CHAPTER II

LITERATURE REVIEW

2.1 Introduction

Nowadays, computer applications and information systems play a very important role in helping organizations to successfully accomplish their business objectives (Hickey & Davis, 2004). As most of the shelf software do not meet the specific needs of the organizations, it becomes necessary for developing custom that were made for meeting the exclusive needs for the software (Suzanne & James, 1998; Christ, 2004). In this context, the system developers play a very crucial role in developing systems to address the needs of business organizations. System developers have to bear in mind that the systems have to be developed, in such a way that, they appropriately meet the objectives and the goals for the organizations. However, prior to the development of the systems, the process of collecting user requirements plays a significant role in the success to the system development.

On the other hand, collecting user requirements is one of the biggest challenges faced by system developers, because the users might not be able to tell developers what they need or might not be able to explain their work processes.

Hence it is crucial to choose the right elicitation methods to successfully accomplish the software development project. It is imperative to choose the right method based on the situation. The main problem arising is the inaccuracy of the requirements gathered from the stakeholder (mainly the users). Besides that, the conventional methods are inaccurate (Zowghi & Coulin, 2005).

2.2 Functional and Non-Functional Requirements

Requirements are described as the precise requirements that a program is predicted to fulfill (Azuma, 2004). The kind and the excellence of the specifications were gathered as "functional" and "non-functional" categories employed by the researcher, based on the classifications as in the following discussion.

There exists a lot of definition for the term "Requirement". According to Oxford dictionary (Oxford dictionary, 2013) requirement is defined as "that which is required", which means, having need, finding something necessary. However, in terms of software engineering requirements can be defined as "a condition or capability needed by a user to solve a problem or achieve an objective" (Dean & Don, 2000). Requirements can be further classified as, functional requirements and non-functional requirements, where the former is defined as a function, which a system or system component must be able to accomplish (IEEE, 1990). It can be described in a variety of means. The most typical ones are, information written in documents, and use cases, where, use cases are usually textual descriptive lists, and diagrams that illustrate the actions of users.

Functional requirements are those, which are ideal to a necessary function, which the system must accomplish. It explains "what" functions are carried out (ISO/IEC, 2007). Non- functional requirements: are also regarded as quality specifications or limitations (Azuma, 2004).

Non-functional requirements are associated with "how" functions, such as performance requirements, precise quality needs, and restrictions (Azuma, 2004; ISO/IEC, 2007; Boegh, 2008; Glinz, 2008). Instances of non-functional requirements consist of employing uncomplicated user interface for a beginner, using radio buttons for option selection, deal with all the software or system functions on the security part, to protect the processes and the data against any probable attacks. The non-functional are any other requirements needed by a system or system component, other than

functional requirements. There are three types of non-functional requirements these namely are Data requirements, Constraints and Quality requirements.

Data requirements are generally assembled collectively with functional requirements, in requirements, and they illustrate how functional requirements need to be replicated in the system. The Constraints are the aspects, which obviously and deliberately limit the system or process. A crucial feature of a constraint is that, it results in a damage or a failure, if it is not properly addressed. Constraints include limitations of the engineering process, systems or system components' functionality, or its life cycle. In quality requirements, these explain the required qualities of the product, which are not specifically associated with functionality. It is essential to maintain the quality of software, hence addressing the quality requirements becomes mandatory. Lack of quality requirements leads the projects miserably to failure. Simultaneously, quality requirements are usually challenging to get or elicit, as against the functional requirements. Basically, users, in interviews or discussion groups, naturally talk about their roles and responsibilities, while, quality is often implied to the knowledge domain, and therefore they do not talk naturally about it. Observations, and studies of similar systems, usually offer more information about functionality than quality (Abdukalykov et al., 2011).

2.3 Requirements Significance

Analysts and experts have reported considerable proofs in the research literature, related to software quality requirements that facilitate successful software delivery. Nelson (2007) has reviewed 10 of software project's failures over the span of two decades. The study has specifically ascribed three of those downfalls to poor requirements analysis. It has been claimed that poor requirements are in eighth position in his list of the top 10 errors made by developers. Wiegers have argued "If you don't get the requirements right, it doesn't matter how well you execute the rest of the project" (Wiegers, 2006). McGovern has declared that "the critical success factor will always be the accuracy and completeness with which the business requirements and goals are captured and traced to the associated details" (McGovern, 2007).

According to Zowghi (2004), the intent of requirements analysis, is to elevate the probability of building the right system, i.e., upon completion the system must satisfy the targeted customers and address their needs to a satisfactory degree. Bergey, et al. (2009) have highlighted that if the quality requirements are not proficiently identified, the ensuing system cannot be properly assessed for success or failure before implementation. The IEEE SWEBOK, (2004), was stated that "It is widely acknowledged within the software industry that Software Engineering (SE) projects are critically vulnerable when these activities [elicitation, analysis, specification, and validation] are performed poorly".

2.4 Requirements Engineering

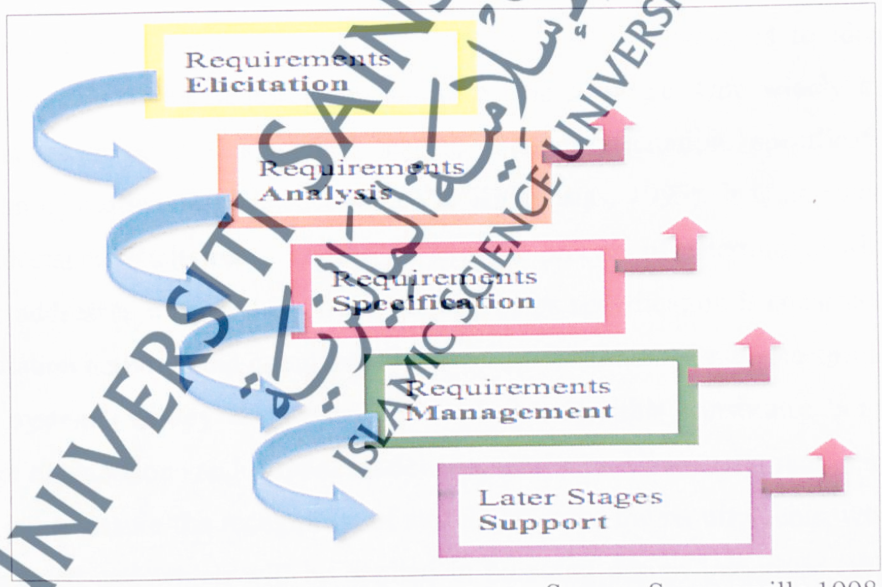
Studies have exposed that problems associated with Requirements Engineering (RE) could cost 10-200 times more to rectify the program after its implementation, than if they were recognized during specifications (Boehm & Papaccio, 1988; McConnell, 2001; Romero & Mariona, 2010). Few other researchers have suggested that, the comprehensive amount of project budget due to requirements flaws is twenty-five to forty percent (Wiegers, 2003). It is noteworthy that, RE is not only crucial, but also irreplaceable. Moreover RE will be disadvantageous if the acceptable sources are not devoted early.

Requirements engineering "is an activity, which aim is to discover, document and maintain a set of requirements" (Pamela, 1997; Raimundas, 2005). "The use of the term engineering implies that systematic and repeatable techniques should be used to ensure that system requirements are complete, consistent and relevant". (Sommerville & Sawyer, 1997). Another definition according Software Test and Evaluation Panel (STEP) is "the disciplined application of scientific principles and techniques for developing, communicating, and managing requirements" (STEP, 1991). Similarly, Loucopoulos and Champion (1989) have defined RE as "the systematic process of developing requirements through an iterative process of analyzing a problem, documenting the resulting observations, and checking the accuracy of the understanding gained". Both these definitions indicate that RE is not just a single activity, rather it is a process, which comprises a variety of phases and actions. Davis

(1990) has pointed out that RE can be classified into elicitation, solution determination, specification, and maintenance. Dorfman has segregated RE into five phases, such as, elicitation, analysis, specification, validation/verification, and management (Dorfman, 1990). Easterbrook and Nuseibeh (2000) have identified that RE has the following phases; eliciting requirements, modeling and analyzing requirements, communicating requirements, agreeing on requirements, and evolving requirements.

Even though the classifications of the RE process which mentioned above have never been disputed, it is implied that the researchers have failed to include a phase to support requirements during the early stages of development. Therefore, some requirements of process engineering have been considered. The requirements are composed of five distinct, but related phases. Figure 2 below shows these phases; however, security RE are not considered to be a sequential process (as each phase can and should affect the others).

FIGURE 2: Requirements Engineering Phases Surveyed.



Source: Sommerville,1998

2.5 Requirements Elicitation

A quite number of authors have stated that requirement elicitation is the first phase of the process, by which a project team determines the organizational needs that must be

addressed by the project effort (Goguen & Linde, 1993; Loucopoulos & Karakostas, 1995; Kotonya & Sommerville, 1998; Escalona & Koch, 2004). Particularly, requirements elicitation concentrates on the preliminary pursuit of identified requirements and possibilities of the social actors (e.g., participants, users) most strongly associated with the system. A report from IEEE (2004) claims that requirement elicitation concerned with software requirements originate from and how the SE can gather them. It is the first stage in understanding the problem, which needs the software to be solved. In addition, Browne and Ramesh (2002) have defined requirement elicitation as the first step in gathering user requirements; it is the process of understanding and acquiring the needs of users and other stakeholders.

The term "elicitation" is used to capture the idea that, the individuals, who will be engaged within the system on a daily basis, are the appropriate source for the identification of needs and opportunities, and that designers must "draw" the requirements from such stakeholders. It is important to note that the requirements elicitation process focuses on "what" is expected to be achieved by the predicted system irrespective of "how" to be achieved.

In the literature, a range of process models have been developed to identify the various features of the requirements phase of the software. One widely employed model suggests three fundamental stages, such as elicitation, specification, and validation of requirements (Loucopoulos & Karakostas, 1995). It is noteworthy that the requirements elicitation is concerned with the process of determining what issues must be addressed by a design effort. Requirements specification is constructed upon the elicitation activity, and comprised the specific documentation of the specifications for the system (Vessey & Conger, 1994). Generally, this constitutes both natural language explanations and official modeling techniques. Ultimately, requirements are carried out to assure the recognition of stakeholders for the requirements, which have been reported and which will be applied in pursuing design initiatives (Wallace & Ippolito, 1997). While this three-stage model indicates a linear strategy to develop requirements. The three phases are generally employed iteratively, often moving progressively in more detailed levels of requirements gathering.

Just as the requirements phase process is critical to the overall success of software design efforts, the requirements elicitation also plays a crucial initial role in the extensive requirements elicitation process. One of the crucial features of requirements elicitation is that, it is typically one of the most important components, by which the project team members acquire knowledge related to organizational domain. Loucopoulos and Karakostas (1995, P.21) have stated that "the importance of requirements elicitation cannot easily be overestimated; when you have to solve somebody else's problem the first thing you have to do is to find out more about it".

The emphasis of the requirements process is generally presented as the process revealing the essential obstacles within the business context, generally known as "problem domain." Consequently, traditional requirements elicitation strategies signify a shortfall based mode of inquiry, which is participating "in a study of what is unsuitable, not working, not up to standard, and in need of a 'fix'" (Whitney, 1998).

In the following section, an overview of the most relevant techniques used in requirements elicitation are presented in the context of a standard software development process. In addition, the classifications of requirements elicitation techniques are demonstrated and briefly highlight a number of the most widely employed methods of requirements elicitation and discuss the associated strengths and the challenges.

Overall, the goal of requirements elicitation is to force the analyst, user, and other stakeholders to understand the requirements that they want to address. Defining requirements calls for effective interaction and open communication between the user and the developer to generate the necessary requirements they want to address and the information that can be used to develop the system that meets the needs of the user (Guinan & Bostrom, 1986).

2.6 Various Requirements Elicitation Techniques

The requirements elicitation techniques facilitates the developers to have an understanding of the requirements for the users, this phase allows the developers to

recognize the requirements of stakeholders, other than the actual users of systems (Browne & Ramesh 2002; Hickey & Davis 2004). Nevertheless, Farzan et al. (2008) have stated that elicitation is the process, which motivates the dynamic contribution of users in system development, which consequently increases the accomplishments and endurance of information systems.

