CORE PROBLEMS IN SOFTWARE REQUIREMENT

A Practical Approach to deal with

Safi Khan, MS in Computer Sciences, Virtual University of Pakistan

1 - Introduction:

In the software development life cycle phase [1], the requirement engineering play a significant role to establish the goal, what to build? How to build? Why to build? In large or complex software systems the developers have mainly to face the problem of requirement in whole software engineering process life cycles. How every, there can be different technologies to solve a particular problem, and there's may be considerable disagreements. There are also some of the disagreements with Brooks' assessment: "that no other part of a development is as difficult to do well or as disastrous in result when done poorly". The purpose of this paper is to convince the readers to deciding what to build can be so difficult, and some time it is really difficult to address these kinds of technical difficulties, and how to improve to achieve the better result in near future.

This paper provide the material to understanding the problems facing during the requirement engineering process [2], however there are many approaches to overcome these problems. These different views highlight the diversity in issues of different kind of software project, and the different assumptions and solutions about their desirable characteristics. This paper provides reasonable solutions to solve these problems in requirements [3] as well as tradeoff involved to develop solution. However, the reader should make his own assessments to deal these problems in differ requirements methods with their desirable approaches.

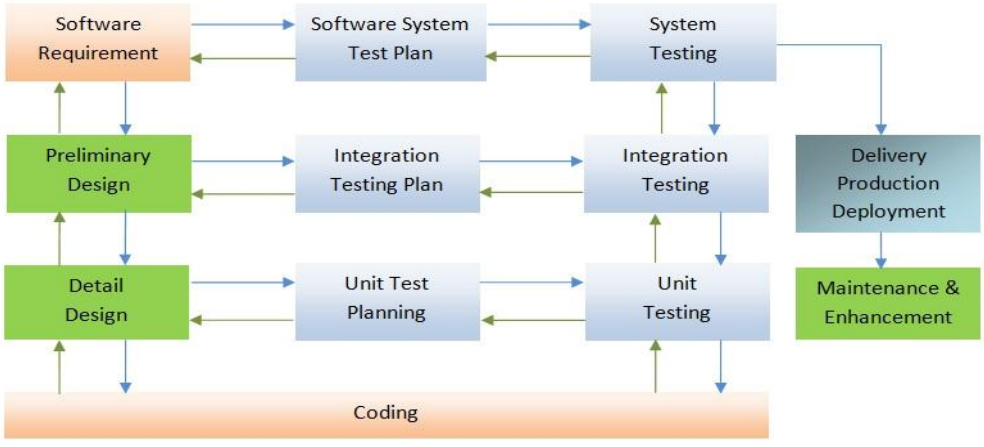

Figure 1: Software Development Life Cycle

We begin with the historical data and very basic terminologies on the requirement engineering problems. We also examine those problems which we are currently facing in requirement engineering phase to achieve the goals. In the Brooks's article [Brooks 87] [4], a lot of discussion is to motivate to investigate, what is required? ("Essential", difficulties) And what is achieved? ("Accidental" difficulties). We will examine some of the discipline approaches which are beneficial to tackle the accidental difficulties in software engineering process. Our focus will be on the written detail technical specification of the requirements with these disciplined approaches. For this we examine the current approaches to address these requirement problems, and finally we examine what are the significant advancement can be occur in the near future which leads to the technical trend.

2 - Requirements & SDLC:

A lot of Software Development Life Cycle (SDLC) [1] has been published many of these uses same terminology. These models have some core elements which take care to manage risk and developing a prototyping model. A general agreement of these models makes comparison in detail decomposition of steps and between control structure and surrounding management. Figure 2 show the relationship between stages of the software development life cycle with different colors which shows the design, testing, and coding and maintenance stages.

For a very large project which contains larger hardware and software system the requirements are written first followed by system design [5] which included the detail description of resources used by the hardware and software. Software Development Life Cycle begins with the requirements analysis which has the equal importance for large and small projects, so in our further discussion we do not take care whether the software system is large or small as shown in figure 2.

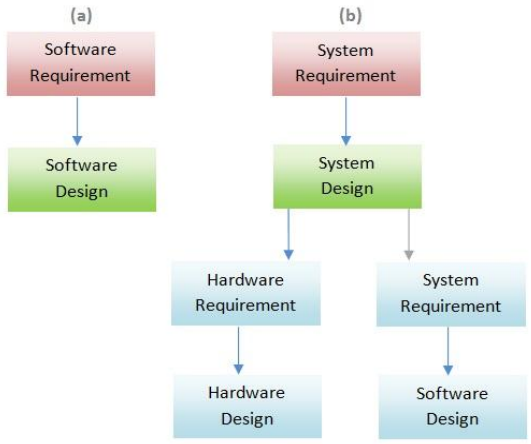

Figure 2: Development Path (a) Software (b) System

The specification of software requirement system has many different roles in the development of a large system. [6]

- For a customer prospective, the requirement document is necessary to delivered which are providing the basic development scheme.
- It provides the scheduling for the manager which they use to measure the progress of day to day work.
- For the designers it provides the specification, how to design? And what to design? Etc.
- For the coders the requirement document defines the final output as well as the final implementation of the functionalities that leads to acceptable state.
- It is the basis for testing verification and validation for the personnel of quality assurance.

