Security Measures in Requirement Development Using Defense in Depth

R.Saranya, Madurai Kamaraj University

***Abstract*:** Human utilizes the electronic machine – computer to minimize his work, which is safe and secure. Software engineering plays a major role in the entire upcoming field. Security is the main principles in all the process. Different principles are used to protect and secure the software process. Now-a- days software engineering processes are practiced with principles and techniques for efficient requirement gathering in the requirement phase. Various risks may occur during the development life cycle of software. To manage the risks, use of multiple defensive strategies are employed and the purpose of using the method helps us to protect if one layer of defense turns out to be inadequate, another layer of defense will, ideally, prevent a full breach. This paper deals with the defense in depth method in requirement development of requirement engineering phase to protect the details of a software process.

Key words: software engineering, requirement engineering, defense in depth, requirement development.

1. Introduction

Software engineering (**SE**) is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. In layman's terms, it is the act of using insights to conceive, model and scale a solution to a problem. There are four fundamental phases in most, if not all, software engineering methodologies. These phases are analysis, design, implementation, and testing[1]. These phases address what is to be built, how it will be built, building it, and making it high quality. These phases will now be defined as they apply to the life cycle stage of product delivery emphasized in this thesis.

Even though this thesis emphasizes the four phases of analysis, design, implementation, and testing in a software engineering methodology as it applies to the software life cycle stage of product delivery, the results are also applicable to the other software life cycle stages of deployment, operations, maintenance, legacy, and finally discontinuation as the system transitions through many versions from cradle to death

1.1 Analysis Phase

The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished. This phase defines the problem that the customer is trying to solve. The deliverable result at the end of this phase is a requirement document. Ideally, this document states in a clear and precise fashion what is to be built [2]. This analysis represents the ``what'' phase. The requirement document tries to capture the requirements from the customer's perspective by defining goals and interactions at a level removed from the implementation details. The analysis phase is summarized in Table 1.1

Table 1.1: The Analysis Phase:

Phase	Deliverable
Analysis	● Requirements Document
	● Domain Ontology
	- Things
	- Actions
	- States
	● Typical Scenarios
	● Atypical Scenarios

The **requirement** document may be expressed in a formal language based on mathematical logic[11]. Traditionally, the requirement document is written in English or another written language. The requirement document does not specify the architectural or implementation details, but specifies information at the higher level of description. The problem statement, the customer's expectations, and the criteria for success are examples of high-level descriptions. There is a fuzzy line between high-level descriptions and low-level details.

If an exact engineering detail needs to be specified, this detail will also appear in the requirement document. This is the exception and should not be the rule. These exceptions occur for many reasons including maintaining the consistency with other established systems, availability of particular options, customer's demands, and to establish, at the requirement level, a particular architecture vision. An example of a low-level detail that might appear in the requirement document is the usage of a particular vendor's product line, or the usage of some accepted computer industry standard, or a constraint on the image size of the application.

Things : The requirement document first of all defines the ontology of the system which is, in the more general sense, the noun phrases

Actions : The requirement document defines the actions that the system should perform. This is expressed, in the more general sense, as verb phrases. Methods, functions, and procedures are all examples of actions.

States : States are defined as a sequence of settings and values which distinguishes one time-space slice of a system from another slice. Every state-full system goes through a series of state changes. Example states include the initial state, the final state, and potentially many error states. Most of the states are domain specific.

States are associated with things in the system. An event triggers a potential state transition which may then lead to an action taken by the system.

Typical Scenarios

A **scenario** is a sequence of steps taken to accomplish a given goal. When the system is completed and the application is available, the customer should be able, in an easy and clearly specified manner, to accomplish all typical usage scenarios for the application. The **typical scenarios** should represent the vast majority of uses for the system. The exact coverage of the system by the typical scenarios vary, but a 90 percent coverage is desirable. Obviously, a system with only one possible usage scenario will be easy to cover while a system with thousands of possible usage scenarios will be much harder to cover. Frequently the 80/20 rule is invoked.

Eighty percent of the functionality of a typical system is accomplished by twenty percent of the work. To accomplish the remaining minority functionality requires the vast majority of the work.

Atypical Scenarios

An **atypical scenario** is something that needs to be accomplished within the system, but only seldom. The actions have to be done correctly, but perhaps at lower efficiency. The customer should hope that an unexpected error condition is an atypical event. Nonetheless, the system should be able to deal with many categories of faults by using several established techniques, such as exception handlers, replications, process monitoring, and roll over. Atypical scenarios and typical scenarios share similar coverage.