According to Zhang (2007), specifications progression is an rigorous communication procedure among stakeholders and the programmers. Therefore, the four kinds of elicitation methods vary depending on the context of communication according to (Zhang, 2007):

a. conversational
b. observational,
c. analytic, and
d. synthetic

Every kind provides a unique communication model among programmers and stakeholders, and demonstrates the characteristics of a strategy. Realizing the category of method facilitates engineers to comprehend various elicitation methods and guides them to choose a suitable technique, for requirements elicitation.

a. Conversational Methods

The conversational method provides a means of verbal communication between two or more people. As conversation is a normal means to convey requirements and concepts, and ask and answer questions, it is efficient to build and comprehend the issues and to elicit generic product requirements (Avison, 1995). Methods in this group are also referred as verbal methods (Avison, 1995). A standard conversational strategy is interview. It is generally used in requirements elicitation (Kotonya & Sommerville, 1998). Other methods under this category include workshop, focus groups and brainstorming (Table 1).

TABLE 1: Conversational methods for requirements elicitation

Method	Literature support	Executor	Illustration
Interviews	(Goguen&Linde,1993; Julio & Ana1996; Kotonya & Sommerville, 1998; Leffingwell & Widrig, 2003)	An experienced analyst with generic knowledge about the application domain	Interviewer discusses the desired product with different groups of people and builds up an understanding of their requirements (Interviewer asks questions to the specialist or end-users, related to a particular topic). If the interview is conducted with pre-defined agenda and questions, it is called structured interview; otherwise, it is an open ended interview
Workshop, focus groups	(Goguen & Linde,1993; Leffingwell & Widrig, 2003)	An experienced outside facilitator	Stakeholder representatives gather together for a short but intensely focused period to create or review high -level features of the desired products
Brainstorming	(Leffingwell & Widrig, 2003; Zowghi & Coulin, 2005; Tsumaki & Tamai, 2005; Chauncey, 2006; Linn, 2008)	An experienced outside facilitator	Brainstorming is a kind of mini conference, held among six to ten experts. Members from different walks of life involve in the brainstorming, chaired by the organizer, who asserts the topic to be discussed

One of the most common types of social relationship is conversation. Generally, people are pleased to express about the jobs they perform, and the challenges they encounter. The verbal needs and limitations are generally called non-tacit specifications. As oral communication is realistic and productive to gather non-tacit knowledge, the conversational methods constitute the principal approach to non-tacit requirements elicitation, by carrying out interviews, workshops or brainstorming sessions (Maiden & Rugg, 1996). Generally, conversational strategies are extremely used in requirements development, however not by itself; they require combining other kinds of techniques to accomplish the software development phase. However, they require a lot of human efforts (Christel & Kang, 1992; Goguen & Linde, 1993): conference sets up and creates transcript and examining records of a live conversation, and consume a lot of time. On the other hand, it aims to accomplish the elicitation process, particularly when employing workshop or brainstorming, e.g. organizing the meeting and making sure that the representatives are available for the meeting (Kotonya & Sommerville, 1998).

b. Observational Methods

The observational method provides a means to develop a rich understanding of the application domain by observing human activities (Zhang, 2007). In addition to non-tacit requirements, some requirements are obvious to stakeholders, but difficult to verbalize, which is called as tacit requirements (Maiden & Rugg, 1996). Verbal communication is frequently weak, when gathering tacit requirements. Consequently, observing how people perform their routine work facilitates gather information, which are challenging to explain in words. Methods under this group are illustrated in Table 2.

TABLE 2: Observational methods for requirements elicitation

Method	Literature Support	Executor	Illustration
Social analysis and Ethnographic study	(Maiden & Rugg, 1996; Kotonya & Sommerville, 1998)	The observer must be accepted by the people being studied as a kindred spirit and must be sufficiently familiar that they carry on with their normal practices as if he was not there	An observer spends a period in a society or culture on making detailed observation of all their practices, (interact with a product in a natural environment). User culture and work environment are observed
Observation	(Goguen & Linden, 1993; Viler & Sommerville, 1999; Nuseibeh & Easterbrook, 2000)		
Protocol analysis	(Goguen & Linde,1993; Maiden & Rugg, 1996)		A subject is engaged in some task, and concurrently speaks out loud and explains his thought

Observational methods seem to be best suited, when individuals find it complicated to communicate their requirements and when analysts search for an improved comprehension of the perspective, where the preferred product has to be utilized (Viller & Sommerville, 2000). Good examples of tacit information consist of the scheduled work, which individuals accomplish day-to-day, spontaneously and in the organizational or social contexts, which most likely affect the specifications. As people are acquainted with the perspective and scenario to their work, they do not deliberately contemplate about the schedules and the working environment. It is challenging for them to enunciate how work is performed, despite the fact that occasionally the routine work is simple to be revealed to others (Kotonya & Sommerville, 1998). Therefore, to be engrossed in the actual work scenario, to acquire the observational facts, can assist engineers to thoroughly, understand the routine of work, the societal group, the organization, and the wider perspective, within which the

product is used. As observation methods fall into the category of longitudinal studies, in general, it takes longer period than the other methods (Myers, 1999), which is considered as main disadvantage of such methods, especially when the project has tight schedule at the requirements stage. Besides this, observation requires compassion and receptiveness to the physical environment (Zhang, 2007). It is easy for observers to perceive a good image about the work's context, but it is normally hard to specify and analyze their perception.

Observational methods are used for understanding complex societies, rather than making judgments about improving or supporting the ways of working (Kotonya & Sommerville, 1998; Zhang, 2007). They are beneficial to discover fundamental elements of routine order, such as the standard design of work, and offer the most relevant information towards designing solutions (Hutchings & Knox, 1995). Consequently, it is generally a good practice to begin with an observational method, to get a preliminary comprehension of the preferred product, when the development team falls short of experiences of product development in a given domain.

c. Analytic Methods

Analytic methods provide ways to explore the existing documentation or knowledge and acquire requirements from a series of deductions. These methods are illustrated in Table 3.

TABLE 3: Analytic methods for requirements elicitation

Method	Literature Support	Executor	Illustration
Requirement reuse	(Goguen & Linden, 1993; Kotonya & Sommerville, 1998; Viler & Sommerville, 1999; Cybulski & Reed, 2000; Knethen et al., 2002; Woo & Robinson, 2002)	Documentation	In this technique the systems within the same product family is used to identify requirements of the desired system
Documentation studies /content	(Kotonya & Sommerville, 1998; Cybulski & Reed, 2000; Knethen et al, 2002)	Documentation	A common method consisting of reading and studying available documentation for content that is relevant to and useful on the requirements elicitation tasks
Analysis Laddering	(Rugg et al., 2002)	Expert 's knowledge	It involves the creation, reviewing and modification of hierarchical content of expert's knowledge, often in the form of ladders (i.e. tree diagrams)
Card sorting	(Rugg & McGeorge, 1999)	Expert's knowledge	The expert is asked to sort into groups a set of cards each of which has the name of some domain entity written or depicted on it

A wide range of studies have focused on the requirements of the preferred product, which consists of problem evaluation, organizational charts, specifications, user manuals of existing systems, research report of competitive systems in market. By understanding it, the engineers capture the information about the application domain, the workflow, the characteristics of product, and map it to the requirements specification (Rugg et al., 2002). Furthermore, they recognize and reuse requirements from the specification of the heritage or identical products. It is always worth searching and filtering for reports and recorded information, relevant to the desired product.

In analytic methods, the mapping techniques are beneficial for knowledge acquisition (Byrd et al., 1992). Multidimensional scaling (Wright & Ayton, 1987; Byrd et al., 1992) allows users to obtain conceptual structure, to recognize aspects and identify cause-effect associations of a process, and variance analysis (Hawgood et al., 1978; Byrd et al., 1992) to use existing system as a basis for determining new system requirements. Generally, these techniques are considered as knowledge acquisition strategies; however, they are also flexible in requirements elicitation.

As explained in Table 3, laddering (Rugg et al., 2002) is utilized to elicit justification and explanation of technological terminologies or subjective terms, and to elicit, how specialists composite their knowledge about a field, and card sorting (Rugg & McGeorge, 1999). Generally, the analytic strategies are not essential to requirements elicitation, because requirements are grabbed ultimately from other sources, instead of end users and clients. Nevertheless, they form secondary variants, to enhance the performance and usefulness of requirements elicitation, particularly when the information from heritage or relevant products is re-usable.

d. Synthetic Methods

According to Maiden and Rugg (1996), the synthetic strategies incorporate various channels of communication, and offer models to illustrate the characteristics and the relationship to the system; they deliver good hints for requirement recognition, in the form of abundant semantic models. For instance, the prototypes offer an initial version

of the system to the users, which can emphasize users about the functions, which are usually in any other case ignored. Storyboard technique, which is categorized between scenarios and prototyping. It presents a procession of prospects beginning from sample components to live interactive reports (Leffingwell & Widrig, 2003). Appreciative Inquiry is a combined method, which includes communication between clients and designers, examine the existing systems, tracking the behaviors of the users and visualize the future system or software, with all the necessary functions. It includes combined strategies, which enhance requirements elicitation process. Examples of synthetic methods are illustrated in Table 4.

TABLE 4: Synthetic methods for requirements elicitation

Method	literature support	Executor	Illustration
Scenarios, passive storyboards	(Kotonya & Sommerville, 1998; CREWS, 1999; Stuart, 2001; Leffingwell & Widrig, 2003; Zowghi & Coulin, 2005)	Analysts and stakeholder representatives communicate and coordinate to reach a common understanding of the requirements	This approach describes the precise details of the current and future processes, which comprises the actions and interactions between the users and the system
Prototyping, Interactive storyboards	(Kotonya & Sommerville, 1998; Andriole, 1999; Nuseibeh & Easterbrook, 2000; Leffingwell & Widrig, 2003)		It provides stakeholders with a concrete (although partial) model or system that they might expect to be delivered at the end of a project. It is often used to elicit and validate system requirements. Prototype generally represents and visualizes the actual parts of system
JAD/RAD	(Maiden & Rugg, 1996; Coughlan & Macredie, 2002; Zowghi & Coulin, 2005; Charles & Francesca, 2008)		It stands for Joint Application Development. Rapid Application Development and emphasizes user involvement through group sessions with unbiased facilitator. This approach is more or less similar to the brainstorming, however, it differs is one aspect, where the stakeholders and the users are also allowed to participate and discuss on the design of the proposed system
Contextual inquiry	(Holtzblatt & Beyer, 1993)		It is a combination of open-ended interview, workplace observation, and prototyping. This method is primarily used for interactive systems design where user interface design in critical
Appreciative Inquiry (AI)	Cooperrider&Srivastva,1987; Cooperrider&Whitney, 2001; Barrett & Fry, 2005; Whitney&Trosten-Bloom, 2003; Stavros&Torres, 2005; Kelm, 2005)		Appreciative Inquiry (acronym "AI") is principally an organizational development method, which focuses on increasing what an organization does well rather than on eliminating what it does badly

The synthetic methods are generally integrated at other phases of the product development life cycle. As the purpose of synthetic methods is to enhance the communication among programmers and the clients, they are appropriate for various

phases of the development process. They efficiently coordinate the requirements stage with the remaining development processes.

2.7 Comparison between Categories of Requirements Elicitation Techniques

The categories are due to the outcomes of dissimilarities between the requirements elicitation techniques, every group has some characteristics, which make all categories to have distinctive characteristics. Table 5 shows a comparison among various groups of elicitation techniques and their benefits and drawbacks.