However, the software requirement may come from the governmental regulators or marketing diverse groups. So far the requirements can be divided into functional and nonfunctional. Functional requirements force to a solution

which takes an input and produce the correct output on the basis of resource provided. Nonfunctional requirements [7] lead to some constraints like safety, reusability, maintenance and dependability. However these constraints are not limited.

These functional and non-functional requirements [8], somehow has the terminologies which are confusing to each other but at the same time it identify some of the common properties which may be different in nature. The benefit of classification of the requirements as functional and nonfunctional also helps to priorities classes which mark the similarities in the quality attributes and also provide the batter understanding the nature of classes. While grouping in common classes, via functional and nonfunctional requirements also helpful in output deadlines and maintainability goals.

Now we focus on the "behavioral requirements" [9] and "developmental quality attributes" [10], which make distinction between that what can be described and what cannot.

- Behavioral requirements lead to acceptable implementation state that incorporates all of the meaningful information at runtime. It can be include all of the constraints like performance, security, timing, accuracy and fault tolerance which impact on the system output as whole.
- Developmental quality attributes define the constraints which refer to static construction of the system. These constraints include the testability, maintainability, reusability as well as changeability.

Behavioral requirements have runtime properties, i.e. behavior of the system at running state, are common in nature, observe the behavior of the system which is actually independent of its implementation methods. While developmental quality attributes leads to representation, i.e. static structure of the system. However when we talk about the development process functions and construction methods we refers to the developmental quality attribute which are more relativistic, i.e. we cannot justify about a design that it can be maintainable or not, or one design is more maintainable than another.

3 - A Big Problem:

Dealing with problems in the requirement engineering is costly as the projects are being large. These problems can be persistent and can be pervasive on stat to another. These problems have the major source in requirements as the software become significant part of the system. Most of the software systems fail due to problems in the requirement engineering. Mostly, the backbone of the project failure is due to poor requirement [11, 12] which lead to different kind of problems at the latter stages in the software development life cycle.

Studies show that at the latter stage the cost of resolving these problems are very costly once it trigger in requirement stage. If these requirements problems are detect at the design or maintenance stage then off course the relative cost increase very quickly. For example as shown in figure 3, if the cost to solve a particular problem at the requirement stage is 100 RS, then this cost would be increase 1000 RS to 2000 RS at the maintenance stage or until the system fails. Cost of such failures can be increase unexpectedly. [14]

As the advancement in the software engineering methodology and tools [15], the problems in requirement cannot be vanished.

This does not mean that the progress in software engineering processes is illusory. As application project being complex significantly, these problems remain unchanged. In conclusion as the project growing ambitiousness, the software systems have outpaced the gains in requirements technology.

Stage	Relative Repair Cost
Requirements	100 Rs
Design	200 Rs to 300 Rs
Coding	400 Rs to 500 Rs
Unit Test	600 Rs to 700 Rs
System test	800 Rs to 900 Rs
Maintenance	1000 Rs to 2000 Rs

Figure 3: Relative cost to repair a software error in different stages

4 - Why is Requirements Hard?

In the requirement engineering the basic concept is the specification of what to do instead off how to do. However this is looks like so simply but really hard to implement in practices. Why it is too hard? [13] To answer this question we should have clear vision of the project goal, need to identify the target area where the errors originates and need to understand why and how the difficulties inherent.

Software engineers agreed on that the requirements should be specify "what" rather than "how". This mean that requirements should specify the problem solving rather than its solution, for example MCQ's Test software it is considered the student id's, total marks, time allowed etc., rather than an algorithm of data structure that is used in the software. The basic concept is that the requirements should be more problem solving specific rather than it is a constraint of the pervious (if any) or subsequent design or implementation. However the software solution is more complex to design, implement and change, off course it is almost very hard to understand by the customer than defining or specifying the problem.

"What" and "how" both are representing a state which is distinguish between then as a function of perspective. In other words, while chosen a function is further a decomposition of system [16] level to define "what" is on next state. The "what" is defining by the customer can further divided in term of software and hardware to define "how" in the next state. "What" defining in the behavioral requirement lead to "how" in the next stage of software design. So in short software requirement cannot be define without these levels of decomposition which should adjust before to meet the agreement the system. So we as a requirement engineer should have the clear view about problem and solution space both.

