**CHAPTER # 03**

**REQUIREMENTS ANALYSIS**

* 1. **Requirements Analysis: An Overview**

**The purpose of the requirements analysis is to establish an understanding of the application domain and to capture, formalize, analyze and validate the user requirements on the system to be built**. Once requirements have been gathered (or elicited) the requirements analysis phase is started. The main purpose of this phase is to investigate the gathered requirements in detail, and also uses to categorize requirements and organizes them into related subsets, explores each requirement in relationship to others, examines **requirements for consistency, omissions, and ambiguity and ranks requirements based on the needs of the customer / users. The requirements** analysis acts as a bridge between the system engineering and the software design. **To build something we must first understand what that “Something” is to be. The process of understanding and documenting this something is called requirements analysis.** Requirements generally express what an application is meant to do: generally they do not try to express how to accomplish these functions. The key objective of the requirements analysis is to describe the requirements **in terms of relationships**, provide a basis for the design and to provide a basis for validation for the software after it is built. The requirements analysis phase starts in parallel with requirements elicitation and involves refining and modeling the requirements to identify inconsistencies, errors omissions and other defects. Requirements analysis is usually done with the use of one or more system models that present an abstract description of the system. These models also act as a bridge between the users, customer and the developers as the models are easily understandable by all the parties. The output of requirements analysis is a document generally referred as software requirement specification (SRS).

Requirements analysis provides pointers and inputs to further elicitation. Activities that are performed in requirements analysis often overlap the requirements elicitation activities. Typical activities that may be classified as requirements analysis are:

* Performing feasibility analysis this typically includes estimation of the cost and confirming that the requirements are technically feasible in the given hardware and the software environment.
* Modeling of the requirements which usually consist of various graphical representations of the functions, data entities, external entities and the relation ships between them. It often includes the depiction of the various “states” of the different entities and the “transformations” required to transition the entities from one state to another.
  1. **Levels Or Categories Of Requirements Analysis**

Debates have regard for some time on who “owns” requirements: the customer or the developers. To deal with this issue we divide requirements analysis into two levels.

* **Customer or C-requirements.**
* **Developer (Detailed) or D-requirements**.

The first level documents the customer’s wants and needs, and is expressed in language clear to him. The results are called customer requirements or C- requirements. The primary audience is the customer community and the second way audience is the developer community. The second level documents the requirements in a specific structured form. These are called developer requirements or D-requirements. The primary audience for D-requirements is the developer community and the secondary audience is the customer community.

* + 1. **Customer Or C – Requirements**

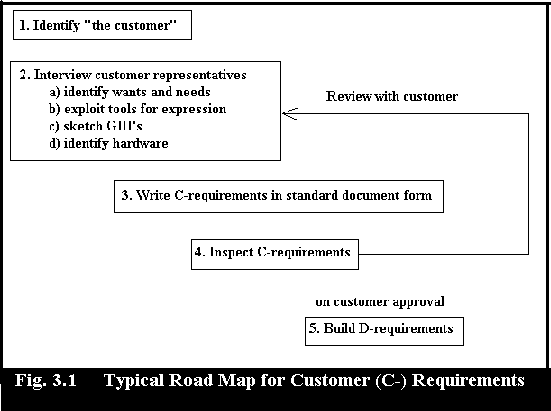
When the development community begins requirements analysis, the customer is typically still forming concepts of what he wants and needs. This is analogous to the requirements gat hering phase between an architect and a client. The analysis of requirements is a person to person activity, care fully organized to produce the best application. The software engineer(analyst) gather the customer or C-requirements by different techniques but usually the interviews are the best way to gather the C-requirements.

**Road Map Of C- Requirements**

The following figure shows the different steps of gathering the C-requirements. The first step is to identify the “Customer or User”, then conduct the interview with the customer which is the main technique used to gather the C-requirements which identifies that what the customer actually wants. After identifying the needs of the customer the analyst is now able to write the C-requirements in a standard document. On that basis the analyst will be further able to build the D-requirements that is the main purpose to write the C-requirements.