TABLE 5: Comparison between various types of elicitation techniques

Type	Example	Stakeholder Participation	Observable phenomena	Future system Knowledge	Understanding the domain	Identifying sources of requirements	Predictive ability of unique attributes elicited
Conversational	Interviews	Definitely stakeholders are the main participants	Not applicable	Some hints or guidelines are provided that might guide the future development of systems. Novel ideas might lead the analyst to envisage future system	Diverse, knowledge sharing might make analyst to understand domains	Nil	Nil
Observational	Observation	Stake holders do not participate, as this method totally involves the observer to observe the activities of the end-user	The whole process is based on the phenomenon of observation (s), hence this criteria is very crucial	Observing the current methods are not easily applied to the development of future concepts	Observational methods clearly makes the analyst to understand the domain	Nil	Depends on the observer
Analytic	Requirement reuse	Stake holders do not participate	A lot of factors including the codes are observed	The analyzing of codes and other existing documents will pave way for getting knowledge about the future system	Again, the code analyzing process will reveal the facts of domain	Definitely, this method needs makes the analysts to understand and identify the source of requirements	Nil
Synthetic	Appreciative Inquiry (AI)	involves collaboration between stakeholders and systems analysts to identify needs or requirements	Some parts of the process is based on the phenomenon of observations, hence this criteria is very crucial also Behaviors of people are observed	The strong Synthetic methods practices, make people to give more ideas about the future systems, Hence gaining knowledge about future system is one of the biggest advantage of this method. Synthetic methods are a strategy for purposeful change that identifies the best of "what is" to pursue dreams and possibilities of "what could be"	As the experts will be made to converse about the domain, it becomes very effective means to understand the domain. This type makes the analyst to unambiguously understand the domain	Synthetic methods generates volumes of data that provide great detail on the origins and consequences of local needs and resource constraints	It's one of the most important features is the predictive and find a unique attributes

Based on the explanation about the various types of elicitation technique, it is evident that, every category has its own advantages and disadvantages. The table above has clearly illustrated the pros and cons of each technique. The conversational methods have many advantages, such as the actual fact, which is very helpful for collecting loaded information about the requirements. Apart from unearthing opinions, the interview technique identifies feelings and goals of different individuals. With the help of interviews, it is easy to dig the details by asking follow-up questions. However, the conversational methods have the following disadvantages: it is very difficult to master the skills of interviewing. The requirement elicitation depends a lot on the behavior and attitude of interviewer. Moreover, the interviewer has to be always unbiased. Even though a lot of information can be collected with the interview method, getting meaningful information may be still uncertain.

The benefits of observational methods can be summarized as follows; the observational methods are better option for fetching basic aspects of routine order. Furthermore, they offer crucial information for designing solution. These methods are very handy, when the development team lacks experience about product domain. However, there are some problems in practicing the observational methods. The most critical drawback is that, observational methods need a lot of time, and these techniques are not good choices, when schedule is tight. Similar to conversational methods, the observational methods are also very difficult to understand thoroughly. Furthermore, observational methods need sensitivity and responsiveness towards physical environment.

Many studies have discussed the analytical methods and give the techniques that are related to this category as mentioned before, for example reusing the requirements of the existing system as a common method for requirements elicitation. There are many advantages of using the existing knowledge to develop the new product, which includes low cost and less time. Despite the disparities of type for users and stakeholders, this method is used very commonly particularly in developing user's interfaces, database and security policies.

Many projects have failed due to the employment of inappropriate elicitation methods. One such example of big project failure is "Ariane 5 Flight 501 (European Ariane 5 expendable launch System)", where, the requirements specifications of Ariane 4 were reused. However, the flight path of Ariane 5 was very much different; hence, the system which developed using the requirements of Ariane 4 was unable to handle the Ariane 5 flight path. This is an important example of the disadvantages of analytic methods, because the failure was found after testing the new software or system. Thus, these methods reuse the old code or software that might completely or partially different from the new software.

The synthetic methods, it is particularly valuable for stakeholders such as, business owners and end users who might not understand the technical aspects of requirements. However, a visual representation of the end product would help giving more clarity. Prototyping as an example may be an interactive screen, a mock-up, a navigation flow or a storyboard. Simple, throwaway prototypes might be executed in the initial stages of discovery, and more detailed, interactive prototypes will be developed once business requirements have been identified (Zhang, 2007).

Another example under the same category is AI technique. In order to clearly understand this technique, it is crucial to dissect the terms and comprehend the meaning in the context: *Appreciation* means to recognize and value the contributions or attributes of things and people around us. *Inquiry* indicates the exploration and identification of possibilities of novel ideas. When combined, this term means that by appreciating what is good and valuable in the present situation, it is possible to discover and understand the means to institute positive change for the future (Kelm, 2005).

The positive aspect of AI is that this approach is constructed upon the advantages of an organization or group. It understands things that are well implemented. Consequently, a very beneficial and optimistic influence on spirits, guarantee and value of individuals and groups, contributors can become empowered and encouraged. It is very easy to involve people, who do not generally get engaged in this kind of activity, due to the fact of the conversational style of questioning and specific focus on

the participants (Gonzales, 2011a; Gonzales & Leroy, 2011). However, the disadvantages of the AI are: it consumes time, needs periodic commitment, to motivate participants and occasionally additional works should be done to get people out of the SWOT (strengths weaknesses, opportunities threat) mindset.

2.8 Why Appreciative Inquiry (AI)?

The first appearance for AI method was in 1987, created by Cooperrider and Suresh Srivastva, They built it to use it in social sciences in order to build the organizations, after that it has been suggested by Carol K. Gonzales and Gondy Leroy in 2011to use the principles of AI that were represented by Cooperrider in 2008. The aim of AI is to build the organizations, processes and systems using the existing successful cases in software development. It is necessary for the developers to find the real requirements for the system development and meeting such needs. Therefore, the information delivery can be more factual by making better processes and new systems in the hope that the developed system will meet the current requirements and meet the future needs.

Elicitation process which uses the AI is carried out to established these goals of the system which is to satisfy user requirements. The advantage of AI is the evaluation for its effectiveness with eliciting user requirements under different circumstances. The results were positive for the participants, the requirements obtained, and the general requirements eliciting process, in addition the studies demonstrated benefits to the requirements that gathered by increasing the number of unique requirements and identifying more quality in gathered requirements (Gonzales, 2011a).

Cooperrider and Srivastva (1987) were the pioneers to focus on the AI as a technique for collecting requirements. They have revealed three important aspects in terms of the conventional approaches: (i) these approaches did not yield the expected outcome and hence were counterproductive, (ii) organizations are deemed as collectively constructed actualities, which were only restricted by human imagination and the shared beliefs of organizational members. In this context, the types of investigation were powerful in building the systems as intended; these problems which aimed to be

solved probably created more problems rather than solving them, (iii) new ideas were the most significant force for change. They have declared that the lack of new ideas in the conventional research is a crucial problem, and hence they have proposed AI as a method, which probably created new ideas, images, and theories that would lead to social innovations.

The evolution of AI has witnessed the introduction of new theoretical justifications and explanations. Cooperrider & Whitney (2001) have proposed the most dominant five principles of AI. Furthermore, additional principles have been proposed by Barrett and Fry (2005). The five principles have been widely accepted (Whitney & Trosten-Bloom, 2003). Many studies related to AI have quoted these principles (Bushe & Kassam, 2005) and non-organizational applications of AI e.g. (Stavros & Torres, 2005; Kelm, 2005). The benefits and bottlenecks of the AI approach are extensively studied, which leads to the emergence of novel ideas and mechanisms.

The AI was proposed by David Cooperrider (1987). However, he refrained from explain of its usage, because he wanted people to focus more on the principles of AI, rather than viewing it as an approach. This has resulted in the emergences of novel ways of conducting AI. During first fifteen years from its introduction, the AI practitioners employed the initial set of four principles (Discovery, Dream, Design and Destiny) as proposed by Cooperrider and Srivastva (1987), which stated that inquiry into the social potential of a social system should begin with appreciation, collaborative, provocative, and applicable. The original method which called for a collective discovery process used "Discovery" grounded observation to identify the best of '*what is*'. "Dream" vision and logic to identify ideals of what '*might be*'. "Design" collaborative dialogue and choice to achieve consent about '*what should be*'. "Destiny" collective experimentation to discover *'what can be'*.

FIGURE 3: Core process (iterative 4-D cycle).

Source: Gonzale, 2011a; 2011

It was not until 1997 that the 4-D model of AI, as shown in figure 3, now almost universally termed as the AI method, was created. Diana Whitney, Cooperrider's collaborator on some of the first AI projects in the 1990's, had a major impact on the evolution of the practice of AI and the most authoritative sources on AI practice are Ludema et al., (2003), Whitney et al., (2003) and Cooperrider et al., (2008).

Appreciative Inquiry proved that it can extract and elicit new and unique requirements; because it focuses on the requirement phase. Have been inserted in the SDLC by Gonzales, Carol K. Leroy, Gondy in 2011 (Gonzales & Leroy, 2011) and proved it is able to provide us a unique requirements.

As it concluded from the above that if integrate the security with SDLC will get a success software, especially in requirement phase, AI proved that is very good in elicit software requirements as mentioned earlier in literature review, Involvement AI to elicit software security requirements through SDLC in particular in requirement phase will gain a successful secure software.

One technique that can produce more accurate and complete requirements is AI (Gonzales et al., 2009). Because of its positive and goal-oriented nature and its use of the participant's 'language', it is indicated that AI can be adjusted to capture user's requirements and address the aforementioned challenges (Gonzales, 2011b). Improvements are needed to address the high costs associated with faulty software requirements. It is also considered that the AI, and its positive approach, can improve gathering requirements (Gonzales, 2011b). AI approach follows certain steps, but it can go into the same SDLC phases, but all AI phases focus on requirement phase to get some unique requirements.

Appreciative Inquiry helps users to imagine future tasks while it encourages them to think about past successes. AI is also similar to scenario-based approaches for eliciting requirements. Scenarios are designed to elicit knowledge through non-routine scenarios as a means of limiting automaticity and improving recall (Browne & Ramesh, 2002).

In conclusion, AI can increase positive methods by presenting a positive future aspect for proposed software resulting in improving the elicitation requirements. There have been little example for valuations which have been conducted to improve requirements. One was part of a system analysis course taught at Case Western Reserve University, Ohio USA, to learn accelerated requirements specification. AI was modified and applied to several system development projects showing success with users during SDLC, effectiveness of the requirements gathered, and creating a common understanding (Bergvall-Kareborn et al., 2008). AI was discussed as a means to improve the motivation to adopt Knowledge Management Systems as well as promote the creation and exchange of knowledge due to its story sharing and positive approach (Avital, 2004).

2.9 Quantifying Software Requirements

According to Eide (2005), quantification "is the ability to identify measurable, mutually properties of similar requirements and the elicitation of corresponding values for each requirement". However, according to Oxford dictionary (2013), a more standard definition of quantification is the ability of determining the quantity, or measuring or expressing as a quantity. Furthermore, the term quantity is defined as the property of things that is principally measurable (Oxford dictionary, 2013). However, in case of requirements, this means to determine components or features of requirements that are quantifiable. Nevertheless, if this is expected to add value to projects, it will not be sufficient to independently determine these attributes for every requirement. For the purpose of gaining the added values, it is essential to categorize the requirements of projects, with identical attributes.

As soon as the common and quantifiable attributes of a category is recognized, then it is essential to elicit the value of each attribute of each requirement. Then the requirements can be appropriately, and probably automatically, authenticated and examined, for obtaining quantification of the requirements. According to Maiden and Maiden (2006) quantifying requirements is one of the most significant challenges faced by requirements analysts. Generally, analysts ask how to perform quantification and what techniques support it. Nevertheless, quantification is also a challenging process, but it improves requirements during the requirements process. Even though quantifying requirements can be difficult, two quantification techniques, such as Volere (Robertson, 1999) and Planguage (Gilb, 2005), which are the most common in quantifying requirements have been discussed.

a) Volere:

Suzanne Robertson and James Robertson (1999) have developed Volere requirements process; Volere is a complete lifecycle approach, employed for collecting project requirements, which recognizes the important attributes of requirements. Fit criterion is one of the most crucial attributes, which describes quantified objectives, which the system must precisely meet the accepted criteria. Even though description of requirements is written in the stakeholders' language, the fit criterion is written in a specific, quantified method, to ensure that, analysts can analyze solutions against the requirement (Maiden & Maiden, 2006). Outlining fit criteria generally modifies the primary requirement by demanding it to be quantified, where the Volere's requirement specification becomes template. Volere employs the requirement type for the development to fit the criteria. Each type of the requirement has features, which can be quantified using analysis. For example, in Volere, usability requirements have two features, such as, ease to use, and ease to learn each feature, Volere guides for quantification in the form of content, motivation, examples, and fit criteria (Robertson, 1999). Volere also provides a straightforward and effectual foundation for quantifying the requirements. Requirements Types that depend on Volere method are:

- Functional requirements: are the essential materials of the product.
- Non-functional requirements: are the attributes required by the functions, such as efficiency, security and usability.