While talking about the problems in the requirements engineering it should be distinguish about the complex and a large software system with less effort where a small team or a single person works. A very large system are developed by the large number of persons may be thousands of peoples. They work as a team and a team for software engineering has also many departments like software engineer, quality assurance personals, accounting personals, market personals as well as management personals. However some of the outside context like different regulatory agencies, market suppliers and different software products also involved.

In the large system there may be multi-version [17] which are using by the different stakeholders. When the software is being developed, using by the customers/stakeholders, it evolves extensively. So the customers as well as the developers of the software understand it better way to how to use and developed it respectively. The

customer knows in better way what they needed and what they want, in the same way the developers knows in better way what to develop and how to develop. When software is used the result are need changes, i.e. taking new requirements, i.e. changing. And as a result the several changes are occurring which are affecting the improvement plans. In that way several versions of software are developed.

When these multi-versioning are taking place in a software product both the quantitative and qualitative problems are occurring in the large scale software system in oppose of small scale software systems. For example, in large scale software project management of large stakeholder is a problem with respect to small scale software project. In small scale software project these kinds of problems are do not exist. One of the quantitative effects is the communication which are being complex in case of large number of workers. In the large projects the problems occur in the following sections, so we as a project developer should take care of these following areas.

When we talk about the multi-version project we establish the goal what the software do:

1. Clearly understanding what is required by the software project.
2. All the stakeholders are involved in this communication.
3. Provide the controlling mechanism for changes and also guaranty that the final project meets the requirements completely.

However the requirements error is generated by one of the following reasons:

1. The developers are failed completely or partially to understand what are required by the customers, end users and all other parties who needed the final product.
2. The developers are failed to capture the requirements completely or partially and also fail some time to communicate the requirements.
3. Sometime the developer failed to manage changes effects to ensure the down-stream conformance, which includes coding, designing, maintenance or integration testing etc.

At the end such failures lead to problems identifying or unacceptable state of the project, sometime project budget and schedule affecting too frequently and some time it leads to failure of project delivered.

4.1 - Essential Difficulties:

While developing the software project when we saw our goals they are look like state forward so why we fail to achieve them? The answer of this question is that some of these goals are inherently difficult because of inherently difficult. Brooks make some distinctions between the essential difficulties [19] to address that why the requirements so hard and how those requirements problems are inherent. He also rectifies problems those are occur through imperfect practice known as accidental difficulties. However the requirement is difficult inherently due to inadequate or imperfect practices. We classified essential difficulties are as follow.

Comprehension:

Actually, the people in some sense do not know what they want. However, it is not that they do not know about the software components or behavior. In fact there are some variation in their ideas and system which are providing them. They do not know the detailed understanding of the software system functionality, its output, how the operations take place and how much long these functionality support. The new system implementation is simple sometime because it can be created to extents the functionality of

the old system. Many problems rectify the behavior of the system depended upon the decisions which are not made yet. When the problems are identifying the raising exceptions are changing exhaustively. So there is need to understand the expected behavior to development of effective design and correct coding.

Communication:

Difficulties arise in communication also while discussing software requirements. However, the conceptual structure [19] is difficult to understand due to its complexity. In large project system we almost use the complex structure. However these complex structures are predefine on the decisions of the peoples or prior construction. These conceptual structures are not easy to visualize because they have no readily comprehensible physical analogue. The regular and predictable structures are the best which can comprehend the limited information at one time, and can get the large amount of information after visualizing. So it leads to inherently difficulties.

In communication the inherent difficulty [20] is due to diversity purposes and due to audiences of requirements specification [21]. The technical specification document [22] is written for some particular audience. The document comprehensibility is depends upon the technical background assumptions and uses of the language. However for many diverse groups these commonalities are do not hold such as managers, system engineers and customers etc.

Control:

The inherent difficulties [20] are also found in software development. It is difficult to visualize which requirements are fulfill the customer need or demand easily and which are harming the budget and schedule of the software project. Requirements should be written in that they are best suitable for planning and for tracking, but the low fidelity become cliché some time. This situation becomes worse when the software is becoming large and large. The best suitable planning for controlling the software development processes [23] should adopted because sometime difficulties to approach these problems frequently and arbitrary changes the requirements. One these changes are at continuous level then the development of stable specifications is not easy, so we should plan effectively to deal with the scheduling time and for the purpose of controlling cost.

Inseparable concern:

While solving these problems the some additional problems are involved and deal with those problems are not easy at all. For example, the development team solves these problems on the basis of the baseline and freezing requirements before design begins. Due to problems of comprehension this is not possible because at the same time the customer do not know what he want until he or she sees. And there are many stakeholders who are from different backgrounds and have different needs in case of requirements, so dealing them by writing different requirements for each are not easy. So it leads to different system specification, a different set of requirements deliver to different set of stakeholders. So this lead to software developer's internal consumption and complexity of the software also increase and manage changing are not easy.

