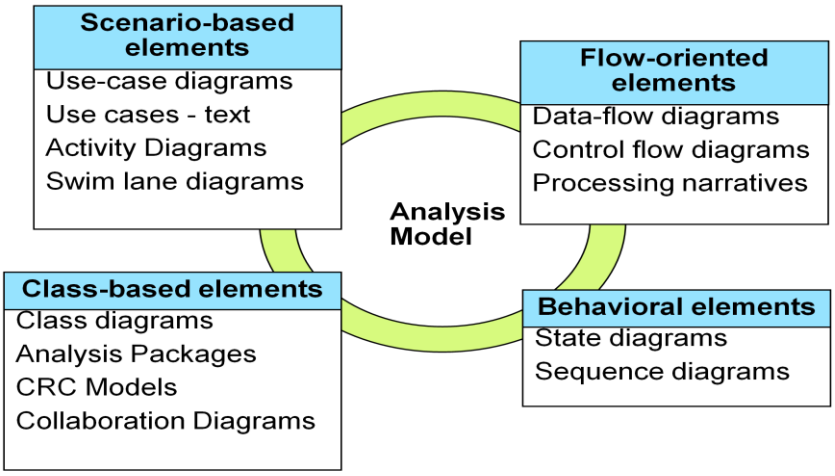
k) **Small details can make or break your user interface**. Have you ever used some software, and then discarded it for the product of a competitor because you didn't like the way it prints, saves files, or some other feature you simply found too annoying to use? I have. Although the rest of the software may have been great, that vendor lost my business because a portion of its product's user interface was deficient.

2.8 Software Document Analysis and Modeling

Goals of Analysis Modeling

- Provides the first technical representation of a system
- Is easy to understand and maintain
- Deals with the problem of size by partitioning the system
- Uses graphics whenever possible
- Differentiates between essential information versus implementation information
- Helps in the tracking and evaluation of interfaces
- Provides tools other than narrative text to describe software logic and policy

Elements of the Analysis Model



2.9 Data Modeling

- Examines data objects independently of processing
- Focuses attention on the data domain
- Creates a model at the customer's level of abstraction
- Indicates how data objects relate to one another

Data modeling concepts

Data Objects

A data object can be an external entity , a thing ,an occurrence or event ,a role, an organizational unit, a place, or a structure.

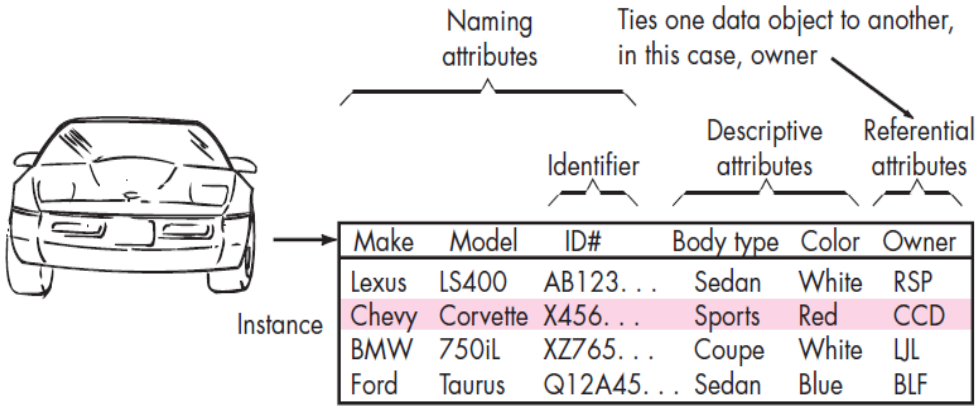
Attributes

- Data attributes define the properties of a data object
- They can be used to

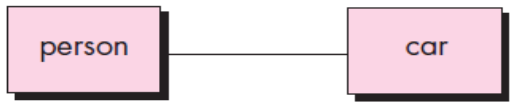
  (1) name an instance of the data object,

  (2) describe the instance, or

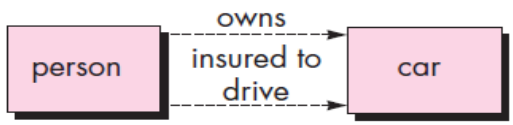
  (3) make reference to another instance in another table
- In addition, one or more of the attributes must be defined as an identifier—that is, the identifier attribute becomes a "key" when we want to find an instance of the data object.