Incomplete and Non-Monotonic Requirements

An entire enumeration of all of the requirements is not possible for nearly all real-life situations. Godel's incompleteness theorem of arithmetic says that there is no finite list of axioms that completely describe integer arithmetic. Expressed in our terminology, there is no finite list of requirements that would completely describe arithmetic. Since integer arithmetic is an underlying foundation of most computer hardware systems and software applications, and since we can't even enumerate the requirements for integer arithmetic, the task of completely enumerating a more complex system is certainly intractable.

In traditional logic, a theory is defined by a finite set of axioms. Theorems within the theory are valid sentences. If new axioms are added to the theory, the already existing theorems remain valid and the theory is extended into a new theory with new theorems added to the established theorems.

In **non-monotonic** logic, adding new axioms to the theory may invalidate existing theorems that were already proven. A new theory is created which is not a simple extension of the old theory, but a collection of new theorems and some of the established theorems.

The requirement gathering process is iterative in nature and more like non-monotonic logic than monotonic logic. An initial collection of requirements, the axioms of the system, define the

capabilities, the theorems of the system. New requirements may lead to a collection of capabilities different than the established capabilities. New requirements may negate old solutions.

Early in the process, some requirements are established. As the process continues, other requirements are discovered which may be in conflict with earlier known requirements, thus leading a different system. Unfortunately, as a system increases in size and complexity, the requirement gathering process becomes more and more intractable. This is especially true when the requirement gathering process is distributed across many individuals from many different disciplines.

2. Requirement Engineering

"What I *need*, not what I *said* I needed"

Requirement engineering has always occupied a primal position in software engineering. If you get the requirement correct, you are very close to getting the software correct has been a universal fact. Many principles and techniques have been proposed for efficient requirement gathering and these have been validated and applied in practice. Every phase is important unlike requirement engineering phase. Security requirement possess certain unique characteristics that prevent them from being treated with other normal functional requirements. The lack of security in the developed system is not as apparent as failures related to performance, tolerance and reliability.

The sub categories of the requirement engineering has been depicted in Fig 1.

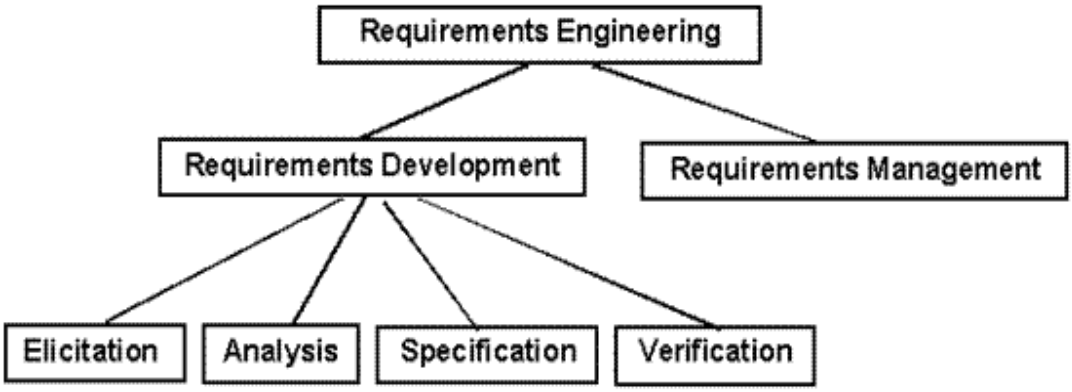

Fig 1. Sub disciplines of Requirement Engineering

2.1 The Core Activities

From the three dimensions of requirements engineering, the three core activities of requirements engineering can be derived. Each core activity significantly contributes to the achievement of one of the three sub-goals of requirements engineering. We explain the three core activities below. The interactions between the core activities, as well as between the core activities and the cross-sectional activities.

2.1.1 Documentation

The focus of the documentation activity is the documentation and specification of the elicited requirements according to the defined documentation and specification rules. In addition, other important types of information such as rationale or decisions must

Three Core Activities be documented. The documentation activity thus distinguishes the following sets of rules:

General documentation rules: These rules apply to all kinds of information to be documented such as interview and meeting protocols, information about the context or decisions and

rationale. These rules define, for instance, the document layout, required document headers, and required document management information such as authors or a version history.