These issues some time become massive because of the interlocking nature of the conceptual structure of the software. Different stakeholder involve in different requirements lead to play different roles. Inherent dependencies between different facts impose constraint on any potential solution.

The implications are twofold. Firstly, dealing with the complex problems we have constraint in the application while adopting the effective strategies, i.e. divide and conquer. In case of isolation in problem the solution sometime become worst and generating more problems. The effective solution in that case must address more the one problem simultaneously. Secondly, there are different tradeoff involve in the solutions, while these solutions have different constraints, compromises etc. In case of compromises, there are different stakeholder are involve generating different goals which leads to different gains and losses, so dealing with require some negotiations.

4.2 - Accidental Difficulties:

Requirements are inherently difficult and the common practice increases these difficulties unnecessarily. We refer the word accidental here in the reverse of the essential. So the accidental difficulties are produce to bad practice in of requirement elicitation, requirement specification and the requirement management stages. There are some accidental difficulties as follows:

Written as an afterthought:

In common, the requirements document is written after the software development. For many projects, requirement specification after implementation proves irresistible. The developers do not do anything before they write code and manager do nothing without scheduling. So the product's intangible nature reduces the intensity to earlier implementation. Documentation makes the actual behavior of the product visible i.e. understandable in batter way that what is needed and how it can developed. So documentation written as afterthought leads the developer and tester to wrong predictions. This the violation of software engineering processes and it suffers the keywords "what" and "how". Documentation produces after development has no plan nor well manages which is the essential part of the development. So as the result there are not guide implementation plan for development and management.

Confused in purpose:

The main requirements become confusing some time because there are a lot of stakeholders suggesting their different ideas, confusing the goal in main requirement document by representing the different view point. An earlier version includes the product virtues of market hyper extolling to sell the product to the customer. The documentation is used to provide the overview as explanatory purpose of the product introductory. The requirement document works as contract for the latitude of the developer and customer as delivered product and making no cost change respectively. The requirement document works as a vehicle of communication for the designer as well as coder, so it helps in incorporate the design and implementation. So the unclear statements are the result, which leads both the real requirement and market requirements as a mixture. So that requirement document serving all the stakeholders in different manners.

Not designed to be useful:

Some time it rushes to implement the requirement with little effort in a rush and self-prophecy so the result is unexpected and no useful requirements. Sometime the requirement as unexpected and not useful so it creates the problem while writing, checking, managing, designing, creation and evaluation, so the overall result is the poor organization. Sometime these specifications are in English prose so reading these requirements needs consciousness in order to execution. So the resulting document sometime becomes non

effective to their technical reference. So it is become unclear to represent the actual requirement and sometime also difficult to manage it and find it in a proper way. As there are no systematic way for the requirement change management, so its specification is difficult to maintain and use, and sometime it is become out of date quickly and its usefulness is lost quickly.

Lacks essential properties:

Sometime the requirement document miss its essential properties like confusion in purpose, lake of forethought and lake in careful design as well as it execution. These all flaws sometime incorporate in the requirement document lead to failures. If the requirement document is lead to that stage in earlier in the documentation phase, it leads to inaccurate design, impreciseness, incompleteness, inconsistencies, as well as includes redundancy.

As the essential difficulties suffer as inherent problems the accidental difficulties lead the project failure, and lead to failure to gain or maintain and control the project application. The essential difficulties lead to no "silver bullet" which sudden overcomes the problems. However the accidental difficulties can be overcome by thought out the problem solution, by using a systematic approach and by introducing the discipline in development process. So these discipline processes helps to attack on the essential difficulties for stable foundation.

5 - Role of a Disciplined Approach:

Accidental difficulties [24] can be overcome by using the disciplined application which analyzing and specifying the software requirement. There are insufficient software development processes yet, due to non-mature standard. However there are general agreements on required qualities of software. In order to completely understand the development process needs to understand its characteristics and its product deeply, and examine its week aspects. In practice, requirement approaches define completely as:

- Process: This is activity sequence in a partially order, describing the activities with entrance and exit criteria, work produced by each activity, and mentioning the kind of people worked.
- Product: The produced work product, providing resources to product, providing information, expected audience which uses the product as well as it acceptance criteria.

However different decomposition suggested by the different author with little similarities in the terminology, the Davis provided variation summary. The concept of requirement consists of two phases.

1. Problem analysis: Precisely understating what is the problem to be solve including the identification the actual system, it constraints, acceptance solutions, who is used it and also tradeoff between conflicting requirement constraints.
2. Requirements specification: Documentation of specific requirements known as (SRS) describing what to build, problem analysis and also it acceptance characteristic includes in SRS.

Conceptual theory involve between these activities to distinction. The developers are switched between these two activities until the problem is understood very well. This documentation helps to analysis the problem very well based on these standardization and specifications.