There are various ways to express the C-requirements:

* If the requirement is simple and stands alone, express it in clear sentences within an appropriate section of the SRS.
* If the requirement is an interaction between the user and the application, express it via a use case.
* If the requirement involves process elements, each taking inputs and producing outputs, then use the data flow
* If the requirement involves states that are going to change at some particular event then express these types of requirements via STD (state transition diagram)



Use cases are widely applicable for describing customer requirements because they capture user application interactions. If a state – transition diagram expresses what the customer wants and needs, and the customer understands the diagram, then its use is appropriate. The same holds for data flow diagrams. Data flow and state-transition techniques are commonly used for expressing designs. If an application is required to track the flow of orders within a company, then a data flow diagram (DFD) showing this process at a high level would be an appropriate form for C-requirements, because the DFD is needed to express what is to be done.

* + 1. **Detailed or D – Requirements**

Software engineers need a base for design and implementation, this base consists of the detailed requirements these are also called “specific requirements”, “functional requirements”, “developer requirements”, or “D-requirements”. D-requirements consists of a complete list of specific properties and functionality that the application must posses, expressed in final detail. Each of these requirements is numbered, labeled, and tracked through implementation. They are consistent with and elaborate upon the C-requirements. The D-requirements are intended to be ready primarily by developers. Customers are interested in them as well and are typically able to understand and comment on many of them.

**Road Map Of D – Requirements**

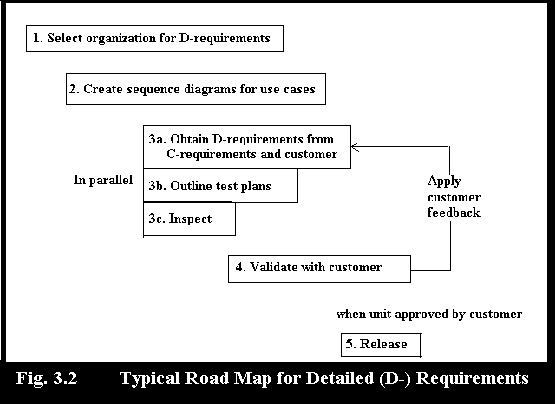
The following figure shows a typical sequence of activities for gathering and documenting D-requirements. There are five steps to obtain the D-requirements. The first step describes the ways in which specific requirements can be organized. Then create the sequence diagrams, after that the third step is started in which the D-requirements are written from the C-requirements. Then we begin writing tests for each of the specific requirements simultaneously with writing the requirements themselves. Although D-requirements are written primarily for developers and their tests are also reviewed with the customer. Then finally the D-requirements should then be inspected and released. Once the D-requirements ha ve been collected, the project documents are updated or reflect the improved project knowledge.

- Organize the specific Requirements.

- Write D Req from C Req.

- Test and inspect for each specific Req.

- Release final Req



The D-Requirements (“developer” or “Detailed” requirements) are written primarily for designers and developers. They are created from C-requirements, as well as from continued customer interaction. The D-requirements must be testable, traceable, and consistent.

* + - 1. **Types Of D – Requirements**

There are several types of requirements. This classification applies to both C- and D- requirements. During the writing of C-requirements, these distinctions are often secondary to getting points across to the customer about the application in general. The classification becomes much more significant when writing the D-requirements, however, because it guides the development and testing process in different ways.

* + - * 1. **Functional Requirements**

Functional requirements specify services that the application must provide. Functional requirements are associated with specific functions, tasks or behaviours of the system that must support these requirements. In terms of the [ISO quality characteristics for evaluation](http://www.issco.unige.ch/projects/ewg96/node55.html#qltiso), the functional requirements address the quality characteristic of functionality.

As you might expect, functional requirements express how the system behaves. These requirements are usually action oriented ("When the user does x, the system will do y.") Most products and applications, conceived to do useful work, are rich with software functional requirements. Software is used to implement the majority of the functionality. When you are defining functional requirements, you should seek a good balance between being too specific in stating a requirement and being too general or too ambiguous. We have found that most functional requirements can be stated in simple declarative statements. Experience has shown us that a one- or two-sentence statement of a requirement is usually the best way to match a user need with a level of specificity that a developer can deal with.