Tabular representation of data objects



(a) A basic connection between data objects



(b) Relationships between data objects

Relationships

- Data objects are connected to one another in different ways.
- Consider the two objects person and car ,a connection is established between them because they are related.

  For example

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

Cardinality and Modality

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

Cardinality.

- The data model must be capable of representing the number of occurrences objects in a given relationship. The cardinality of an object/relationship pair in the following manner:
- Cardinality is the specification of the number of occurrences of one [object] that can be related to the number of occurrences of another [object].
- Cardinality is usually expressed as simply 'one' or 'many. Taking into consideration all combinations of 'one' and 'many,' two [objects] can be related as

- **One-to-one (I:I)**—An occurrence of [object] 'A' can relate to one and only one occurrence of [object] 'B,' and an occurrence of 'B' can relate to only one occurrence of 'A.'
- **One-to-many (I:N)**—One occurrence of [object] 'A' can relate to one or many occurrences of [object] 'B,' but an occurrence of 'B' can relate to only one occurrence of 'A.'
- **Many-to-many (M:N)**—An occurrence of [object] 'A' can relate to one or more occurrences of 'B,' while an occurrence of 'B' can relate to one or more occurrences of 'A.'
- Cardinality defines "the maximum number of objects that can participate in a relationship"It does not, however, provide an indication of whether or not a particular data object must participate in the relationship.

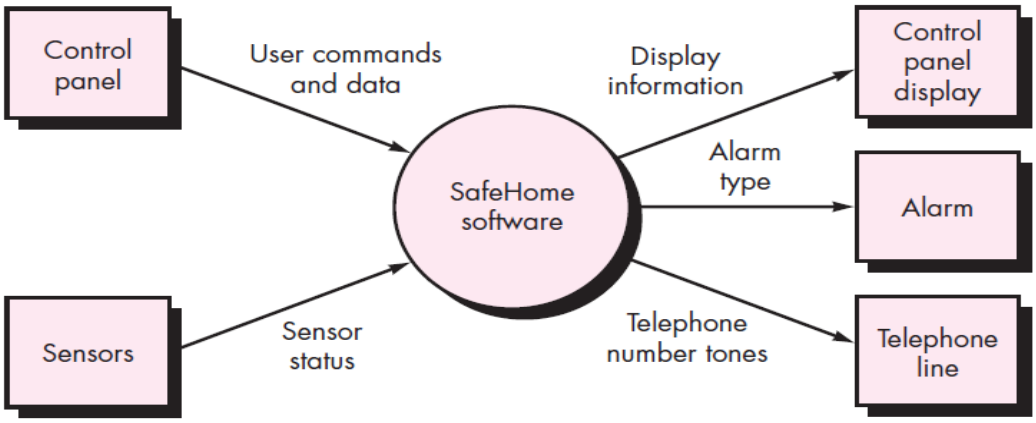
Modality.

- The modality of a relationship is 0 if there is no explicit need for the relationship to occur or the relationship is optional. The modality is 1 if an occurrence of the relationship is mandatory.

2.10 Functional and Behavioral Models

Functional Model - Data Flow Diagram (DFD)

- The DFD takes an input-process-output view of a system. That is data objects flow into the software ,transformed by processing elements,and resultant data objects flow out of the software