5.1 - Problem Analysis:

It is an informal process which sense structure by acquiring its information, understating complexity, relations and all the investigating parts which are under the current development process of developing software and its various parts. This needs to valuable domain knowledge for effective understanding the complexity. However the basis issues in that analysis phase are: [26]

- Election of requirements effectively and completely form different stakeholders and sources
- Decomposition of requirement problems in different phases.
- Information organization in understandable manners.
- Communication of problem with different stakeholders
- Mechanism to resolve the confiscation between different stakeholder requirements.
- Stopping mechanism

5.2 - Requirements Specification:

To facilitate the good development [27], the effectiveness of SRS depends upon its usability and good result of analysis. These benefit of usability decreases while taking no care about customer, product designer, product implementers, tester and main when the requirement are elicited. When the project becomes more complex and large, the above participant should more take care. SRS plays many roles in multi-person and multi-version development [27].

1. The general agreement between the customer and developer are capture in SRS document. It is a contractual license which plays significant role to understand what to build.
2. All the problems analysis and document in SRS document confirming its completeness. However the additional analysis is needed to capture the missing information.
3. SRS document provide the functionality of the system as well as its constraints, somewhere indicating the design freedom. SRS ensure that the requirements are gathering during in the requirement phase rather than in programming phase.
4. SRS also estimate about budget of the project. Primary tools can help for tracking this information.
5. SRS is also known as a test plan tool which helps to lead the project to the acceptance state.
6. SRS document provide the standard behavior which helps maintainer of the project to maintain and record the changes.

SRS is the primary document which controls the technical specification for disciplined software approach. Multi person development team needed to develop the SRS document in large projects to deal with complexity. For development of the accurate system, first should be understand the problem exhaustively and should record all the result which results of comprehensive understanding. In that way the overall picture of the project developed and specification can be defined which one can communicate effectively analysis the results.

A disciplined approach should be needed to address the accidental difficulties, which one should be carefully analysis. Requirement should be completely understand before the development process begins. SRS is the primary document which communicates requirements with other stakeholders so the document should be design carefully then it should maintain.

6 - Requirement for the software:

A good software requirement document should have the following characteristics [28]:

- Complete: SRS document are specifying acceptance criteria so one it should be complete stating all the relevant information according to the customer needs. The information which is not includes before the design stage should also identify.
- Implementation Independent: SRS document should not concern about the implementation so it should be implementation independent for design and its implementation detail.
- Unambiguous and Consistent: SRS document should be conflict free because if the conflicting arises on any requirement, the diverse customers do not agree on it whether the right product being developed. So each requirement should have single interpretation.
- Precise: SRS document specifying the acceptance behavior so one it should specify the range of acceptance values for each of the input. So minimum and maximum acceptance delay should be considered.
- Verifiable: Requirements those are checked against the ambiguity and providing the implementation satisfaction are considered as verifiable. However, behavioral requirements can be verified against any given test case whether it fulfill or not the acceptance state.

The packaging properties of a good specification for a SRS document as given blow:

- Modifiable: That issues related to easy of change. Since different organizations have no easy of change mechanism so analysis process must identify the easy of change for all possible manners and one it should predict all possible changing area and their relevant likelihood of occurrences. So one it should be organized the specification in such a way that it should limit the any extra modification.
- Readable: The parties i.e. diverse stakeholders should understand the SRS document, the problems and goals should describe clearly relate to the elements of problems which can easily understand by all the parties and reflecting the observable behavior of the software.
- Review and Referencing: SRS is a primary document which is representing the requirements specification. It is a repository that store all the decision made during the specification phase that what should be build and what should not. It is a primary origin of dispute among the diverse stakeholders. So the document should organized in such a way that it should referencing easily, quickly and clearly to make a decision belong to any relevant stage.

To overcome the difficulties that are associated with writing requirements must providing the specific technique that supports both the packaging and semantic properties. A conceptually well-defined structure support both semantic as well as packaging properties. However the conceptually both the structure and packaging properties treat distinctly allowing changing in presentation of SRS document without change the meaning.

However, to development of an ideal SRS document is very difficult task especially in a short duration of time. This becomes unachievable sometime due to that reason. Sometime customers do not know what they want or still they are changing requirements so the complete requirements gathering in not possible before starting the design phase. So

the conflicting solutions are arises due to that different requirements. In example, English language prose that are representing the requirements which may be ambiguous due to that language however the formal languages are unambiguous but they cannot understand easily from different stakeholders.

So the ideal SRS [29] become unachievable but it is important to document an SRS because

- It provide the basic to the product and processes goal to an organization
- Progress measurement is achievable
- Help to understand to developer what to build when to build and when to finish that phase.