- Project constraints: the attributes that restrict the product due to the budget or the time available to develop the product.
- Design constraints: inflict limitations in the process of designing the product.

b) Planguage:

Tom Gilb (2005) has developed Planguage, a language influenced by keyword, which helps analysts to write quantifiable, testable quality requirements. Gilb has asserted that, one could possibly control those that he/she can measure. Quantification and measurement are essential for anyone who wants to deal with requirements. In this context, Planguage provides more precise process guidance for the analyst; furthermore, it has also been extensively employed to address industrial problems. The most significant concept of Planguage is a scale of measure, which describes measurable system attributes such as, dependability and functionality. A scale describes a functional description of what is being assessed, and the unit of assessment.

Planguage has been designed to quantify qualitative statements in plans, specifications, and designs. It has plenty of benefits such as, easy to learn, adaptable, compact, extensible, and avoids oversights by offering a reliable range of parameters, for quality requirements (Simmons, 2001). Planguage makes sure that, analysts accomplish all estimations and measurements according to the scale. Due to the fact that, analysts must be able to measure, the position of requirements on a scale is essential. Planguage offers meters, generally known as tests, which are realistic methods for measuring. Consequently, it develops Volere's measurable fit criteria with mechanics for quantification, utilizing scales and meters. Furthermore, Planguage also acknowledges that, quality requirements such as, functionality can be complicated, and hence it possibly decomposes them.

Volere and Planguage can help you challenge and improve your requirements with numbers, but still general techniques, did not assert the quantification of requirements alone, but also it focused on other criteria such as reliability, usability requirements, measurable and testable quality requirements (Non-functional

requirements). Moreover, Planguage which is known as a test method, is practical method for measuring.

2.10 Related Issues in Security Requirements Engineering

Software systems are exposed to frequent and critical security threats, over the years technology has witnessed growth in both negative and positive sides. It has become easy for the intruders to effortlessly attack the systems. Hence, there are some issues related to the security requirements engineering as follow:

2.10.1. Security Requirements

It is noteworthy that, security is a non-functional requirement, which becomes more and more vital and distinctive in its needs, nonetheless certain needs are coupled with all other functional and non-functional requirements, and structured into productive architectures, models, and application. Just like other non-functional requirements, the distinctive characteristics and needs of security, make it challenging and generally unproductive, to get specified with traditional requirements methods, which makes the necessity for employing security requirements engineering (Romero & Mariona, 2010). In the next section the software security, and security requirements engineering shall be explained. It is thought that it is essential to focus on requirements elicitation for the purpose of avoiding intentional or accidental attacks. It is essential to concentrate on security aspects right from the word go, in line with other aspects of the system.

2.10.2. What Software Security Is?

According to McGraw, (2003), it is crucial to realize the security challenges induced by software and how to manage them. Furthermore, Stoneburner et al, (2001), have considered security as a system property, they have claimed that, "security is much more than a set of functions and mechanisms; IT security is a system characteristic as well as a set of mechanisms that span the system both logically and physically."In terms of non-functional requirements, security is considered as an exclusively complicated and difficult aspect; according to Ian Alexander (2002), "security is

unlike all other areas in a specification, as someone is consciously and deliberately trying to break the system".

Software security comprises three principal objectives, such as, confidentiality, integrity, and availability (CIA) of the information resources created, processed and stored by the software. Similarly, conservation of confidentiality indicates the avoidance of illegal disclosure; whereas, conservation of integrity refers to preserving information from illegal modifications; and the conservation of availability indicates preventing unauthorized devastation or rejection of access or service (Redwine et al., 2004; Romero & Mariona, 2010).

Unfortunately, quite often, the developers fail to give prominence to security from the very start of a software development, rather it is only included only after the software has been developed, consequently, most of the software are vulnerable to unwanted attacks (Khan, 2008; Hans, 2010; Futcher (a), 2011). Viega and Mcgraw (2001) have stated that generally, software is developed with nominal focus on security. Therefore, Romero and Mariona (2010) have stressed on the significance of security in software development; especially due to the consequences software security over society. According to Redwine et al. (2004) there are many different attacks by human via malicious software from external areas over internet network, grow significantly. Furthermore, the insider attacks are still dangerous.

Software security has to be considered critically, however, Lynn and Michael (2003) considers that, fundamentally software security is a very challenging endeavor, due to the following two primary factors:

a) Omnipresence of networks

Omnipresence of networks as a consequence of increasing connections of computers, by means of the Internet, there is a huge rise in the number of attacks and the effortlessness of making those attacks. Nevertheless, the development of networked systems signifies that, software is more vulnerable in those networked systems, particularly due to poor defense.

b) Extensibility of Software

An extensible host will accept updates or extensions, generally known as mobile code, to ensure the evolution of the efficiency of the system. Regrettably, the extensible systems have the negative side as well, because it tough to defend against software vulnerabilities, which disguise as an undesirable extension.

2.10.3. Security Requirements Engineering

According to Whitman and Mattord, (2003) Security Requirements Engineering (SRE) process is one of the most disregarded factors of a SDLC. Among the most important factors for this negligence is that, conventional requirements engineering methods tend not to meet security needs. There are a minimum two reasons, which make the SRE do not assist, or poorly assist, the needs of security; initially, security requirements are generally not intended for basic investigation and modeling, and the absence of competence from developers, to develop secured software.

With regards to security, the concept behind RE is pretty new; according to Codesecurely, (2006) suggests, one of the most ignored parts of a security enhanced SDLC is the SRE process. Of late, the concept of contemplating security at the early stages of software development has become preferred, because of the inadequacy of conventional requirements engineering (Firesmith, 2007); therefore it is considered that, SRE can offer excellent support for making sure that, security is included in a system in contrast to "bootstrapped" to it down the line (Lewis, 2002).

As mentioned earlier, probably the one of most disregarded factors of a SDLC is the security requirements process (Araujo, 2007). It is deemed that at the requirements phase, security needs to be a vital concept for realizing, not just how to safeguarded a software, but what have to be performed as a way to make sure the satisfaction of clients with the end-result (Richardson, 2010). The majority of the studies have revealed a wide range of actions associated with SRE, ranging from elicitation to verification, and maintenance of software requirements; therefore, it is considered that these routines can be particularly employed to examine SRE.

2.11 Methods and Best Practices in Security Requirements Engineering

Due to the progressive development of functional approaches to SRE, and the recognition of systems to encourage organizational use, project managers can perform superior job of making certain that, the final product proficiently fulfills security requirements. The selection of these strategies depends not only on their reputations, but also on their effectiveness in eliciting security requirements, which include

a. Security Quality requirements Engineering (SQUARE) and
b. Multilateral Security Requirements Analysis (MSRA)
c. Comprehensive, lightweight Application Security Process (CLASP)
d. Unified Modeling Language (UML)

In the following subsection, further detail on the various strategies will be presented.

2.11.1. Security Quality Requirements Engineering (SQUARE)

SQUARE is regarded by Mead et al. (2005) as an extensive technique for SRE for the purpose of involving SRE into the procedures program development (Mead et al., 2008). SQUARE demands effectiveness in genuine software development projects, and thus presents an organizational framework for accomplishing the pursuits of SRE. It is considered that, SQUARE is mutually executed by requirements engineers and stakeholders.

Following its preliminary development, SQUARE has been employed in a collection of customer case studies. The outcomes of case studies were published in a number of studies (Chen et al., 2004; Xie et al., 2004; Gordon et al., 2005). Furthermore, model tools were also formulated to facilitate the process, which comprises the following nine steps:

a. *Agree on definitions:* This level enables apparent connections between requirements experts and stakeholders.
b. *Identify security goals:* In the beginning, the stakeholders will declare several security objectives, and those objectives are structured in this step, and as well as the disputes are fixed.

c. *Develop artifacts*: The following relics are to be gathered: system architecture diagram, use case scenarios/diagrams, misuse case scenarios/diagrams, attack trees, and standardized templates and forms. These items constitute the foundation for the next phases of the method.

d. *Perform risk assessment:* In this step, the weaknesses and risks, relevant to the system are determined, in addition to the chances that the risks will result in attacks. The authors have suggested implementing current risk evaluation techniques.

e. *Select elicitation technique*: The method that has been chosen in this phase might be utilized in the subsequent step to implement the real security requirements elicitation. Just as before, SQUARE indicates to employ current strategy being selected for the specific project.

f. *Elicit security requirements*: An essential factor in this phase is to make sure that the requirements are proven, and that they do not have implementation or design restrictions, rather than requirements.

g. *Categorize requirements*: The elicited specifications are classified (not less than) in accordance with the following conditions: important, unimportant, system-level, software-level, architectural constraint restrictions. Considering that the latter are not regarded as requirements, their presence signifies that the past steps should be implemented again.

h. *Prioritize requirements*: It is presumed that, only a few requirements can be applied; consequently, the most essential requirements have to be determined.

i. *Requirements inspection*: In this final phase, the requirements are analyzed for vagueness, variances, and inappropriate sense. The outcome of this phase is the ultimate security requirements information for the stakeholders.

Table 6 illustrates the revised and base lined draft process after completing case studies. Primarily, steps a to d are basically actions that predate SRE, however, they are essential to make certain that it is productive. Brief explanations are presented below (Mead et al., 2005).

TABLE 6: SQUARE Process

Step	Input	Techniques	Participants	Output
Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements engineer	Agreed-to definitions
Identify assets and security goals	Definitions, candidate goals, business drivers, policies and procedures, examples	Facilitated work session, surveys, interviews	Stakeholders, requirements engineer	Assets and goals
Develop artifacts to support security requirements definition	Potential artifacts (e.g., scenarios, misuse cases, templates, forms)	Work session	Requirements engineer	Needed artifacts: scenarios, misuse cases, models, templates, forms
Perform risk assessment	Misuse cases, scenarios, security goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including threat analysis	Requirements engineer, risk expert, stakeholders	Risk assessment results
Select elicitation techniques	Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of security needed, cost/benefit analysis, etc.	Work session	Requirements engineer	Selected elicitation techniques
Elicit security requirements	Artifacts, risk assessment results, selected techniques	Joint Application Development (JAD), interviews, surveys, model-based analysis, checklists, lists of reusable requirements types, document reviews	Stakeholders facilitated by requirements engineer	Initial cut at security requirements
Categorize requirements as to level (system, software, etc.) and whether they are requirements or other kinds of constraints	Initial requirements, architecture	Work session using a standard set of categories	Requirements engineer, other specialists as needed	Categorized requirements
Prioritize requirements	Categorized requirements and risk assessment results	Prioritization methods such as Analytical Hierarchy Process (AHP), Triage, Win-Win	Stakeholders facilitated by requirements engineer	Prioritized requirements
Inspect requirements	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan, peer reviews	Inspection team	Initial selected requirements, documentation of decision-making process and rationale

Source: Mead et al., 2005

SQUARE is extensive security requirements engineering technique, which advocates taking advantages of other approaches designed in the area, including the objectives of CIA. Each step of SQUARE ends with quite a number of exit conditions, which have to be satisfied prior to the commencement of the subsequent phase. Additionally, the final phase is specifically devoted to validate the specifications. Nevertheless, no official authorization has been conducted.

2.11.2. Multilateral Security Requirements Analysis (MSRA)

The purpose of the MSRA method is usually to utilize the concepts of multilateral security (Rannenberg et al., 1999; Gu¨rses & Santen, 2006; Gu¨rses et al., 2006) throughout the RE stage in systems development. Nevertheless, this is performed by: (i) assessing the requirements of security and privacy of the stakeholders of a system to be developed, (ii) figuring out issues, and (iii) combining the opinions of several stakeholders. The technique gets both, from hypotheses on multilateral security, and viewpoint-oriented requirements engineering. For the purpose of describing various security requirements of the stakeholders, the users MSRA intricate security requirements from the viewpoints of the various stakeholders, concerning incorporated features of a system. Balancing of the objectives of multilateral security initiates the security requirements. The objectives of security are determined by abundant classifications, extracted from the triad of CIA, including, attributes such as, responsibility and facades etc. The objectives of security, and later specifications, consist of the characteristics of stakeholders, who show interest in the requirement, counter-stakeholders towards whom a requirement is expressed, and several other features that are outlined in the following sections (Gu¨rses & Santen, 2006).

A stakeholder is defined as a person or an organization, which has an interest in the proposed system (Rannenberg et al., 1999; Gu¨rses & Santen, 2006; Gu¨rses et al., 2006). Consequently, the illustration of the security requirements is not constrained to the functional users of the proposed system, the latter being described as actors. Instead, a difference is made, which enables the illustration of both, the stakeholders and the end users.