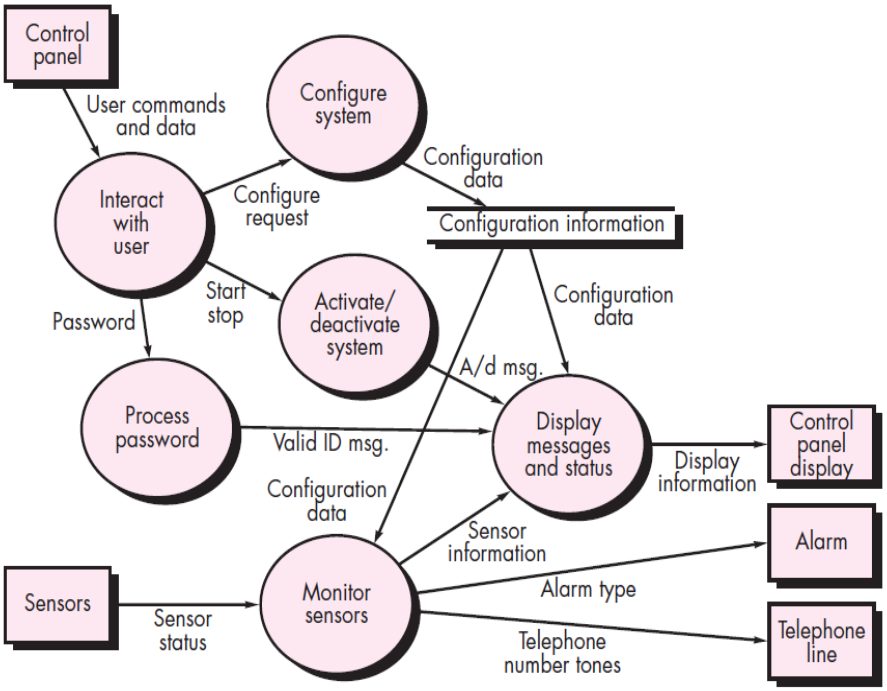
Context level DFD for the SafeHome security function

Guideliness

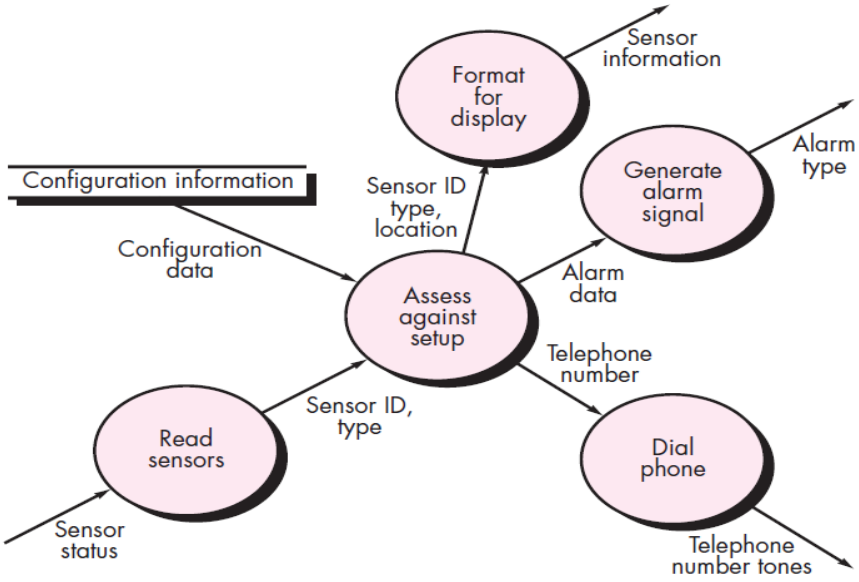
- The level 0 data flow diagram should epict the software /system as a single bubble.
- Primary input and output should be carefully noted.
- Refinement should begin by isolating candidate processes ,data objects and data stores to be represented at the next level.
- All arrows and bubbles should be labelled with meaningful names
- Information flow through continuity must be maintained from level to level and
- One bubble at a time should be refined.

Flow Modeling Notation

- The level 0 DFD is now expanded into a level 1 data flow model.
- An effective approach is to perform a “grammatical parse” on the narrative that describes the context level bubble. That is we isolate all nouns and verbs in a safehome processing narrative derived during the first requirement gathering meeting.
- verbs are safehome processes,that is they may represented as bubbles in a subsequent DFD.
- Nouns are either external entities (boxes),data or control objects(arrows) or data stores(double lines).



Level 1 DFD for Safe Home security function



Level 2 DFD that refines the monitor sensors process

Contol Flow Model

- Large class of applications are driven by events rather than data,produce control information rather than reports or displays and process information with heavy concern for time and performance .
- Such application require the use of control flow modeling in addition to the data flow modeling .