So in conclusion, understanding of requirement full before the system design is difficult and the perfect SRS document is a very difficult task. However there are many adoptive approaches in literature but do not produce the definitive SRS document. So some time the developer's directly develop the design phase or make some prototype, sometime this work well but this is inconsistent with the software engineering principles. For controlling the engineering processes technical specification are very important but these are not providing the perfection before anything is done. However the development of SRS document is representing the basic goals of whole processes. Sometime there may be lack of understanding of the complete environment and writing skills but it cannot change the facts.

7 - State of Practice:

A large number of techniques have been developed. Do deal with complexity and ease of change the software engineering technique adopted as coding problems firstly and then deal other problems in life cycle. For structure programming [30] and OO programming concepts [31], structured and OO design and analysis techniques [32] are used respectively. In this section we characterized these techniques with the regard of their general strengths and weaknesses relative to the requirements, specification, quality and analysis.

7.1 - Functional Decomposition:

Abstraction [33] of coding details is achieved by the functional decomposition [34] focuses on the understanding and specification of the processes. These specifications ensured the requirement behavior by mapping the inputs to outputs. This analysis processes mapping from top to down. Identifications are performs first for associated functions as a whole. Subsequent functions are subdivided into small group of functions generating the hierarchy of functions [46] which are also providing the functional interface [47]. Each hierarchy provided sufficient detail about the processing steps which are also control the processing of the sub functions. "Call" structure used for functional hierarchy for the implementation. In decomposition the specification is implemented as stated by the language and concept of the implementers.

However functional specifications are difficult to communicate, use, change and for design decisions because these specifications are written by the language of implementation [36], so these are difficult to understand by the other stakeholders. As there may different ways can be adopted for functional decomposition that are not the actual part of the requirements. The processes need to understand fully and comprehensively detail should be provided with the scenario that what has been done in the previous stage. So the functional decomposition is based on the components which are closely coupled. When changing is need to perform on any function, it needs to completely understand

the whole senior i.e. all the related functions. As the complexity increasing within the software, the weaknesses of functional decomposition need to address more concisely for better understanding of non-technical personals.

7.2 - Structured Analysis:

Structure analysis can be handling the accidental difficulties attending problem analysis to less extent, requirement specifications by using functional decomposition. Here we introduce how the structure programming reduces the complexity by gaining the intellectual control using the functional decomposition.

- A conceptual model describing for all the problems commonly.
- General direction and its order can be suggested by the set of procedures.
- Some heuristics or set of guidelines should be supported the problem specification.
- Product quality can be evaluated by defining the some criteria.

In the structure analysis we decompose the function into sub functions and our focus transfer form process step to data process. Here data transformation is analysis by the testers, it source and destination specifying the data storage, in some form data dictionary [37]. Data flow diagrams [38] are helpful in that case where arcs represent the data movement and the data transformation is represented by nodes. Data definitions and descriptions stored in the repository facilitate the data flow in the dictionary to done the processing in form of transformations. Incoming data is transform on the processing node before the output. Textual specification called “MiniSpec” is used at the detail level including like English prose. Decision tables and procedure definition language (PDL) [39] are also used.

The structure analysis approach is basically developed for the management information system (MIS). Some modern approaches are also used in structure analysis for dataflow. Control behavior specifications can be capture in embedded control system by using structure analysis. Functional decomposition can be extends by the structured analysis which a systematic approach to problem analysis, decomposing it into parts and defining the relation between its parts. A well define approach in structure analysis address the accidental difficulties. A conceptual structure which is a graphical representation of specification [41] can address the problems like in communication and comprehension which are based on assumptions that data decomposition handles clearly and less change to be performed on the functions.

Structure analysis technique evolving continuously and widely used, and also under common criticisms. It provides insufficient guidance while in problem analysis and confusion in between data modeling [42] and data transformations [43] and somewhere in aggregation. Some time it arise confusion when to stop decomposition. Some time it needs to examine the intermediate values between the data flow and data transformations which are actually not required.

In that way the week constraints imposed on the conceptual model [44]. However there was goal of developers to create a general approach for system modeling that can best fit for hardware and software applications as well as human enterprises. But unfortunately, these kinds of generality can achieved abstracting any semantics that are not common for all these types in which potential system being modeled. For a particular system, the conceptual model provides less guidance, and provides no differences when apply on requirement analysis

and design analysis. So it results week syntactic criteria for quality access to specification of structured analysis. For example, in data flow diagram, consistency and completeness transformation determination is limited at each level.

However it is possible to develop the dataflow specification that help in easy to understand, effectively communicates with other stakeholders and one can capture requirements behavior correctly. So structure analysis is used by many of developers. The weakness of the conceptual model addressed the specification quality which heavily depend expertise experiences. They should provide as many as necessary disciplines for that unconstrained model.