Functional requirements collectively define what a system does, typically in terms of visible changes that you can effect in the system. or that it can cause in the outside world. Functional requirements are typically binary in nature: you either meet a requirement or you don’t.

A functional requirement specifies a function that a system has to perform:

* A functional requirement defines **what** is to be done with the system. not how to implement it.
* A functional requirement defines that the action to be performed as the inputs to generate the outputs (precondition/post condition).

**Non Functional Requirements**

Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless. Define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc. These non functional requirements specify the timing constraints that the application must observe. Customers and developers negotiate constraints on elapsed time for computations, RAM usage, secondary storage usage etc.

Non-Functional Requirements in Software Engineering presents a systematic and pragmatic approach to `bring quality into' software systems. Systems must exhibit software quality attributes, such as accuracy, performance, security and modifiability. However, such non-functional requirements (NFRs) are difficult to address in many projects, even though there are many techniques to meet functional requirements in order to provide desired functionality. This is particularly true since the NFRs for each system typically interact with each other. Before we go into specifics we will state some overall goals and requirements to our infrastructure, these goals are actually the non functional requirements.

**TYPES OF NON FUNCTIONAL REQUIREMENTS**

**Performance Requirements:** performance requirements are a critical part of real time applications in which actions must complete with in specified time limits.

Performance requirements represent the performance of the system is required to exhibit to meet the needs of users.

* What is the acceptable throughput rate?
* What is the acceptable response time?

**Reliability and Availability Requirements:** Reliability requirements specify reliability in quantified terms. This kind of requirement recognizes that applications are unlikely to be perfect and so circumscribes their extent of imperfection. Availability closely related to reliability quantifies the degree to which the application is to be available to its users.

**Usability:** The system will not be used directly by users of NFR but will be more of an API to the rest of NFR. Thus, we focus on making the interface clean and simple, rather than providing services to users.

**Control or Security Requirements:** Control requirements represent the environment in which the system,

* Must access to the system or information be controlled?
* What are the privacy requirements?
* Does the criticality of the data necessitate the need for special handling (backups, offsite storage, etc.) of the data?

Security is an important issue in an open, distributed system. It is also a large field far beyond the scope of our project. These requirements may affect the selection of hardware and operating system, and the design of interfaces and database components.

**Error Handling:** This category of requirements explains how the application must respond to errors in its environment. For example what should the application do if it receives a message from another application which is not in an agreed upon format ? these are not errors generated by the application itself. In some cases error handling refers to actions which the application should take if it finds it self having committed an error because of a defect in its construction.

**Efficiency Requirements:** Efficiency requirements represent the systems ability to produce outputs with minimal waste.

* Are there duplicate steps in the process that must be eliminated?
* Are there ways to reduce waste in the way the system uses it resources?

**Information Requirements:** Information requirements represent the information that is pertinent to theusers in terms of content, timeliness, accuracy, and format.

* What are the necessary inputs and outputs? When must they happen?
* What is the required data to be stored?
* How current must the information be?
* What are the interfaces to external systems?

**Interface Requirements:** These requirements describe the format with which the application communicates with its environment. These requirements define internal or external control and data flows.   
   
An interface is a boundary between two systems. It can be described in terms of what is exchanged across the boundary.

* **Communication Interfaces**.   
  The networks and protocols to be used.
* **Hardware Interfaces**.   
  The computer hardware the software is to execute on.
* **Software-Interfaces**.   
  How the software should be compatible with other software: applications, compilers, operating systems, programming languages, database management systems.
* **User Interfaces**.   
  Style, format, messages, responsiveness.

**Service Requirements:** Service requirements represent needs in order for the system to be reliable, flexible, and expandable.

* Who will use the system and where are they located?
* Will there be different types of users?
* What are the appropriate human factors?
* What training devices and training materials are to be included in the system?
* What are the reliability/availability requirements?
* How should the system be packaged and distributed?
* What documentation is required?