- To select potential candidate events, the following guidelines are suggested:
  - List all sensors that are “read” by the software.
  - List all interrupt conditions.
  - List all “switches” that are actuated by an operator.
  - List all data conditions.
  - Recalling the noun/verb parse that was applied to the processing narrative, review all “control items” as possible control specification inputs/outputs.
  - Describe the behavior of a system by identifying its states, identify how each state is reached, and define the transitions between states.
  - Focus on possible omissions—a very common error in specifying control; for example, ask: “Is there any other way I can get to this state or exit from it?”

The Control Specification

- A control specification (CSPEC) represents the behavior of the system in two different ways.
- The CSPEC contains a state diagram that is a sequential specification of behavior.
- It can also contain a program activation table—a combinatorial specification of behavior.

2.11 Behavioral modeling :State Transition Diagram(STD)

The behavioral model indicates how software will respond to external events or stimuli. **To create the model, you should perform the following steps:**

a) Evaluate all use cases to fully understand the sequence of interaction within the system.

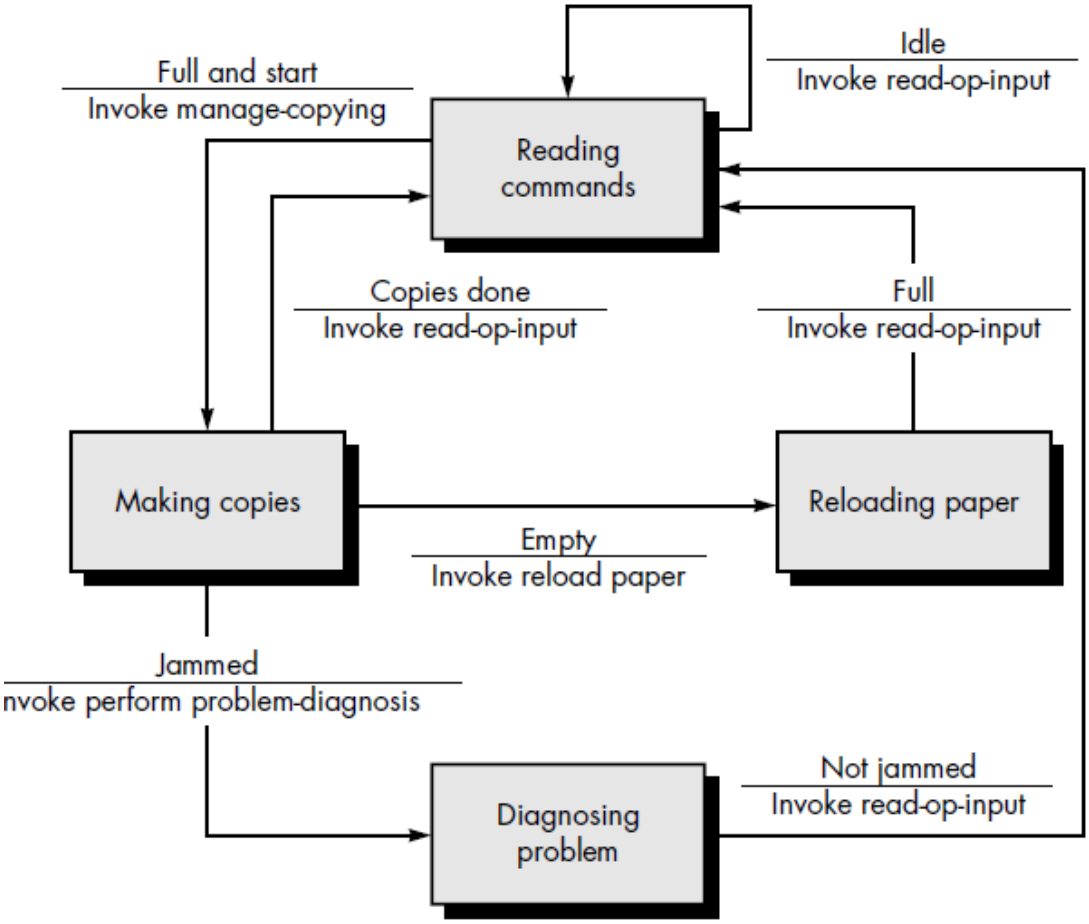
b) Identify events that drive the interaction sequence and understand how these events relate to specific objects.
c) Create a sequence for each use case.
d) Build a state diagram for the system.
e) Review the behavioral model to verify accuracy and consistency.