Finally, the structure analysis provides less support for SRS meet for quality criteria. Mathematical expressions are also unsuitable for representing their relations or detail specification with the dataflow diagrams in case of value, time and interval. For that purpose the detail specification given here with help of pseudo-code or English prone. So these constraints provide less support for an SRS document which one could independently implement, precise, consistent, complete and verifiable. Some of the packaging goals such as reusability review and ease of reference cannot be achieved by the data flow diagrams and attendant dictionaries. So no published methods can guide for process step and structure for producing an SRS.

7.3 - Operational Specification:

Two requirement dilemmas are addressed in the operational approach. First is that, we do not know that what we are building, until the design process is complete. Second is the problem of inheritance move from the requirement specification (what to build) toward the design specifications (how to build). The requirement specification are however closer to the design that are easier to transition but it is more difficult to make a design decisions that are made before proper time. The operational specification approach developed the executable requirements specifications to deal with these kinds of problems. The key elements of the operational specifications are the formal specification language [45] and an engine that can execute that well written in formal specification language. The properties of that formal specification may be analyzing by the automated support provided in the operational approach. The operational specification has two following approaches:

1. It make the requirement specification executable without developing a system to guide or lead us the phenomenon that “what to build”. It validate the customer needs by capturing the accurate system specifications with the less labor cost.
2. It allowing the developers to more specific abstract design decisions by simplifying design procedure as well as its implementation.

There is limited access achieved by operational specification for general applications due to failure to achieve the semantic differences between the computational model and conventional programming. The benefit of this approach is that it predicate operational model [48] to the problem domain without any conceptual structures that are belong to the solution domain. Earlier simulation can be achieved by the operational specification which is actually satisfying the properties like consistent, precise, unambiguous and verifiable. Completeness can also be achieved by via specification to simulation. However these properties cannot be suitable with implementation independently.

7.4 - Object Oriented Analysis (OOA):

Two significant sources are involved in the contribution to current view of OOA namely OO design and information modeling [49] both addressing different set of concepts. This technique is different than the structure analysis where the functions are decompose in the sub functions and then try to relate each other with different type of relationships. In the OOA one it could be decompose the problem in different sets of interacting objects which are on the basis of relationship and the entities. Related set of data can be encapsulated in the objects, and its process and state also. Thus there is a signification difference that in OOA both the data and its related processing are store together than the structure analysis. Objects provide functions in case of services or methods and can be hide their internal detail, structure or data. Since, they provide processing, services and data to the services that are invoked by the object interfaces. Objects are dynamically behaves by the message passing. Static relations are capture in the aggregation [50] and by classification in the problem domain by the analyst. They capture the aggregation on whole/part and capture the classification on class/instance (known as "is-a" or inheritance relationship). OOA structural components are objects, classes; services and aggregation follow some of analytic principle which is based on requirement problems.

Essential difficulties like control, communication and comprehension can be handled by the OOA. Requirements and design [51] can be distinguished in case of what and how respectively, by adopting the principle of problem domain. The problem domain specifies the model relation between entities so one it can completely understand by the stakeholders and other domain experts that ensure the comprehension of the requirements. Information hiding principles [52] and abstraction supports the control and communication problems by abstracting the detail and hide the irrelevant detail which leads to improving readability and reduce the complexity. So the objects provide some time the reusable pattern for function and data.

Key principles of a particular method decrease the potential benefits of OO analysis [54]. Essential aspects of an application can be modeled by defining the relationship between objects in application domain. This technique is not providing in any corresponding conceptual model of the same domain. While in structure analysis there are generic type modeling techniques rather than application type like OOA. While developing the decomposition for a specific object this phenomena leads to insufficient guidance. Such as in the structure analysis i.e. the developer bears the difficulties to choose appropriate data flow and transformation. Same in the OOA there are difficulty to choose the appropriate object and its relations.

One can develop the system based on it physical structure [55] i.e. structured the system and specifying its requirements on basis of real world models which are usually stable. But at the same time it is not necessary that all the real world things have a physical analog and likely to change arbitrary. However basic requirement structure on the physical structure is a valuable heuristic and it can develop the requirement specifications that are complete and consistent.

Another problem is that the semantics of the OOA methodology typically based on the software's conceptual structure, not on the problem domain which is actual need of the analysis. So the analysts bear the overhead to take care of object language properties and features rather than problem domain

properties. The simple example is networking where one object need to pass a message to other object while containing the information defined in another.

Another serious problem is the most OOA methodologies leads inadequately to develop a good SRS documents. Many of object oriented technique present provided in the literature are represent the specification mechanism informally so there are not way to check the SRS document against its completeness, consistency and verifiability. None of the technique interpreted which are directly address the problems that are helpful to develop the SRS as a reference document. The emphasis of an OOA technique is to address the analysis not the specification. A well written SRS ensure all the principles that are applied to understand and modeling the problem when the result are written on the same document, and time otherwise. For a particular object's specification the tradeoff involve inherently which requires a disciple approach for SRS not for problem. OOA provide the mechanism to deal with packaging problems as well as modifiability and organizing specification for the reviews [53] and other references.