**Economy Requirements: ·**

Economy requirements represent the need for the system to reduce costs or increase profits

* What are the areas of the system where costs must be reduced?
* How much should costs be reduced or profits be increased?
* What is the timetable for development?
* What are the budgetary limits?

**Constraints:** Design or implementation constraints describe limits or conditions on how the application is to be designed or implemented. These requirements are not intended to replace the design process. They merely specify conditions imposed upon the project by the customer, the environment, or other circumstances. Constraint requirements set restrictions on how the user requirements are to be implemented.

**Maintainability Requirements**.

These requirements can only be verified in the operations phase. Maintainability is enhanced by:

* Use of a high-level programming language.
* Minimizing the volume of code.
* Use of tools that allow easy modifications.
* Building in features to allow to locate faults quickly.
* Provision of remote maintenance facilities.
* Assigning parameter values at runtime.

**3.3 Design Constraints**

The third class of requirements design constraints typically imposes limitations on the design of the system or the processes we use to build a system.

*“ Restrictions on the design of a system, or the process by which a system is developed, that do not affect the external behavior of the system but that must be fulfilled to meet technical, business, or contractual obligations.”* For example,

Usually, a requirement allows for more than one design option; a design is a conscious choice among options. Whenever possible, we want to leave that choice to the designers rather than specifying it in the requirements, as they will be in the best position to evaluate the technical and economic merits of each option. Whenever we do not allow a choice to be made, the design has been constrained, and a degree of flexibility and development freedom has been lost.

Design constraints can also be found in the developmental infrastructure immediately surrounding the system to be developed. These usually include

* **Operating environments:** "Write the software in desired language."
* **Compatibility with existing systems:** "The application must run on both our new and old platforms."
* **Application standards:** "The developer must follow some particular standard such as ISO etc."
* **Corporate "best practices" and standards**: "Compatibility with the legacy data base must be maintained."

Another important source of design constraints is the standards under which the project is being developed. Almost all projects will have some design constraints. Generally, the best way to handle them is to follow these guidelines.

* Distinguish them from the other requirements.
* Include all design constraints in a special section of your collected requirements package, or use a special attribute so that they can be readily aggregated. That way, you can easily find them and review them when the factors that influenced them change.
* Identify the source of each design constraint. By doing so, you can use the reference later to question or to revise the requirement.
* Document the rationale for each design constraint. In effect, write a sentence or two explaining why the design constraint was placed in the project. Such as, "Why did we put this constraint in there?".

**3.5**  **Methodologies And Tools Used For Requirements Analysis:**

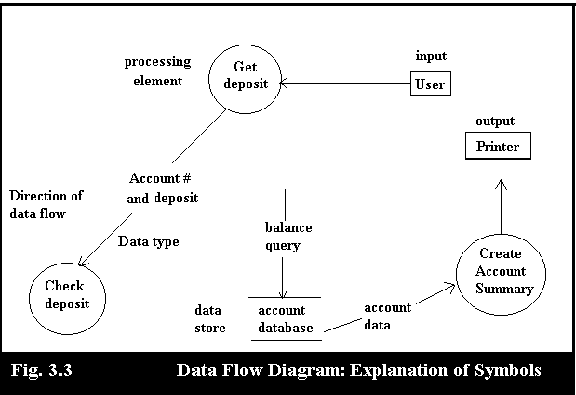
The greater challenge is that how to express clearly what customer want and need. Words alone can be appropriate to express the customer needs, but for many applications these words are not sufficient so for that the narrative text needs to be supplemented by various kinds of tools through which the engineers can clearly express the customer’s concept. Some of the most widely used tools for requirements analysis includes the data-flow diagrams, entity-relationship diagrams, object-class models, state-transition diagrams and use cases.

**3.5.1 DFD ------- Data – Flow Diagrams**

Data flow diagrams are used to study the manner in which information enters a system and the manner in which it is transformed as it flow through the system. Data flow diagrams graphically represents the system using symbols for data source/ destination (also called external entity), process, data flow and data store. Multiple data flow diagrams are used to depict an existing or proposed system the first diagram is also called “Context diagram” or “level-0 diagram” sets the context of the system with respect to the major inflows and outflows from the system. The next level (level-1) of the diagram explores the system into the high level processes and depicts the interactions between these processes. Each process from the level-1 DFD can now be broken down into lower level DFDs.