Identifying Events with the Use Case

- In general, an event occurs whenever the system and an actor exchange information.
- "homeowner uses the keypad to key in a four-digit password." In the context of the requirements model, the object, **Homeowner,** transmits an event to the object **ControlPanel.** The event might be called password entered.
- The information transferred is the four digits that constitute the password, but this is not an essential part of the behavioral model.
- It is important to note that some events have an explicit impact on the flow of control of the use case, while others have no direct impact on the flow of control.
- For example, the event password entered does not explicitly change the flow of control of the use case, but the results of the event password compared (derived from the interaction "password is compared with the valid password stored in the system") will have an explicit impact on the information and control flow of the SafeHome software.

State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered: (1) the state of each class as the system performs its function and (2) 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.


Idle
Invoke read-op-input
Full and start
Invoke manage-copying
Reading commands
Copies done
Invoke read-op-input
Full
Invoke read-op-input
Making copies
Reloading paper
Empty
Invoke reload paper
Jammed
nvoke perform problem-diagnosis
Not jammed
Invoke read-op-input
Diagnosing problem

A simplified state transition diagram for the photocopier software is shown in the above figure. The rectangles represent system states and the arrows represent transitions between states. Each arrow is labeled with a ruled expression. The top value indicates the event(s) that cause the transition to occur. The bottom value indicates the action that occurs as a consequence of the event. Therefore, when the paper tray is **full** and the **start** button is pressed, the system moves from the *reading commands* state to the *making copies* state. Note that states do not necessarily correspond to processes on a one-to-one basis. For example, the state *making copies* would encompass both the *manage copying* and *produce user displays* processes shown in the Figure.

2.12 Structured analysis

Structured analysis consist of three modeling techniques

- Functional model – Data Flow Diagram (DFD)
- Data Model – Entity Relationship Diagram (ERD)
- Behavioral Model – State Transition Diagram (STD)

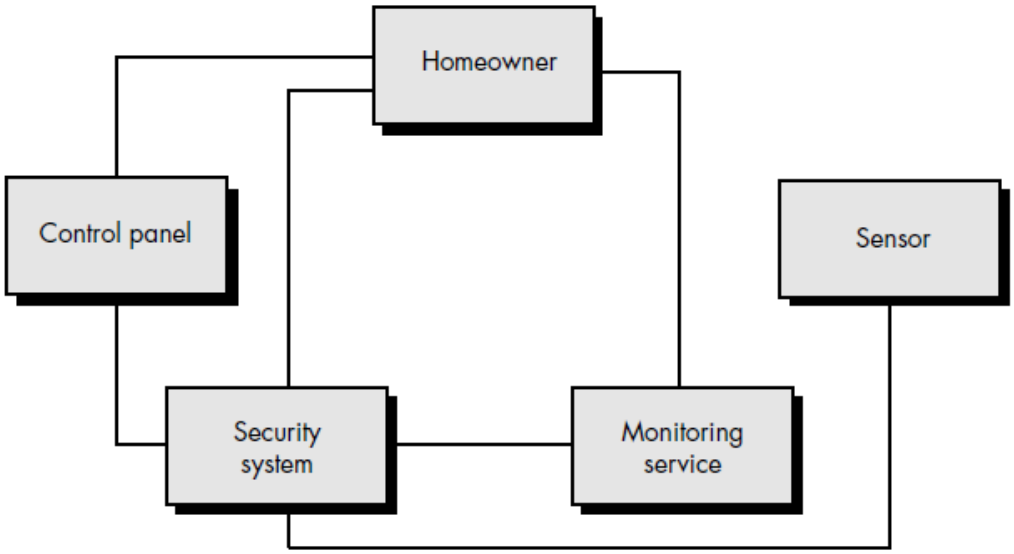
Data model : Entity Relationship Diagram (ERD)

The system analyst and customer identifying the functional requirements and it will be modeled.