According to Gu¨rses et al. (2005) the variant Confidentiality Requirements Elicitation and Engineering (CREE) of MSRA, focuses only on the requirements of confidentiality. Few other studies have centered on the description of the confidentiality requirements in CREE, and the utilization of ineffective logic to evaluate vagueness and problems (Onabajo & Weber, 2008).

On the other hand, counter-stakeholders are those, to whom security goals are focused on, nevertheless, they might or might not be detrimental assailants, or actors of the system. Additionally, MSRA works with an information model and its components are the objects of the various security requirements. The information model has higher level of contemplation than a data model, which would be essential for a functional specification of the future system. Moreover, the supplemental features of a security requirements are as follows: (i) the user of the security requirements; (ii) the level of understanding among stakeholders, in terms of the security requirements; (iii) the objective of the requirement (however in case of CREE this is mere confidentiality or consent); (iv) the information related to requirement; (v) the sternness, which expresses if the security requirements constitutes a affirmation about the security of information, which it is not clearly addressed; and (vi) the basis, which expresses the need of securing information.

Furthermore, short-term applicability is considered as a feature, which defines how long the security aspects must be maintained. The instance includes functionality of system, which associates with identical security pursuits of a single or a group of stakeholders. Nevertheless, they are beneficial to recognize issues among the objectives of security. The following are various types of conflicts security goals which can exist: (i) for a single stakeholder, among the exclusive requirements she/he has towards numerous attacks; (ii) among the several stakeholders of an instance; and (iii) among the requirements of instances, irrespective of stakeholders (Gu¨rses et al., 2006).

As soon as the issues and disparities are resolved, the objectives of security are enhanced as security requirements. Moreover, additional disputes might exist among, security requirements, functional requirements, and other non-functional requirements. The approach implies to cautiously examine issues and solve them, either during requirements analysis, through design, or using arbitration systems at runtime (Rannenberg et al., 1999).

The following are the primary techniques of the multilateral security requirements analysis method, as soon as the initial functional requirements analysis for the main

functionalities of the system is determined (Rannenberg et al., 1999; Gu¨rses & Santen, 2006; Gu¨rses et al., 2006):

a. *Identify stakeholders*: Stakeholders are those who have, practical, safety, solitude, or information interests in the future system.
b. *Identify episodes*: Episodes are identical to cases, however, they have lower granularity, and determining collection of functionalities would be significant to users. Episodes are employed to segment the objectives of security and are eventually beneficial in discovering issues among multiple security goals.
c. *Elaborate security goals*: Determine and illustrate the objectives of security of the various security stakeholders for every episode.
d. *Identify facts and assumptions*: These are the attributes of the setting, which are appropriate for proclaiming security goals.
e. *Refine stakeholder views on episodes*: Detail the views of stakeholder by considering conceptions, presumptions, and the human relationships among episodes.
f. *Reconcile security goals*: Figure out the issues of security goals, compromise among contradicting objectives, and identify a reliable range of security system requirements.
g. *Reconcile security and functional requirements*: Business aspects for security and vice versa, regarding inconsistent practical and security requirements.

Scope: MSRA is incorporated into the specifications research stage, and can be used once the preliminary functional specifications of the system are established. Here, All CIA goals are regarded, despite the fact that, the emphasis on convenience is focused on the objectives of secrecy and reliability. Furthermore, MSRA focuses on multilateral protection, focusing on stakeholder opinions, the conditions of security requirements, and reclamation of inconsistent specifications. MSRA utilizes UML models to get episodes, the information model, along with the running and protection specifications (Rannenberg et al., 1999; Gu¨rses & Santen, 2006; Gu¨rses et al., 2006).

Validation and quality assurance (QA): MSRA lacks precise validation strategies, furthermore, it does not also ensure integrity of security requirements, despite the fact that multilateral security is an endeavor to determine the requirements of security and privacy, in a system for all stakeholders, which is intended to discover requirements, which constitute a massive technical viewpoint that could have been otherwise neglected. In terms of the multi-directional perspective towards security, issues are the main topic for the method. This technique obviously deals with the relationships among security requirements and also between security and functional requirements. Eventually, characterization operates with inadequate logic, and proposes computerized analysis of the requirements for conflicts and vagueness. Non-functional requirements, apart from, security are not regarded. The MSRA is a iterative method, which means that, after accommodation of security goals into security requirements, the communications are estimated to influence the functional requirements of the system, which needs to evaluate the security requirements (Rannenberg et al., 1999; Gu¨rses & Santen, 2006; Gu¨rses et al., 2006).

2.11.3. Comprehensive, Lightweight Application Security Process (CLASP)

Comprehensive, Lightweight Application Security Process is a unique method for security requirements (John Viega et al., 2005). CLASP is a life cycle approach, which proposes several different activities throughout the SDLC, for the purpose of enhancing security. CLASP is an activity-oriented, role based set of process components, which is guided by formalized best practices (John Viega et al., 2005). CLASP is developed to facilitate software development teams to incorporate security into the stages of present and new SDLC, in an organized and iterative means (buildsecurityin.us-cert.gov, 2012).

The CLASP process is offered by means of five advanced aspects called CLASP Views. These views enable the CLASP users to swiftly comprehend the process of CLASP, including, how CLASP components communicate and how to utilize them in a specific software development life cycle.

a. **Concepts-view**

This view offers an advanced introduction to CLASP, by quickly explaining, the relationship between the five CLASP Views, the seven CLASP best practices, the CLASP classification, the relationship between CLASP to security guidelines, and an example pattern for implementing components of CLASP process.

b. **Role-Based View**

This comprises the role-based introductions to the CLASP process.

c. **Activity-Assessment View**

This facilitates project managers to analyze the suitability of the 24 CLASP pursuits, and decide on one of the subsets. Two sample road maps (i.e., legacy and new-start) are offered by CLASP, to choose relevant pursuits.

d. **Activity-Implementation View**

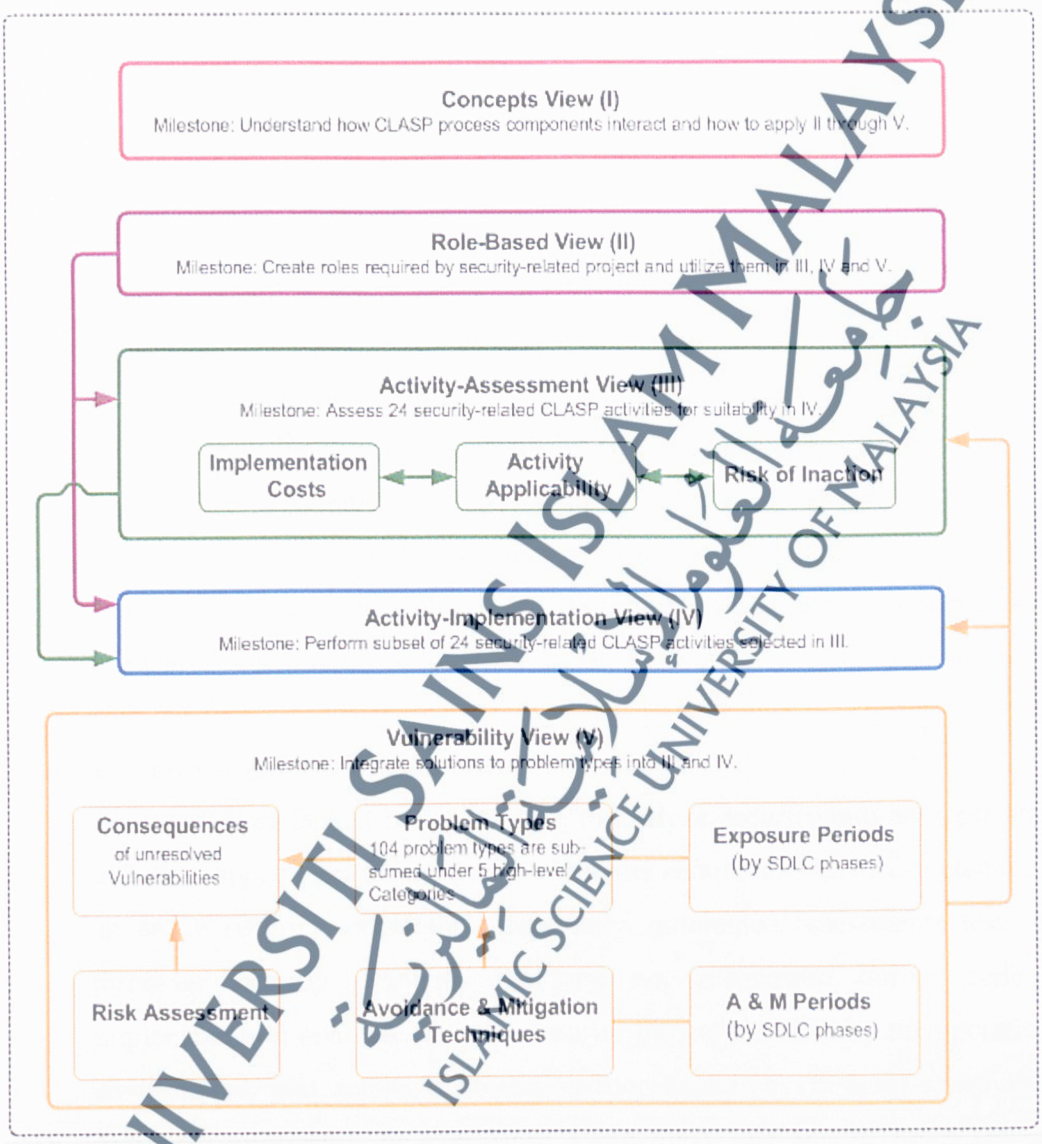
This features the 24 activities related to CLASP security, which can be incorporated into a software development process. The development step of the SDLC transposes any subset of the 24 activities related to CLASP security, which were assessed and accepted in activity assessment, into executable software.

e. **Vulnerability-View**

This view includes a collection of the 104 fundamental types of complications, which were recognized by CLASP based on security weaknesses in application source code. Fundamentally, CLASP classifies all the types of complications into five groups. Nevertheless, a specific type of challenge is generally not considered as security weakness; typically, it is a mixture of complications, which generate a security issue, leading to weaknesses in source code. CLASP Vulnerability Use Cases are affiliated with the vulnerability view, which illustrate conditions in which security services are susceptible to attack, at the application layer. The CLASP users are offered simple an easy examples of

the relationship among security-unaware source coding, and probable ensuing vulnerabilities in basic security services, by means of use cases.

FIGURE 4: CLASP views and their interactions.



Source: buildsecurityin.us-cert.gov, 2012

CLASP Best Practices

The CLASP best practices are the foundation for all software development activities, which are associated with security issues; the software development activities include: planning, designing or implementing, which comprises the utilization of all tools and strategies that assist CLASP in a software development project (Gregoire et al., 2007). The following are the seven CLASP best practices:

a. **Institute Awareness Programs**

Crucial aspects of security, and strategies might be unfamiliar to the in-house software developers and others engaged in software development and implementation. Consequently, at the beginning it is essential to gain knowledge. It is significant that, project managers as the motivators behind most application development, or project enhancement, they will take into consideration the security aspect to be an essential goal of project, both, by training and responsibility. Awareness programs can be conveniently integrated, making use of external specialists if needed, and can provide great results, by ensuring that other activities promote the effective implementation of secured software.

b. **Perform Application Assessments**

As a matter of fact, it is not possible to analyze security into an application, therefore application testing and evaluations should still be a core element of an entire security technique. Especially, automated assessments tests can discover, security problems that are not recognized during code or implementation reviews, discover security threats unveiled by the operational environment, and act as a protection mechanism, by finding downfalls in design, requirements, or execution. Typically, a test analyst holds test and assessment functions, or by the QA organization, however, can extend to the complete life cycle.

c. **Capture Security Requirements**

Guarantee that, security requirements has the identical degree of ownership, as all other prerequisites. Nevertheless, application architects and project

managers can effortlessly focus on functionality, when determining requirements, as they assist the more significant objective of the software, to provide worth to the organization. Security concerns can easily track, so it is vital that, security requirements be a specific part of any application development effort. The following are the factors to be considered:

i. Realizing how applications will be employed, and how they could possibly be misused or infected.
ii. The resources (data and services) accessed or provided by the software, and the level of protection is suitable to address the risks of any organization, restrictions to which it is subjected, and the prospective effect on its reputation should an application be exploited.
iii. The application design and potential attack patterns.
iv. Potential preventive controls and their value and efficiency.

d. Implement Secure Development Practices

A vital objective of software security is to generate and sustain multiple-use source code, which fortifies the fundamental security services in software and over an organization's applications. This objective is most effectively accomplished, by applying secure development practices into an organization's general development process as soon as possible in the SDLC.