Documentation rules: These rules apply to each requirement documented at different stages of the requirements engineering process. The rules aim to ensure a sufficient quality of the documentation of the requirements mainly for use in other requirements engineering activities (e.g. negotiation or validation) while at the same time keeping the documentation effort low. Documentation rules may, for instance, prescribe a specific template to be used for documenting the requirements.

Specification rules: These rules apply to all requirements which are included in the requirements specification. The specification rules aim to ensure high quality of the specified requirements which are used in the subsequent development activities as key input or might be part of contracts. The specification rules may prescribe, for instance, the use of syntactic requirements patterns or a requirements specification language.

In general, specification rules are typically more restrictive than documentation rules. Depending on the intended use of a requirements artefact, different documentation and specification formats can be used. For example, a requirement might be documented using natural language to facilitate communication with a typical end user, while at the same time be specified using a formal requirements language to support the system architect in defining the system architecture. In general, different stakeholders prefer different documentation/specification formats. Hence a requirements artefact may have to be translated from one format into another one. The issue becomes keeping the documentation/specification of a requirement held in different formats consistent across formats when undertaking any change. The documentation activity is discussed in more detail.

2.1.2 Elicitation

The goal of the elicitation activity is to improve the understanding of the requirements, i.e. to achieve progress in the content dimension. During the elicitation *dimension* activity, requirements are elicited from stakeholders and other requirement sources.In addition, new and innovative requirements are collaboratively developed.The requirement sources relevant for the system are not always known at the beginning of the process. An essential task of the elicitation activity is therefore the systematic identification of relevant requirement sources. Relevant requirement sources include the stakeholders involved in the process, existing documentation,

and existing predecessor systems. Requirements are elicited, for example, by interviewing the stakeholders or by analysing existing documents and systems. In addition, innovative requirements (which can typically not simply be elicited from a requirement source) are developed in a collaborative and creative process. The development of innovative requirements can, for example, be supported by applying creativity, techniques such as brainstorming. The elicitation activity is described in more detail .

2.1.3 Negotiation

The system has to fulfill the needs and wishes of different stakeholders. Obviously, *stakeholder opinions* the needs and wishes of the different stakeholders can vary. Each stakeholder has his/her own view about the system to be developed. The different opinions of the stakeholders can be in conflict with one another. The goal of the negotiation activity is therefore twofold: First, all conflicts between *conflicts* the viewpoints of the different stakeholders have to be detected and made explicit. Second, the identified conflicts should be resolved (as far as possible). Depending on the cause of the conflict, different strategies can be applied for resolving it. At the beginning of the requirements engineering process, typically the viewpoints of the different stakeholders differ significantly. Ideally, at the end of the requirements engineering process, the negotiation activity has identified and resolved all conflicts which exist between the different stakeholders involved.

3. Issues in Requirements Elicitation

There are many problems associated with requirements engineering, including problems in defining the system scope, problems in fostering understanding among the different communities affected by the development of a given system, and problems in dealing with the volatile nature of requirements. These problems may lead to poor requirements and the cancellation of system development, or else the development of a system that is later judged unsatisfactory or unacceptable, has high maintenance costs, or undergoes frequent changes. By improving requirements elicitation, the requirements engineering process can be improved, resulting in enhanced system requirements and potentially a much better system.

Requirements engineering can be decomposed into the activities of requirements elicitation, specification, and validation. Most of the requirements techniques and tools today focus on specification, i.e., the representation of the requirements. This report concentrates instead on elicitation concerns, those problems with requirements engineering that are not adequately addressed by specification techniques. An elicitation methodology is proposed to handle these concerns.

This new elicitation methodology strives to incorporate the advantages of existing elicitation techniques while comprehensively addressing the activities performed during requirements elicitation. These activities include fact-finding, requirements gathering, evaluation and rationalization, prioritization, and integration. Taken by themselves, existing elicitation techniques are lacking in one or more of these areas.

4. Defense in depth

Defense in depth is the coordinated use of multiple security countermeasures to protect the integrity of the information assets in an enterprise. The strategy is based on the military principle that it is more difficult for an enemy to defeat a complex and multi-layered defense system than to penetrate a single barrier.

4.1 Defense in Depth and Requirement Development

Each stage of Requirement development requires verification and to validated in line with documentation.