7.5 - Software Cost Reduction (SCR) Method:

So far we have discussed the techniques to analysis the problems in different ways focusing the issues to develop a good SRS document. Software cost reduction methods [56] are applied on the real time system to Improve the feasibility as well ensures the effectiveness of advanced SE techniques. The techniques like abstract interfaces [57], cooperating process sequentially [58], information hiding and formal specification are useful to develop software which one can achieve as easier to change, understandability and maintenance.

Since there are no such technologies that can produce an effective SRS document, however this is the first step to develop software. In the SCR method [59] one it should be identify the properties of a good SRS document and principle's to develop it. The SCR techniques used mathematical and formal approaches to address the unambiguous specifications precisely and verifiability, and the same time it ensure the completeness and consistency. The SCR also provide some of the techniques that are helpful to support the packaging goal of SRS such as separation of concerns which leads to readability and easy of change. It uses the table format as a standard structure for SRS specifications to improve readability as well as modifiability to ensure the further references and also reviews. However, almost same objective provide other requirement techniques, but the Software Cost Reeducation project is unique then other, facilitate to develop SRS standard, define standard, specification and review methods provides complete and significant system which are publically available.

8 - Conclusion:

The requirements are actually hard to solve. There are needs of some discipline approaches due to a large number of essential difficulties which mislead to requirements understanding as well as their specifications. Additionally, many of the problems in requirements have not stand alone solution technically. Essential difficulties need to handle technically in order to manage complexity and one which should carefully examine the diverse audience otherwise it will lead to limited success. Discovering new methods by developers are very useful.

Complexity do not leads to uncertainty or impossibility to achieve positive goals but at the same time all the discuss approaches above have some of the weaknesses and the all of these approaches making the requirement specification systematic, controlled and efficient process. However there is not a stand-alone or some easy path that can uses for requirement specification, any of the technique which provides certainty can preferable in a discipline manner. If the requirements are written in well manner by a discipline approach there is chance to improve and the rest of the development phases can be done very well.

It is observed that the requirement specification document bear a cost because such a difficult task cannot be handled by personnel without any training or inadequate experience or some time without sufficient resources. A responsible management needed to handle such kind of efficient task adequately because some of the difficult decision should take properly at difficult time before any disaster occurs. Otherwise your project goes down!

9 - References:

1. What is the Software Development Life Cycle?
2. A Framework for Software Product Line Practice
3. Requirements Analysis Problems
4. David Brooks' 5-Step Guide to Being Deep
5. What is system design? definition and meaning
6. What is software requirements specification
7. Non Functional Requirements
8. Differentiating between Functional and Nonfunctional
9. Behavioral views for software requirements
10. Quality attributes in quality assurance and quality
11. How to Prevent the Negative Impacts of Poor Requirements
12. Prevent poor requirements – a major risk to project
13. Complexity requirements are too hard
14. The cost of product requirements
15. Technological Advances in Software Engineering
16. The role of logical decomposition in system architecture
17. How To run Multiple Versions of the Same Program
18. Addressing the essential difficulties of software engineering
19. Conceptual Structure
20. The Inherent Complexity of Software
21. Software requirements specification
22. How to Write a Technical Specification
23. Best Software Methodologies with Pros & Cons
24. Accidental Difficulty
25. Problem Analysis and Structure
26. What is software requirements specification
27. Multi-person Development of Multi-version Programs
28. Characteristics of a Good Software Requirements
29. What is the ideal SRS software?
30. What is Structured Programming?
31. Object Oriented Programming Concepts
32. Structured Vs, Object Oriented Analysis and Design
33. What is abstraction?
34. Functional Decomposition Definition
35. Mapping input to output elements automatically
36. Language Implementation
37. What is Data Dictionary
38. Data Flow Diagram - What is a DFD?
39. What is Procedural Language?
40. Embedded Control Systems Design
41. Graphical representations to formal specifications
42. What is data modeling?
43. Data Transformations - Data + Design
44. What is conceptual model? definition and meaning
45. Formal Specification Languages

46. hierarchy of functions
47. Introduction to Functional Interfaces
48. Predicate Programming Guide
49. Object Oriented Analysis and Design Tutorial
50. What is data aggregation?
51. Requirements vs. Design - Does it Really Matter?
52. Information Hiding and Encapsulation in OOP
53. Software Reviews and Comparisons
54. Object Oriented Analysis and Design Tutorial
55. Physical Structure of Management Information
56. Software Cost Reduction Introduction
57. Abstract classes vs. interfaces
58. Cooperating Sequential Processes
59. Software Cost Reduction Methods in Practice
60. Goals Of SRS Document