Creating Entity - Relationship model (ERD)

ERD is constructed in a iterative fashion. The following steps are required.

1. During requirement elicitation, the customer asked to “Things” which are required. Things evolve input and output data object.
2. Taking the objects one at a time, define connection exist between data objects.
3. If connection exists, create one or more object relationship pairs.
4. For each object /relationship pair, cardinality and modality are explored.
5. Steps 2 to 4 are continue iteratively.
6. The attribute of each entity are defined.
7. ERD is formalized and reviewed.
8. Steps 1 to 7 are repeated until data modeling is complete.



In Safe home software, the things are Homeowner, Control panel, Security System, Monitoring service, Sensor etc. lines connecting the objects are noted. For example, direct connection exist between sensor and security system is noted. Single connection between sensor and security system is identified. One or more object relationship pair are identified.

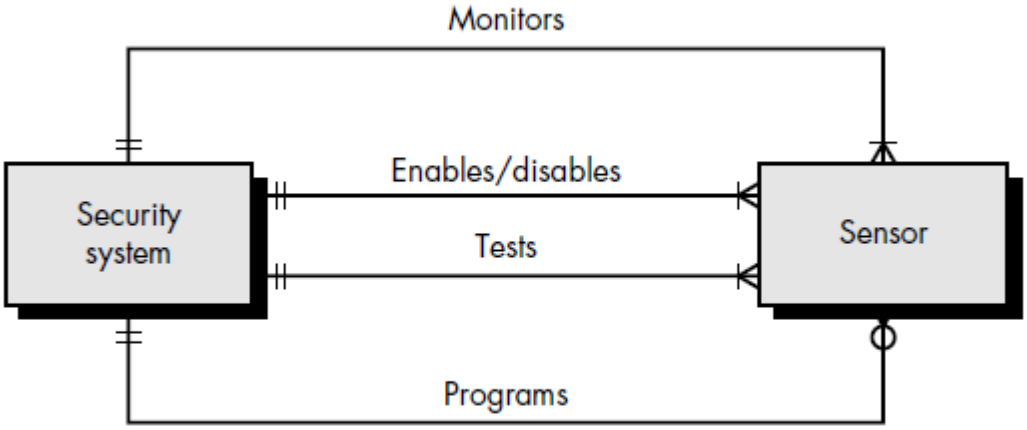
For example ,

Security system monitors sensor

Security system enables/disables sensor

Security system tests sensor

Security system programs sensor



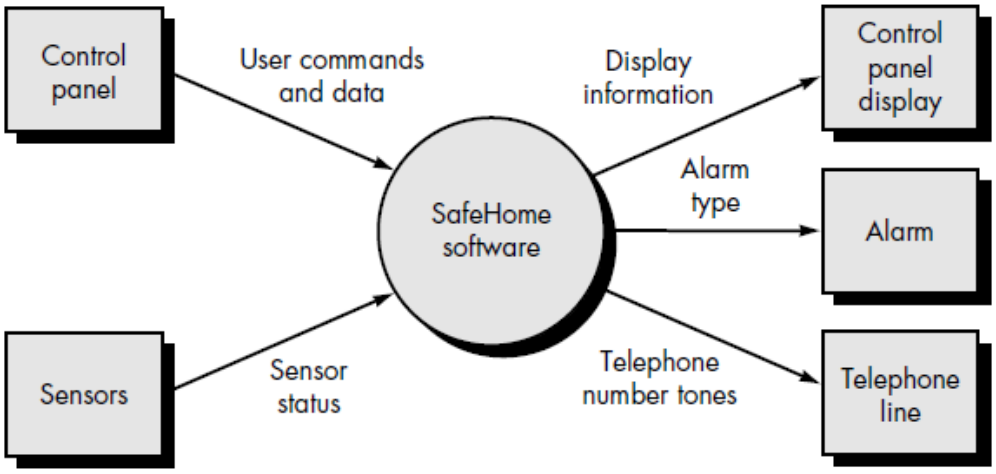
Cardinality and modality between object/relationship has been identified. For example, Security system to sensor is 1:m relationship exists. Each attribute studied with its attributes. For example, sensor object consists of sensor type, internal identification number, zone location and alarm.