This CLASP Best Practice provides a comprehensive group of process elements. It offers well-described, role-based activities, which guide project teams when applying security concepts to design; incorporate security evaluation into the source management process, and apply resource procedures and security technologies. It also offers comprehensive sample coding guidelines, and relics like the 104 CLASP Problem Types, which methodically assist a project team to avoid security weaknesses in source code.

e. **Build Vulnerability Remediation Procedures**

It is specifically significant in the perspective of program up-dates and improvements to determine, which actions will be employed to recognize, evaluate, focus on, and resolve weaknesses. Establishing resolving techniques will accelerate response and reduce risks by interpreting tasks, obligations, and procedures, to adhere, after recognition of weaknesses. Removal techniques are often fed by application tests, both in in-house or third party, and help to manage information when disclosure happens.

f. **Define and Monitor Metrics**

A project development team will not be able to deal with what it cannot determine. However, applying efficient analytics monitoring attempt can be a challenging aspect. In spite of this, metrics are important factors of a general software security attempt. They are essential in determining the present security stance of an organization, concentrate on the most crucial weaknesses, and expose how effectively or improperly the investments in improved security of an organization are performing.

g. **Publish Operational Security Guidelines**

Security does not conclude when a program is finished and implemented in a development environment. This makes the majority of current network and functional protection investment strategies needs that those given the job of tracking and handling the protection of the systems are advised and well-informed; they need guidance and support on the security requirements and how to effectively employ the features built into the application.

The 24 CLASP Activities

CLASP enables effortless incorporation of its activities related to security, into present application development processes. Every single CLASP activity is portioned into distinctive process elements, and associated with one or more exclusive project functions. By doing this, CLASP offers guidance to project team (e.g., project managers, security auditors, developers, architects, testers, and others), which is

convenient to their way of performing. This results in step-by-step enhancements to security, which is readily feasible, and measurable (Gregoire et al., 2007).

It is noteworthy that, the role of project security auditor is developed by CLASP (Gregoire et al., 2007). Primarily this role investigates the present state of a project and tries to ensure the security of the current state of the project in the following phases of project: requirements, design, and implementation. Table 7 lists the 24 CLASP Activities and their related project roles and shows the recommended distribution of these activities across the CLASP Best Practices.

TABLE 7: CLASP activities, related project roles, and best practices

CLASP Best Practices	CLASP Activities	Related Project Roles
a. Institute awareness programs	Institute security awareness program	• Project manager
b. Perform application assessments	Perform security analysis of system requirements and design (threat modeling)	• Security auditor
	Perform source-level security review	• Owner: security auditor • Key contributor: implementer, designer
	Identify, implement, and perform security tests	• Test analyst
	Verify security attributes of resources	• Tester
	Research and assess security posture of technology solutions	• Owner: designer • Key contributor: component vendor
c. Capture security requirements	Identify global security policy	• Requirements specifier
	Identify resources and trust boundaries	• Owner: architect • Key contributor: requirements specifier
	Identify user roles and resource capabilities	• Owner: architect • Key contributor: requirements specifier
	Specify operational environment	• Owner: requirements specifier • Key contributor: architect
	Detail misuse cases	• Owner: requirements specifier • Key contributor: stakeholder
	Identify attack surface	• Designer
	Document security-relevant requirements	• Owner: requirements specifier • Key contributor: architect

d. Implement secure development practices	Apply security principles to design	• Designer
	Annotate class designs with security properties	• Designer
	Implement and elaborate resource policies and security technologies	• Implementer
	Implement interface contracts	• Implementer
	Integrate security analysis into source management process	• Integrator
	Perform code signing	• Integrator
e. Build vulnerability remediation procedures	Manage security issue disclosure process	• Owner: project manager • Key contributor: designer
	Address reported security issues	• Owner: designer • Fault reporter
f. Define and monitor metrics	Monitor security metrics	• Project manager
g. Publish operational security guidelines	Specify database security configuration	• Database designer
	Build operational security guide	• Owner: integrator • Key contributor: designer, architect, implementer

CLASP Resources (including CLASP Keywords)

The essential aspect of the CLASP procedure resources, helps the CLASP activities such as, planning, implementing, and performing. For example, performing the sample coding guideline worksheets resource can assist project managers to recognize, which of the 104 CLASP problem types could create security threats, while developing a specific software system. Consequently, this understanding could be applied to identify how CLASP activities should be performed. A number of CLASP resources are beneficial for projects, which use tools to help automate CLASP process components.

CLASP depends on substantial fieldwork by secure software employees, where the system resources of numerous development life cycles were divided into generate a

complete set of security requirements. These ensuing requirements form the basis of CLASP's Best Practices, which can help organizations to methodically handle any probable weaknesses, and can result in the failure of primary security services (e.g., confidentiality, authentication, and authorization).

Validation and quality assurance (QA): In CLASP it is feasible to implement conventional verification strategies. For this reason, a suitable tool is offered. CLASP contemplates elicitation of security requirements. An essential part of the CLASP process is resources, which assists the planning, implementing, and performing CLASP activities.

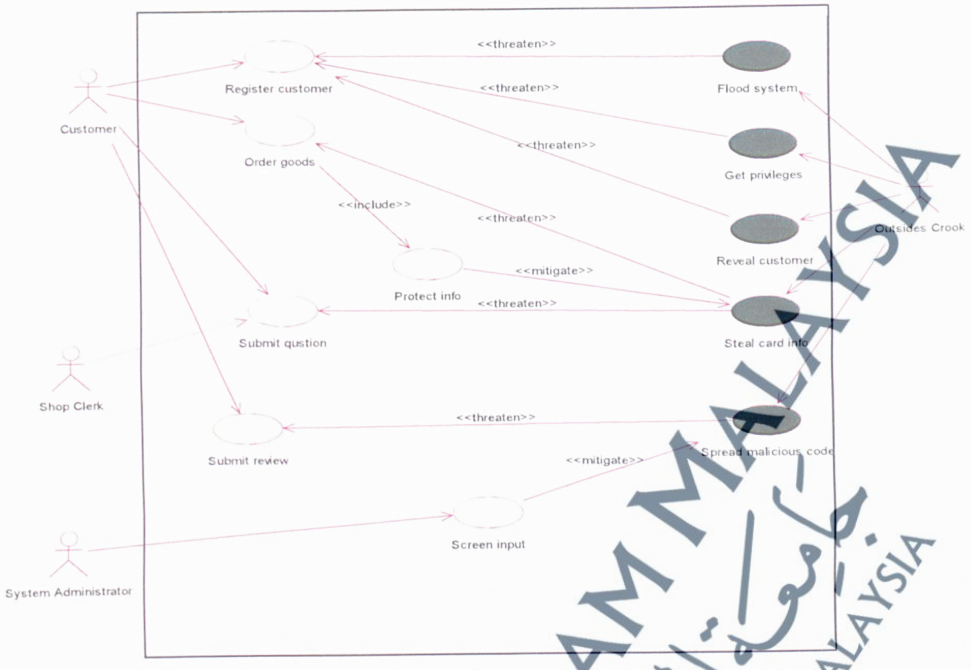
2.11.4. Unified Modeling Language (UML)-Based Approaches

In this section, the approaches to SRE, which employs the UML (Santen & Schmidt, 2010; UML Revision Task Force, 2012) notation shall be discussed because it cares about security issues during software industry.

a. Misuse Cases

Sindre and Opdahl (2001) have expanded the conventional use case method to contemplate misuse cases, which signify unwanted behavior in the system to be developed. Misuse cases are opened up by intruders. A use case diagram (See Figure 5) contains both, use cases and actors (referred as named ellipses and named stick figures, respectively), in addition to misuse cases and abusers (referred as graphically inverted use cases and actors).

FIGURE 5: Use case diagram containing misusers and misuse cases.



Source: Sindr & Opdahl, 2001

A use case is associated with misuse case, making use of a guided organization. An organization directed from a misuse-case to a use case, has the typecast "endanger". A use case diagram can include security use cases, which are exclusive use-cases. A relationship directing from a security-use-case to a misuse-case has the typecast "minimize". It is known that, standard use cases signify requirements, whereas, security-cases signify security requirements, and misuse-cases signify security threats.

Due to the fact that, use case diagrams merely give a summary to the system functionality, the basis of the enclosed uses cases is grabbed in an affiliated textual explanation, which is based on a format to be completed by an analyst. Sindre and Opdahl (2001) have extended, the template, which makes it ideal with outlining misuse-cases, assisting comprehensive elicitation, and evaluation of security threats. In addition, Sindre and Opdahl have presented a repetitive technique, depending on general risk and threat analysis:

i. Establish significant features in the system.
ii. Determine security objectives for every single feature.

iii. Recognize threats for every security objective, by distinguishing stakeholders, which might deliberately damage the system or its environment. Recognize patterns of activities, which might result in deliberate damage.

iv. Recognize and examine risks for the threats (making use of conventional strategies).

v. Determine security requirements for the threats, to coordinate risks and protection costs. Implementing misuse-cases results in a use-case diagram, including use cases, security uses cases, and misuse cases. The approach neither points to a conventional basis nor an attacker model.

Misuse-cases are appropriate to design a system, which includes distinct security needs. It is feasible to contemplate all three CIA goals. It includes prevalent risk and risk analysis strategies. The relationship between practical and security needs is considered in terms of associated use cases and security use cases, in a use case diagram. The method does not contemplate elicitation requirements, integrity of requirements, acceptance, confirmation, inconsistent requirements, nor the relationship of security and other non-functional requirements.

b. Secure-Unified Modeling Language (UML)

Lodderstedt et al. (2002) have presented a UML-based modeling language known as Secure-UML, which facilitates the development of secured, distributed systems. Especially, their method aims at implanting role-based access control policies in UML class diagrams, by employing a UML profile. The UML profile identifies a terminology for annotating class diagrams, with appropriate accessibility control information. However, Secure-UML does not contemplate an attacker model.

Lodderstedt et al. (2002) have focused on designing role-based access control policies, which is relatively a partial mechanism to accomplish the goals of confidentiality and integrity. However, availability is not included in this method. Secure-UML does not contemplate requirements (in terms of security requirements in the theoretical framework) elicitation, completeness of the set of requirements, validation or verification, nor interaction and conflicts of requirements.

c. Unified Modeling Language (UML) sec

Ju¨rjens (2003) has proposed a UML-based modeling language known as named UMLsec, which facilitates the development of security-critical systems. This method considers a number of security requirements, based on the CIA triad. These requirements are represented in various UML diagrams by using generalizations, limitations, and tagged values, which are described in a UML profile. The UMLsec extensions are specifically identified and have a conventional semantics. The study above had considered an attacker model, depending on the adversary tag. It also views domain knowledge concerning assumptions. Ju¨rjens' strategy concentrates on the design of a device, and it includes all the three objectives of CIA.

Ju¨rjens have stated that, the conventional groundwork allows them to implement typical verification strategies. For this reason, a tool is provided. However, the UMLsec does not contemplate elicitation of requirements, completeness of the set of requirements, verification, conflicting requirements, nor possible interaction of security, functional, and other non-functional requirements.

2.11.5. Comparison between Methods and Best Practices in Security Requirements Engineering

This comparison provides dissimilarities among the methods, and best practices, these approaches have some unique features. Table 8 summarizes some major approaches related to SRE, associated with tasks recommended, included in the requirements phase.

TABLE 8: Comparison between methods and best practices in security requirements engineering

Approaches and Best Practices	Year	Focusing on security requirement level	Level of focusing	Consider elicitation of requirements	Confidentiality, Integrity and availability (CIA)	Stakeholder participation	Oriented toward system
SQUARE	2005	Yes	Requirements Elicitation	Yes	CIA	Client	System
MSRA	2006	Yes	Analysis	Yes	CIA	Actor	System
CLASP	2005	Yes	SDLC	Yes	C	Client	System
Misuse cases	2001	Yes	Analysis and Design	No	CIA	Actor	Machine
Secure-UML	2002	No	Design	No	CI	User	Machine
UMLsec	2003	No	Design	No	CIA	Actor	Machine

Table 8 has illustrated the approaches, which have generally recommend misuse or threat identification. Most of the methods used for SRE also support this task. Many of the methods had also recommended identifying normal/user requirements. Therefore, in general, three tasks (CIA) are identified as being highly important to SRE, despite the fact that, the recommendations of the approaches might differ; the second impression is, most of the approaches and methods do not consider starting the security process from the roots, especially in their tasks requirements phase, but at the same time consider security requirements in different phases of SDLC like analysis and design phase.