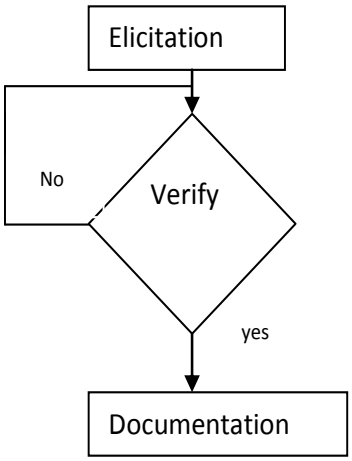

Fig 2. Flowchart for elicitation of requirement development

In requirements engineering, **requirements elicitation** is the practice of collecting the requirements of a system from users, customers and other stakeholders. The practice is also sometimes referred to as **requirements gathering**.

The term elicitation is the fact that good requirements can not just be collected from the customer, as would be indicated by the name requirements gathering. Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do. Requirements elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping[7, 8].

Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation process. Requirements elicitation is a part of the requirements engineering process, usually followed by analysis and specification of the requirements.

Commonly used elicitation processes are the stakeholder meetings or interviews. For example, an important first meeting could be between software engineers and customers where they discuss their perspective of the requirements [12].

The necessary steps to be followed in requirements elicitation is,

- Assess the process and technical feasibility for the proposed system
- Identify the people who will help specify requirements and understand their organizational bias[5]
- Define the technical environment (e.g., computing architecture, operating system, telecommunications needs) into which the system or product will be placed
- Identify "domain constraints" (i.e., characteristics of the business environment specific to the application domain) that limit the functionality or performance of the system or product to be built
- Define one or more requirements elicitation methods (e.g., interviews, focus groups, team meetings)
- Solicit participation from many people so that requirements are defined from different points of view; be sure to identify the rationale for each requirement that is recorded
- Identify ambiguous requirements as candidates for prototyping
- Create usage scenarios or use cases to help customers/users better identify key requirements[6].

Figure 2 shows the flow of elicitation working with relevant to the customer need. This will enhance the quality in the developing product and provide high security in the requirement elicitation[4]. The procedure to verify the gathered requirements,

Step 1: Identify the real problem, opportunity or challenge

Step 2: Identify the current value which show that the problem is real

Step 3: Identify the goal(g) to show the problem has been addressed and the value of meeting it

Step 4: Identify the as-is cause(s) of the problem, as it is the causes that must be solved, not the problem directly

Step 5: Define the business what that must be delivered to meet the goal(g).

Step 6: Specify a product design how to satisfy the real process requirements.

5. Conclusion:

The procedure is applied to the requirement development is discussed in this paper can be applied to various stages. Here, requirement elicitation is considered and the procedure for gathering is proposed, the goals of the product has been assed. Each goal have certain values which will be identified and assed properly.

6. References:

[1] Roger.S.Pressman. Software Engineering Principles and Practice , Sixth Edition, 2002.

[2] *Ronald LeRoi Burback*, Software Engineering Phases,1998.

[3] Alexander, Ian and Beus-Dukic, Ljerka. *Discovering Requirements: How to Specify Products and Services*. John Wiley, 2009.

[4] Goldsmith, Robin F. *Discovering Real Business Requirements for Software Project Success*. Artech House, 2004.

[5] Miller, Roxanne E. *The Quest for Software Requirements*. Mavin Mark Books, 2009.

[6] Sommerville, Ian and Sawyer, Pete. *Requirements Engineering: A Good Practice Guide*. John Wiley, 1997.

[7] Kotonya G. and Sommerville, I. Requirements Engineering: Processes and Techniques. Chichester, UK: John Wiley & Sons

[8] Software Requirements Engineering Methodology (Development) Alfor,M. W. and Lawson,J. T. TRW Defense and Space Systems Group. 1979.

[9] Thayer, R.H., and M. Dorfman (eds.), System and Software Requirements Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[10] Royce, W.W. 'Managing the Development of Large Software Systems: Concepts and Techniques', IEEE Westcon, Los Angeles, CA> pp 1-9, 1970. Reprinted in *ICSE '87, Proceedings of the 9th international conference on Software Engineering*.

[11] Sommerville, I. *Software Engineering*, 7th ed. Harlow, UK: Addison Wesley, 2006.

[12] Ralph, Paul (2012). "The Illusion of Requirements in Software Development". *Requirements Engineering*.