Functional model – Data Flow Diagram (DFD)

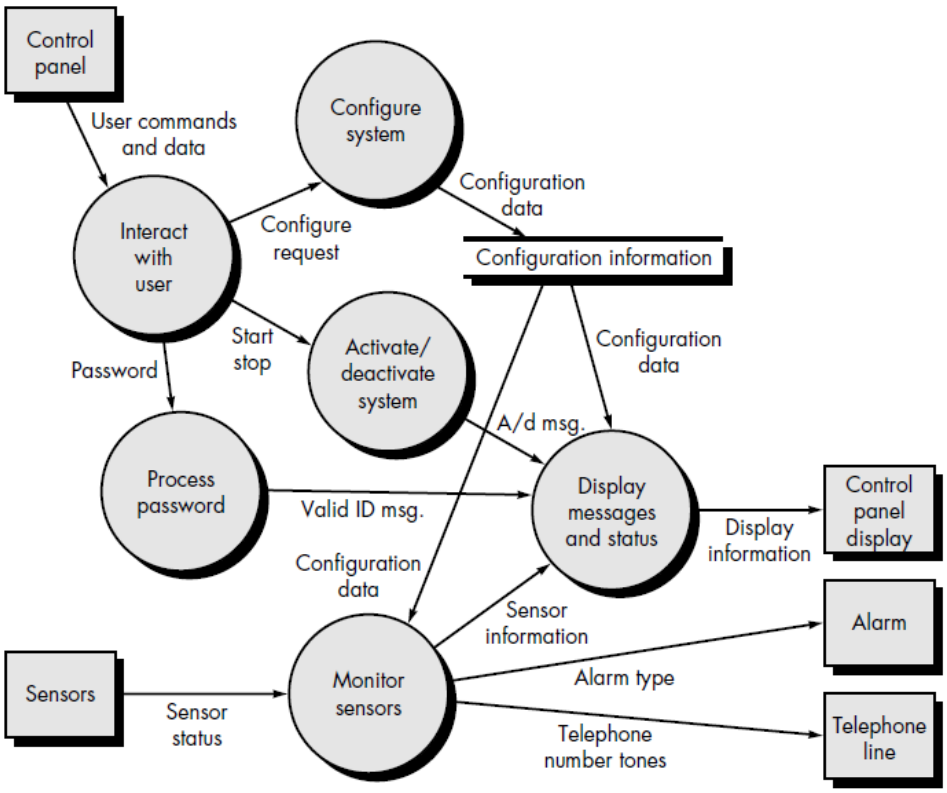
Creating DFD contains the following guidelines.

a) Level 0 DFD depicts as a single bubble.
b) Primary input and output clearly noted.
c) Refinement should begin by isolating candidate process.
d) All arrows and bubbles should be labeled with meaning
e) Information flow continuity must be maintained.
f) One bubble at a time should be refined.