2.12 Quantifying Security Requirements in Software

In the course of secure software developments, it is vital for the designer to understand, the weaknesses present in each and every phase, and the degree of destruction that these weaknesses can cause to different features of the software; furthermore they also have to realize the security requirements that have to be incorporated, to address these vulnerabilities, and what is the cost effectiveness of each security requirements. Nevertheless there is a lack of studies in this domain

(Khan, 2008; Khan, 2011; Savola et al., 2012). The information measured in terms of vulnerabilities is significant, because if a problem leading to vulnerability is not fixed in an early phase, the cost of solving might raise tenfold with every additional development phase (Khan, 2008; Hanny, 2010). Table 9 illustrates some of the studies related to quantifying security in software industry through SDLC.

TABLE 9: Some studies about quantifying security in software

Author/year	Problem	Objectives	Method	Result
(Khan, 2008)	There is no reliable concrete technique to quantify security of an SDLC artifact (requirements specification, design document, and source code)	-Propose a methodology, which will enable developers to evaluate the security state of a particular SDLC artifact -To perform this evaluation in an additional SDLC phase "security assessment" after requirements, design, and implementation phases. -Prioritizing vulnerabilities based on the potential damage they can cause	Quantification Methodology using math	This would allow developing more secure software and particularly, prioritizing vulnerabilities based on the potential damage they can cause
(Uusitalo et al., 2009)	It is difficult to state whether a certain software product is developed securely enough	Discuss heuristics for evaluating security assurance methods used during the SDLC, and how these evaluations could be transferred to evaluating the whole system	Used the math	Presents security assurance evaluations heuristics that are the first step towards our security assurance evaluation tool. The heuristics take the security assurance methods used in each phase of SDLC into account. They are meant to give guidelines on the trustworthiness of the SDLC
(Alkussayer, 2010)	Able to assess the security of software under development at an early stage (e.g., the design stage)	-Reducing the probability that flaws will be introduced and ensuring that stakeholder requirements have been met. -Focusing on a stage where changes will cost just a fraction of what they would cost in later stages (e.g. implementation)	-Development of a systematic security evaluation framework that aids in assessing the level of security supported by a given architecture and provides the ability to qualitatively compare multiple architectures with respect to their security support -Use math	Present a systematic process for generating a security scenario template that simplifies the assessment process

(Tang, 2010)	Risk assessment is the core and foundation of software project. risk management, directly affecting other processes and even the success or failure of the software projects	Provide a new way to effectively reduce the risk probability and increase the rate of the success of the software development	Using Fuzzy theory	Comes up with a new model of software project risk assessment
(Futcher (b), 2011)	These interconnected computers and networks can be attacked at various points, putting the associated information at risk. A substantial portion of these attacks on systems occur through exploiting vulnerabilities in the software that forms an integral part of the system. This raises the question of 'Why do these vulnerabilities exist in software?'	Address some key aspects related to the security and trustworthiness of a software application functioning within a specific environment	Give key aspects of secure trusted software	By considering these key aspects, a higher level of security and trust could be provided for all stakeholders including the information owners, software developers and users of the software

As highlighted in the above table 9, there is a lack of works related to quantifying the security in software industry. Nevertheless, these studies have reviewed the situation of security in the SDLC, and have focused on the quantifying information about security. However, the problem is how to evaluate the software's security (to check whether the product is developed with adequate security) and how to quantify security of SDLC artifacts. This can be addressed by proposing a framework, methodology or a model. It is noteworthy that, none of these studies have focused on the most important phase, which is requirements phase that is the first phase of SDLC, where all business requirements and security requirements have to be defined, related to the business requirements. Moreover, none of these researches have used the best practices in SRE, which is the core of SRE process.

2.13 Requirements Elicitation Technique for Security Requirement

Donald Firesmith (2003) has claimed that most of requirements engineers are not adequately skilled to elicit, evaluate, and identify security requirements. Subsequently, the requirements engineers generally confound security requirements with architectural security components, which are typically employed to accomplish requirements, and result in making architecture and design decisions. Haley et al., (2007) have acknowledged the same problem; they have demonstrated that, a number of standards (such as the Common Criteria and the US National Institute of Standards and Technology computer security handbook4, which identify security requirements regarding security mechanisms). Nevertheless, they have stated, "Defining requirements in terms of function leaves out key information: what objects need protecting and, more importantly, why the objects need protecting" (p, 2).

Haley et al. (2007) have defined security requirements as "constraints on the functions that operationalize one or more security goals." The study has considered this issue with those, who establish security requirements as high-level security goals; they have argued that, this makes it challenging to make the requirements distinct to facilitate designers, and to validate that the requirements are met. Similarly, they have also advocated security requirements that "express what is to happen in a given situation, as opposed to what is not ever to happen in any situation" (Haley et al., 2007).

The CLASP states that all requirements should be smart requirements, such as precise, considerable, acceptable, realistic, and traceable. However, CLASP provides no examples. On the other hand, D.G. Firesmith (2003) has given such examples; he has defined a security requirements as "a detailed requirement that implements an overriding security policy." He has suggested to partition security requirements into categories, such as, recognition, reliability, and privacy requirements. For example, the requirement "The application shall identify all of its client applications before allowing them to use its capabilities" is an identification requirement, whereas, "The application shall not allow unauthorized individuals or programs access to any communications" is a privacy requirement.

Haley and his colleagues also give examples of security requirements, such as "The system shall provide personnel information only to members of human resources department" (Haley et al., 2007). By revealing security requirements related to precise functional requirements, they have claimed that, they can accomplish adequate uniqueness to help designers, and enable them validate that, the requirements are essentially satisfied. The SE Institute's SQUARE has nine main steps as mentioned in earlier section (Mead et al., 2005): Agree on definitions, Identify security goals. Develop artifacts. Perform risk assessment. Select an elicitation technique. Elicit security requirements. Categorize requirements. Prioritize requirements. In addition, inspect requirements.

SQUARE depends on the relationship between requirements engineers and the stakeholders of an IT project, where facilitation by a RE team is of major importance. Despite the fact that SQUARE is expected for the early phases of software development, step 3 in specific needs some previous design activity. This is because, the artifacts that are to be formulated or determined, involving system architecture diagrams. However, it is not easy to understand what SQUARE includes and does not include, because developers can select a number of approaches for the various stages. For example, the main SQUARE technique does not incorporate at assisting the identification; however, one case study utilizes survivable system analysis (Mead et al., 2005), which contains identification of essential assets and services. Haley et al., (2007) have proposed a framework, which comprises four main steps: Identify

functional requirements. Identify security goals including assets, threats, management principles, and business goals. Identify security requirements. In addition, verify the system.

The authors have suggested problem diagrams according to Jon Hall et.al, (2005) works, they have also recommended that, verification involve, formal debate, the primary technique would probably work with more informal techniques. Iteration between requirement and design activities is an essential part of the framework. Gratifying a security requirement might cause new assets, ensuing in new security requirements.

Boström et al. (2006) have considered security RE in a distinct framework agile development, with a focus on Extreme Programming (XP) practices. They have proposed the following steps for distinguishing and dealing with security requirements Determine significant features. Create experiences of attackers (that is, brief, informal descriptions of how an attacker might abuse the system). Evaluate risk story of attackers. Work out attacker and user stories. Determine security-related user stories. Establish security-related coding standards. In addition, crosscheck abuser stories and countermeasures.

This process expands XP user stories, to involve security requirements. The primary ideas ought to be appropriate for other types of development processes. Moreover, CLASP is a significant effort for securing SDLC. It describes a range of process items that one can incorporate into any software development process. Due to the general nature of CLASP, the steps defined are not definite, for example, Boström et.al, (2006). The requirements phase of the Clasp green-field roadmap has two steps that is, steps recommended for new software development:

a. Document security-relevant requirements (for example, recognizing business requirements and functional security requirements and dependencies) in order to establish risk avoidance and solving inadequacies and conflicts.
b. Identifying resources and limitations of trust, such as, network-level design and data resources.

A number of studies have suggested use-cases as a beginning for determining security requirements. Despite the fact that, the green-field roadmap does not include the task "detail misuse cases," CLASP describes this task as a possible requirements activity. Eduardo Fernandez (2004) has indicated that, use cases are beneficial for identifying the privileges that each actor requires, and for contemplating feasible attacks. Gunnar Peterson (2004) has suggested use and misuse cases as a foundation for security requirements, however has stated that, one might require additional non-functional requirements (Peterson, 2004). Kenneth van Wyk and Gary McGraw (2005) have suggested using abuse cases. Abuse cases are also one of McGraw's "touchpoints" for the requirements and use cases phase, together with security requirements (McGraw, 2006). However, McGraw gives little detail on this security requirements touchpoint.

Lipner and Howard (2005) have described the Microsoft Trustworthy Computing Security Development Lifecycle, by concentrating on planning security activities, which might occur later in the SDLC. They have also stated that, one should determine essential security objectives collectively with security feature requirements, which are based on the needs of clients and conformity with standards. One will determine other security feature requirements, as part of threat modeling, which occurs during design. Even though Threat modeling is a part of the design phase, it also has some steps, which might be considered for the requirements phase (Torr, 2005). Apvrille and Pourzandi, (2005) have suggested four steps for the security requirements and analysis phase:

a. Identify the security environment and objectives.
b. Determine the threat model.
c. Choose security policy, which includes prioritizing according to the information's sensitivity.
d. Evaluate risk.

Apvrille and Pourzandi have intended that the developers accomplish these steps, however, they have not explained the steps in much depth. Due to the fact that the aim is to stress on a lightweight technique; nevertheless some conventional methods have

been deliberately omitted. However, this also signifies that, Howard Chivers's work on automated risk analysis does not fall into the scope of this study, because it is based on formal modeling (Chivers, 2006). The same goes for Axel van Lamsweerde's work on intentional ant models (van Lamsweerde, 2004).

A number of studies have focused on addressing security concerns at the requirements phase. One work that has examined the usefulness of the Common Criteria has been presented by Shi and Sun (Montenegro et al., 2003). Nevertheless, these efforts are restricted only to examination of the use of the Common Criteria, in offering security confidence for software systems.

Hope et al. have analyzed misuse-cases and abuse-cases in the viewpoint of placing them to work (Pauli & Xu, 2005). They have highlighted the need of handling security in the course of the lifecycle, starting from the RE, where, misuse and abuse cases can be employed to identify security requirements. Nevertheless, the number studies have employed misuse and abuse cases, for indicating security requirements (Schneier, 2000; Sindre & Opdahl, 2000, 2001; Shi and Sun, 2001; Moore et al., 2001; Alexander, 2002, 2003; Kwan and Berry, 2004).

2.14 Exhibiting the Problems of Current Frameworks and Methods Related to Security Requirements

There is no reliable concrete technique or method to quantify security requirements in software industry (SDLC artifact) (Khan, 2008; Khan, 2011; Savola et al., 2012). This review of methods, frameworks, and best practices had revealed that, there is a lack of prevalent agreement on defining security requirements. More particularly, a number of methods do not acknowledge on the degree to which the requirements should express definite security measures. SQUARE needs some design work as background material for security requirements elicitation. Microsoft has considered some of the other approaches' requirement activities, as part of the design phase. CLASP identifies risk avoidance to be a requirements activity, while, others contemplate this as a part of design. One reason for these dissimilarities might be the unique focus on iteration.

Agile development, Boström et al. (2006) have focused, on iterative, unlike what is apparently the case with other approaches, such as SQUARE.

The methods also provide various levels of detail as to how to perform the tasks. For example, CLASP and Apvrille suggestions are more standard, and therefore less definite, as against Boström et al. (2006), who have focused on XP development. These approaches also need various levels of expert knowledge. SQUARE depends on a requirements team, which facilitate the process. Haley et al. (2007) have suggested artifacts, which are most likely too complicated for normal developers. Other approaches, such as those proposed by Boström et al. (2006), Microsoft, and CLASP, are more compact.