In data flow diagrams the nodes shown as circles or rectangles represent processing units. The arrows among them denote the flow of data. Data stores are places where data resides such as data bases are denoted by a pair of horizontal lines enclosing the name of the data store. Suppose for example that our customer is trying to explain a kind of banking application that she wants starting with deposits into an account. This deposit function might be getting the deposit from the user, and checking the deposit transition to make sure that it is legitimate. These functions are represented by circles in figure. Next the type of data flowing between these functions is noted on the figure, the account number and the deposit amount. The user is involve too and so should be represented.

The DFD serves two purposes: (1) to provide an indication of how data are transformed as they move through the system and (2) to depict the functions (sub functions) that transform the data flow. The DFD provides the additional information that is used during the analysis of the information domain and serves as a basis for the modeling of the function.

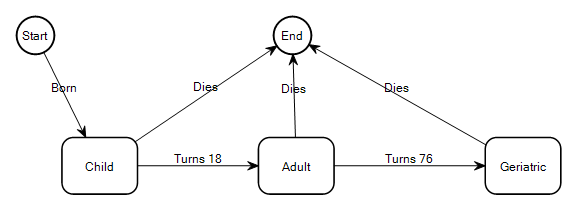


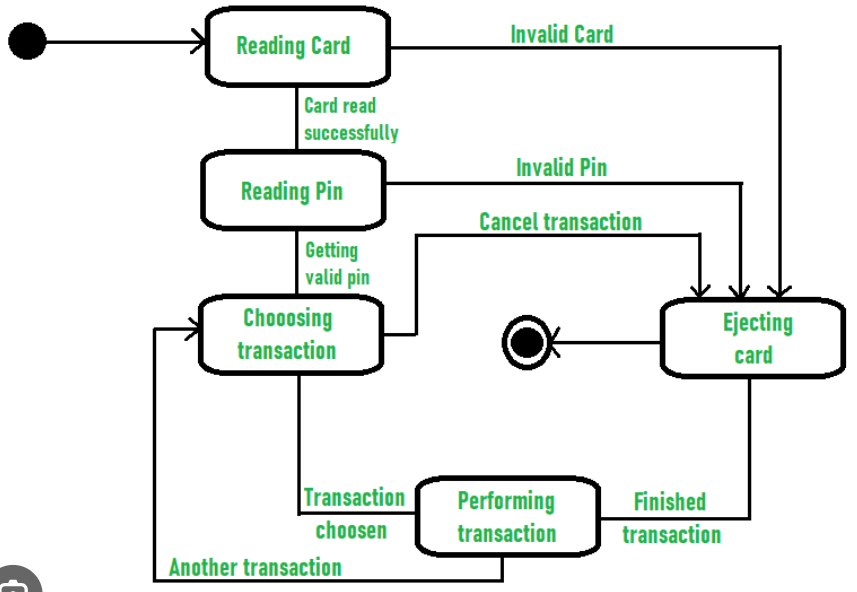
A data flow diagram (DFD) is a graphical representation that depicts information flow and the transform that are applied as data move from input to output. The basic form of data flow diagram is known as a data flow graph or a bubble chart.

**3.5.4 STD -------- State Transition Diagrams**

STD (State transition diagram) is a behavioral model of the system that indicates how the proposed system will behave in response to external events. States are some time called phases or stages. A state transition diagram represents various “states” of the system and the manner in which transition will be made from one state to another on receiving a stimulus. STD is typically used to model real time and process control type of systems. The STD can be examined by the user and the analyst to confirm that the invalid transitions are not required and all other transitions are valid. STD can be used for a whole system or a portion of a system. How ever they are ideal for describing the behavior of a single object. STD help the users, analysts and designer understand the intended behavior of the system, by providing a compact, language independent visual representation on certain aspects of the system. STD help in ensuring that all the required states and permitted transitions are understood and documented and that the system does not permit unnecessary transitions. In large complex systems it is near to impossible to depict all possible states in a single diagram and verify the correctness and completeness. In such cases the state transition diagram may be structured into high and low level. STD do not show processing that is required it shows only the states and events to the extent that these events cause a change in the state of the system. The STD does not provide any processing logic for the developers to build software and for this reason just STD are not an adequate technique to model requirements. STD should typically be used to complement other techniques.