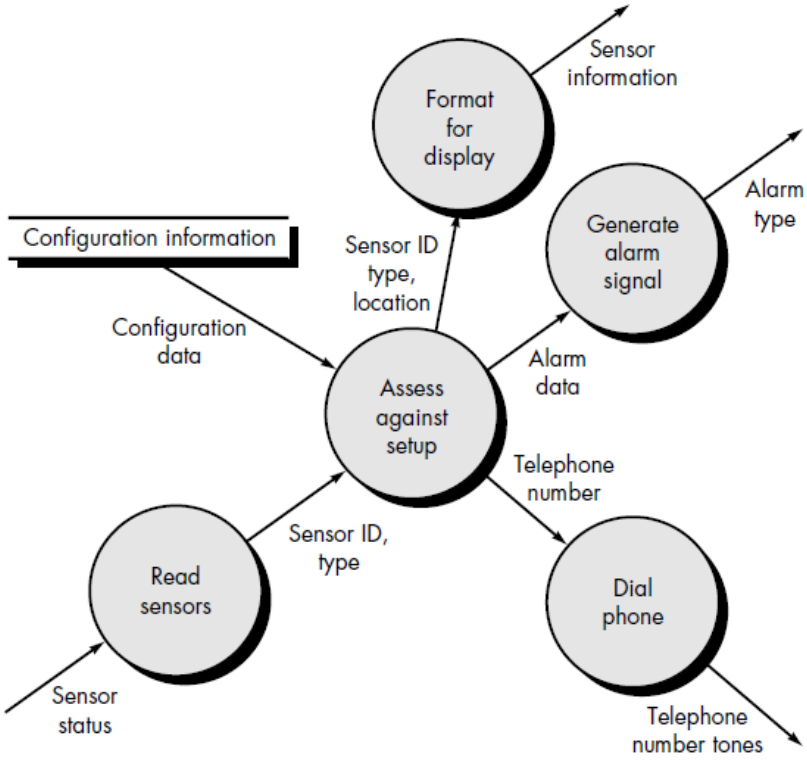
Level 0 DFD of safe home software



Level 1 DFD of Safe Home software

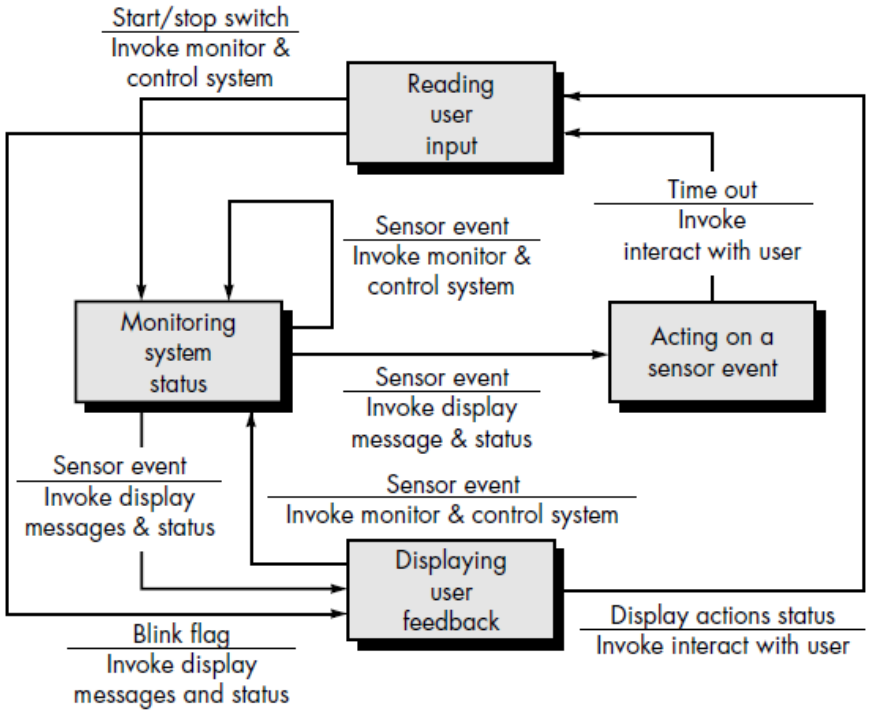


Level 2 DFD of safe Home software



Behavioral Model – State Transition Diagram (STD)

State transition diagram for Safe home software



2.13 Data Dictionary

The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations. the data dictionary is always implemented as part of a CASE "structured analysis and design tool." Although the format of dictionaries varies from tool to tool, most contain the following information:

- Name—the primary name of the data or control item, the data store or an external entity.
- Alias—other names used for the first entry.
- Where-used/how-used—a listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity.

Data Construct	Notation	Meaning
	=	is composed of
Sequence	+	and
Selection	[|]	either-or
Repetition	{ }*n*	*n* repetitions of
	( )	optional data
	* ... *	delimits comments

Appendix – A

Sample SRS document template is available in the following url :

https://www.google.co.in/search?q=software+requirement+specification+template&oq=Sware+requirement+sfication+templatew&gs_l=psy-

Reference Books:

1. Pressman, “Software Engineering and Application”, 6th Edition, Mcgraw International Edition, 2005.
2. Sommerville, ”Software Engineering”, 6th Edition, Pearson Education, 2000.