The methodical limitations have not failed to be recognized. The different approaches might weigh the activities diversely, some methods are more recognized than others, and the comparison criteria might be substandard. Table 8 has illustrated the approaches, which have generally recommend misuse or threat identification. The majority of the typical artifacts used for SRE also support this task. Many of the methods had also recommended identifying objectives and assets. Therefore, on the surface, three tasks (CIA) are identified as being highly important to SRE, despite the fact that, the recommendations of the approaches might differ; the second impression is, most of the approaches and methods do not consider starting the security process from the roots, especially in their tasks requirements phase. This current work differs from these studies, as a single method is not used; rather a number of methods are adopted to find integrated method, and the requirements phase with security aspects will be covered, using the shared and commonly used tasks among best practices. The mathematical algorithms have not contributed in a wide range of studies to solving security requirements problems, especially fuzzy theory algorithms. Therefore, the next context will address fuzzy soft set theory.

2.15 Fuzzy Soft Set Theory

Fuzzy is information granulation underlies the remarkable human ability to make rational decisions in an environment of imprecision, uncertainty and partial truth (Zadeh, 1996). Its principal constituents are fuzzy, computing, and probabilistic reasoning. Soft computing is likely to play an increasingly important role in many application areas, including computer systems (Lotfi, 1994; Zadeh, 1996).

2.15.1. Fuzzy and Computer Systems

Yet, despite its intrinsic importance, fuzzy information granulation has received scant attention except in the context of fuzzy, in which fuzzy underlies the basic concepts of linguistic variable. In fact, the effectiveness and successes of fuzzy in dealing with real-world problems and computer systems are very large (Zadeh, 1996). One of the successes in science is that of according respectability to what is quantitative, precise, rigorous, and categorically true (fuzzy) (Lotfi, 1994).

Fuzzy logic is an extension of standard Boolean logic (Zadeh, 1989). Fuzzy logic starts with the concept of a fuzzy set. A fuzzy set is a set without a crisp, clearly defined boundary, admitting the possibility of partial membership (Yuan & Khoshgoftaar, 2000). Fuzzy logic has become an important tool for number of different applications ranging from the control of engineering system to artificial intelligence. Practical applications of fuzzy logic pose a unique set of problems. The design of systems, which apply fuzzy logic to make use of human knowledge and experience, is a daunting task without facing engineering problems of real world systems (Sram, 2011). The early works leading to fuzzy logic were by (Togai et.al, 1986; Togai et.al, 1987; Watanabe et.al, 1988; Yamakawa (a), 1988; Yamakawa (b), 1988), it was fuzzy logic chips and fuzzy computer hardware.

2.15.2. Fuzzy

There are some mathematical tools for dealing with uncertainties. Two of these tools are fuzzy set theory, developed by Zadeh (1965), and soft set theory, introduced by

Molodtsov (1999), that are related to this work. At present, work on the soft set theory is progressing rapidly. Maji et al. (2003) defined operations of soft sets to make a detailed theoretical study on the soft sets. By using these definitions, the applications of soft set theory have been studied increasingly. Soft decision making (Chen et al., 2005; Herawan et al., 2009; Herewan & Deris (b), 2009; Feng et al., 2010; Cagman & Enginoglu (a ,b), 2010; Cagman et al. (a ,b), 2010).

By using fuzzy sets, Maji et al. (a), (2001) defined the fuzzy soft set theory and many scholars study the properties and applications of it (Deschrijver, 2007; Ahmad & Kharal, 2009; Aygunoglu & Aygun, 2009; Cagman et al. (a ,b), 2010).

2.15.3. Preliminaries

In this section, will recall some basic concepts of fuzzy sets, soft sets and fuzzy soft sets.

Let $U = \{x , x ,..., xn\}$ be the initial set of universe and E be the set of parameters. Given $A \subseteq E$, and let the power set of U and the set of all fuzzy subset of U be represented as $P(U)$ and $\tilde{P}(U)$ respectively (Sulaiman & Mohamad, 2012).

Definition 1. (Zadeh, 1965) A fuzzy set $\tilde{F}$ on U is the set characterized by the membership function $\mu_{\tilde{F}}: U \rightarrow [0,1]$.

Definition 2. (Molodtsov, 1999) A pair (F, E) is called a soft set over U where F is a mapping given by $F: E \rightarrow P(U)$.

Definition 3. (Maji et al.(a), 2001) A pair $(\tilde{F},A)$ is called a fuzzy soft set over U where $\tilde{F}$ is a mapping given by $\tilde{F}: A \rightarrow \tilde{P}(U)$.

2.15.4. Soft Sets

In this sub section, the definition of the soft set theory will be given. A more detailed theoretical study of this concept is given in (Cagman & Enginoglu (a), 2010).

Definition: Let U be an initial universe, $P(U)$ be the power set of be the set of U, E all parameters and $A \subseteq E$ Then, a soft set F_A over U is a set defined by a function representing a mapping as in equation (1).

$$f_A : E \to P(U) \text{ such that } f_A(x) = \varnothing \text{ if } x \notin A. \ldots\ldots(1)$$

Here, f_A is called approximate function of the soft set F_A , and the value $f_A(x)$ is a set called x-element of the soft set for all $x \in E$. It is worth noting that the sets $f_A(x)$ may be arbitrary. Some of them may be empty, some may have nonempty intersection. Thus, a soft set F_A over U can be represented by the set of ordered pairs as in equation (2).

$$F_A = \{(x, f_A(x)) : x \in E, f_A(x) \in p(U)\} \ldots\ldots(2)$$

Note that the set of all soft sets over U will be denoted by $S(U)$.

2.15.5. Fuzzy Sets

This sub section present the basic definitions of fuzzy set theory (Zadeh, 1965) that is useful for subsequent discussions. More detailed explanations related to this theory may be found in earlier studies (Dubois & Prade, 1980; Klir & Folger, 1988; Zimmermann, 1991). A fuzzy set is a class of objects with a continuum of grades of membership. Such a set is characterized by a membership (characteristic) function which assigns to each object a grade of membership ranging between zero and one.

Definition: Let U be a universe. A fuzzy set X over U is a set defined by a function μ_X representing a mapping as in equation (3).

$$\mu_X : U \to [0,1] \ldots..(3)$$

Here, μ_X called membership function of X, and the value $\mu_X(u)$ is called the grade of membership of $u \in U$. The value represents the degree of u belonging to the fuzzy set X. Thus, a fuzzy set X over U can be represented as in equation (4) as follows,

$$X = \left\{ \left(\mu_X(u) / u \right) : u \in U, \mu_X(u) \in [0,1] \right\} \quad \ldots..(4)$$

Note that the set of all the fuzzy sets over U will be denoted by $F(U)$.

2.15.6. Fuzzy Soft Sets

By using fuzzy sets, Maji et al. (a), (2001) defined fuzzy soft sets (*fs*-sets). In this sub section, by using the definition, given in (Çağman & Enginoglu, 2010a), will redefine the *fs*-sets and their operations. More detailed theoretical study of this concept is given in (Çağman et al. (a), 2010). Fuzzy soft set, a more general concept, which is a combination of fuzzy set and soft set. The concept of fuzzy soft sets as introduced by Maji et al. (2003) to the possibility fuzzy soft set. In our generalization of fuzzy soft set, a possibility of each element in the universe is attached with the parameterization of fuzzy sets while defining a fuzzy soft set.

In sub section 2.14.4 (Soft sets), the approximate function of a soft set is defined from a crisp parameters set to a crisp subsets of universal set. However, the approximate functions of *fs*-sets are defined from crisp parameters set to the fuzzy subsets of universal set. To avoid the confusion will use $\Gamma_A, \Gamma_B, \Gamma_C, \ldots$, etc. for *fs*-sets and $\gamma_A, \gamma_B, \gamma_C, \ldots$, etc. for their fuzzy approximate functions, respectively.

Definition: Let U be an initial universe, E be the set of all parameters, $A \subseteq E$ and $\gamma_A(x)$ be a fuzzy set over U for all $x \in E$. Then, an *fs*-set Γ_A over U is a set defined by a function γ_A representing a mapping in equation (5).

$$\gamma_A : E \to F(U) \text{ such that } \gamma_A(x) = \varnothing \text{ if } x \notin A. \quad \ldots..(5)$$

Here, γ_A is called fuzzy approximate function of the *fs*-set Γ_A, and the value $\gamma_A(x)$ is a fuzzy set called x-element of the *fs*-set for all $x \in E$. Thus, an *fs*-set Γ_A over U can be represented by the set of ordered pairs as in equation (6)

$$\Gamma_A = \{(x, \gamma_A(x)) : x \in E, \gamma_A(x) \in F(U)\} \quad \ldots..(6)$$

Note that from now on the sets of all *fs*-sets over U will be denoted by *FS(U)*.

2.15.7. Fuzzy Parameterized Soft Sets

In this subsection, by using the definition, given in (Çağman & Enginoglu (a), 2010), will define the fuzzy parameterized soft sets (*fps*-sets) and their operations. More detailed theoretical study of this concept is given by Çağman et al. (b), (2010). In sub section 2.13.3 (Soft sets), the approximate function of a soft set is defined from a crisp parameters set to a crisp subsets of universal set. But the approximate functions of *fps*-sets are defined from fuzzy parameters set to the crisp subsets of universal set.

Through this sub section, the fuzzy subsets of parameters set are denoted by the letter $X, Y, Z, \ldots$, to avoid confusion and complexity of the symbols.

Definition: Let U be an initial universe, $P(U)$ be the power set of U, E be the set of all parameters and X be a fuzzy set over E with the membership function $\mu_X : E \to [0,1]$. Then, an *fps*-set F_X over U is a set defined by a function f_X representing a mapping shown in equation (7)

$$f_X : E \to P(U) \text{ such that } f_X(x) = \varnothing \text{ if } \mu_X(x) = 0 \ldots..(7)$$

Here, f_X is called approximate function of the *fps*-set F_X, and the value $f_X(x)$ is a set called x-element of the *fps*-set for all $x \in E$. Thus, an *fps*-set F_X over U can be represented by the set of ordered pairs as in equation (8)

$$F_X = \{(\mu_x(x)/x, f_X(x)) : x \in E, f_X(x) \in P(U), \mu_X(x) \in [0,1]\} \ldots..(8)$$

Note that from now on the sets of all fps -sets over U will be denoted by $FPS(U)$.

2.16 Research Gap

Practically, requirements elicitation phase has been facilitated by a wide range of techniques (Goguen & Linde, 1993). Every single technique includes several advantages, and has helped the developers to understand organizational domains in a wide range of ways. For making the requirements elicitation phase to be more effective and accurate, it is essential for the RE professionals to identify and analyze a number of techniques (Escalona & Koch, 2004). Obviously, each requirements elicitation technique consists of specific constraints or limitations. Moreover, there is a lack of studies related to security requirements elicitation, and there is a little focus on security in the requirements phase in SDLC.

Based on the literature review, there is a lack of studies relevant to quantifying the security in software industry. However, some studies have assessed the security problem in SDLC, and have concentrated on the quantifying information about security. On the other hand, the problem resides in evaluating the software's security (to examine whether the product is developed with sufficient protection) and how to quantify security of SDLC artifacts. This can be resolved by proposing a framework, methodology or a model. It is significant that, till recently none of these studies have concentrated on the most crucial phase, which is requirements phase, that is the first phase of SDLC, where all organizational and security requirements need to be defined. Furthermore, these studies did not employ the best practices in SRE, which is the key aspect of SRE process.

As discussed earlier, fuzzy logic has a wide impact in computer systems, especially in hardware industry, but, there is a lack of utilization or adaptation of the fuzzy soft set (fuzzy theory) to signify a contribution in computer science field, particularly in security of SE, because the option of using fuzzy soft set theory to address a problem in a particular field has its own obstacles, because it is a very appropriate and accurate decision making tool.

2.17 Summary

This chapter discusses and provides the necessary background and literature review about requirements engineering, security requirements and fuzzy theory, its features, and its limitations. It shows the relationship between success of software products and security requirements and it shows a comparison between methods and techniques that are used to elicit and quantify security requirements; the main focus in this chapter is on the CLASP best practices, SQUARE method and AI technique. Illustrate fuzzy soft set theory history and its contribution in computer science field, particularly in security of SE.