STD indicates how the system behaves as a consequence of external events. To accomplish this the STD represents the various modes of behavior called states of the system and the manner in which transitions are made from state to state. State transition model is a good way to explain the concept of operations. State transition models are commonly used as design tools as well.





**3.5.5 Use - Cases**

Since early days of requirements elicitation and analysis, analysts have been using “scenarios” to describe to the user how the user will interact with the system. An object oriented method this is called the “use case” approach. Though we discuss in the context of requirements analysis, it is also a very powerful tool during requirements elicitation, documentation and review.

A use case is a description of the interactions between the “system” and an “actor” external to the system. The actor could be a person, a group of persons, or another system that interacts with the proposed system. The use case describes all the steps that the actor and the proposed system ill need to perform to achieve a desired objective. Use case diagrams provide a high level visual representation of the user requirements. Once the use cases and the actors are identified, the following questions may be asked by the analyst to gather details regarding the use case:

* what is the actor trying to accomplish here?
* What information will the actor have to provide the system?
* What information will the system have to provide the actor?
* In what sequence will the information exchange take place?

Requirements are often naturally expressed as an interaction between the application and an agency external to it, such as the user, the use case, is a very useful way to express customer requirements in the form of such interactions. A use case is identified first by its name, and by the type of the user of the application called the actor. It consists of a typical interaction between an actor and the application.

In UML notation, an ellipse denotes a use case and the agency which is external to it is denoted by an actor. Use cases are useful for specifying requirements, design and test cases. Since high level use cases can express the customer’s vision of how the application is to work that all will be include in the section of software requirement specification (SRS).

As requirements are gathered by several informal meetings or applying the gathering techniques after that the software engineer (analyst) 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. Define somewhat more formally, an actor is anything that communicates with the system or product and that is external to the system itself.

It is important to note that an actor and a user are not the same thing. A typical user may play a number of different roles when using a system, whereas an actor represents a class of external entities (often, but not always, people) that play just one role. Once actor has been identified, use cases can be developed. The use cases describe the manner in which an actor interacts with the system. In general a use case is simply a written narrative that describes the role of an actor as interaction with the system. Use case provides an unambiguous scenario of interaction between an actor and the software, that scenario will be perceived differently by different actors.

* 1. **Summary**

The purpose of the requirements analysis is to establish an understanding of the application domain and to capture, formalize, analyze and validate the user requirements on the system to be built. Once requirements have been gathered (or elicited) the requirements analysis phase is started. The output of requirements analysis is a document generally referred to as a requirement specification of software requirement specification (SRS).

The requirements analysis are divided into two levels. ***(1)Customer or C-requirements.***

***(2) Developer (Detailed) or D-requirements***. The first level documents the customer’s wants and needs, and is expressed in language clear to him. The results are some times called customer requirements or C- requirements. The second level documents the requirements in a specific structured form. These are called developer requirements or D-requirements. There are three main types of D-requirements (1) Functional Requirements (2) Non-Functional requirements and (3) Design Constraints.

There are some analysis principles, by applying these principles, the analyst approaches a problem systematically. The ***information domain*** is examined so that function may be understood more completely. ***Models*** are used so that the characteristics of function and behavior can be communicated In a compact fashion. ***Partitioning*** is applied to reduce complexity. ***Essential and implementation views*** of the software are necessary to accommodate the logical constraints imposed by processing requirements and the physical constraints imposed by other system elements.

The greater challenge is that how to express clearly what customer want and need. Words alone can be appropriate to express the customer needs, but for many applications these words are not sufficient. So for that the engineers used tools for requirements analysis includes the data-flow diagrams, entity-relationship diagrams, object-class models, state-transition diagrams and use cases.