Requirements Elicitation Introduction

Nancy Mead

September 2006

ABSTRACT: Using an elicitation method can help in producing a consistent and complete set of security requirements. However, brainstorming and elicitation methods used for ordinary functional (end-user) requirements usually are not oriented toward security requirements and do not result in a consistent and complete set of security requirements. The resulting system is likely to have fewer security exposures when security requirements are elicited in a systematic way.

In this article we briefly discuss a number of elicitation methods and the kind of tradeoff analysis that can be done to select a suitable one. Companion case studies can be found in Requirements Elicitation Case Studies. While results may vary from one organization to another, the discussion of our selection process and various methods should be of general use. Requirements elicitation is an active research area, and we expect to see advances in this area in the future. We expect that eventually there will be studies measuring which methods are most effective for eliciting security requirements. At present, however, there is little if any data comparing the effectiveness of different methods for eliciting security requirements.

ACKNOWLEDGEMENT: This material is extracted and adapted from a more extensive report by Lydia Chung, Frank Hung, Eric Hough, and Don Ojoko-Adams [Chung 06].

OVERVIEW OF SEVERAL ELICITATION METHODS

The following list is a sample of methods that could be considered for eliciting security requirements. Some have been developed specifically with security in mind (e.g., misuse cases), whereas others have been used for traditional requirements engineering and could potentially be used for security requirements. In the future we may have a better understanding of how the unique aspects of security requirements elicitation drive selection of a method. We also note recent work on requirements elicitation in general that could be considered in developing such a list [Hickey 03, Hickey 04, Zowghi 05] and in doing the selection process [Hickey 04]. We briefly describe each of these elicitation methods:

- misuse cases [Sindre 00, McGraw 06, p. 205–222]

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

- Soft Systems Methodology [Checkland 90]
- Quality Function Deployment [QFD 05]
- Controlled Requirements Expression [Christel 92, SDS 86]
- issue-based information systems [Kunz 70]
- Joint Application Development [Wood 95]
- feature-oriented domain analysis [Kang 90]
- critical discourse analysis [Schiffrin 94]
- Accelerated Requirements Method [Hubbard 00]

Misuse Cases

A use case generally describes behavior that the system owner wants the system to show [Sindre 00]. Use-case models and their associated diagrams (UCDs) have proven quite helpful for the specification of requirements [Jacobson 92, Rumbaugh 94]. However, a collection of use cases should not be used as a substitute for a requirements specification document, as this approach can result in overlooking significant requirements [Anton 01]. As a result, it is controversial to solely use use-case models for system and quality requirements elicitation.

Misuse cases apply the concept of a negative scenario—that is, a situation that the system's owner does not want to occur—in a use-case context. For example, business leaders, military planners, and game players are familiar with analyzing their opponents' best moves as identifiable threats. Misuse cases are also known as abuse cases. A deeper discussion of abuse cases as an approach for identifying security requirements can be found in [McGraw 06].

One significant characteristic of misuse cases is that they seem to lead to quality requirements, such as those for safety and security, whereas other elicitation methods are focused on end-user requirements, so their effectiveness in the identification of security requirements is unknown. Use cases describe system behavior in terms of functional (end-user) requirements. Interplay between misuse cases and use cases could improve the efficiency of eliciting all requirements in a system engineering life cycle. Misuse cases and use cases may be developed from system to subsystem levels—and lower as necessary. Lower level cases may draw attention to underlying problems not considered at higher levels and may compel system engineers to reanalyze the system design. Misuse cases are not a top-down method, but they provide opportunities to investigate and validate the security requirements necessary to accomplish the system's mission.

Soft Systems Methodology (SSM)

SSM deals with problem situations in which there is a high social, political, and human activity component [Checkland 90]. The SSM can deal with "soft problems" that are difficult to define, rather than "hard problems" that are more technology oriented. Examples of soft problems are how to deal with homelessness, how to manage disaster planning, and how to improve Medicare. Eventually technology-oriented problems may emerge from these soft problems, but much more analysis is needed to get to that point. SSM is composed of seven stages:

1. Find out the problem situation.
2. Express the problem situation through rich pictures (i.e., representations of organizational structure and processes pertinent to the problem situation).
3. Select how to view the situation and produce root definitions.
4. Build conceptual models of what the system must do for each root definition.
5. Compare the conceptual models with the real world.
6. Identify feasible and desirable changes.
7. Make recommendations to improve the problem situation [Checkland 90].

The primary benefit of SSM is that it provides structure to soft problem situations and enables their resolution in an organized manner. It compels the developer to discover a solution that goes beyond technology.

Quality Function Deployment (QFD)

QFD is "an overall concept that provides a means of translating customer requirements into the appropriate technical requirements for each stage of product development and production" [QFD 05]. The distinguishing attribute of QFD is the focus on customer needs throughout all product development activities. By using QFD, organizations can promote teamwork, prioritize action items, define clear objectives, and reduce development time [QFD 05].

Although QFD covers a broad portion of the product development life cycle, the earlier stages of the process are applicable to requirements elicitation for software engineering. These stages include

1. identifying the customer (stakeholders)
2. gathering high-level customer requirements
3. constructing a set of system features that can satisfy customer needs
4. creating a matrix to evaluate system features against satisfaction of customer needs

Note that the evaluation of features and needs could also be used for prioritization of requirements, in the context of a QFD requirements elicitation activity.

Controlled Requirements Expression (CORE)

CORE is a requirements analysis and specification method that clarifies the user's view of the services to be supplied by the proposed system and the limitations imposed by that system's operational environment, in conjunction with some degree of performance and reliability investigation [Mullery 79]. CORE provides methods and notations for every phase of "elicitation, specification and analysis of requirements, and results in a structured data flow form of specification" [Finkelstein 92]. CORE is a mature method with a set of guidelines on how to apply the method to a problem [SDS 86]. The method is a flexible approach to requirements elicitation, permitting it to be applied to a wide set of problems. CORE encourages contributions from many different communities to develop requirements. CORE delineates the tasks of the members of this community (e.g., Viewpoint Authorities) and structures the communication between these groups [Christel 92]. An incremental examination of information flows and processing activities can be executed using CORE, with each previous step providing the foundation for the present step of specification. CORE assists in discovering design limitations.

Issue-Based Information Systems (IBIS)

Developed by Horst Rittel, the IBIS method is based on the principle that the design process for complex problems, which Rittel terms "wicked" problems, is essentially an exchange among the stakeholders in which they bring their personal expertise and perspective to the resolution of design issues [Kunz 70].

Any problem, concern, or question can be an issue and may require discussion and resolution in order for the design to proceed. The IBIS model centers on this give and take that constitutes the design process. The model was developed over 20 years ago and has been implemented effectively in varied design situations from architectural design to planning at the World Health Organization.

The IBIS model focuses on the articulation of the key issues in the design problem. Each issue can have many positions. A position is a statement or assertion that resolves the issue. Often positions will be mutually exclusive of each other, but the method does not require this. Each of an issue's positions, in turn, may have one or more arguments that either support or object to it.

There are several types of links among the concepts in IBIS. For example, a position responds to an issue with a "responds to" link. Arguments must be linked to their positions with either "supports" or "objects to" links. Issues may generalize or more narrowly focus other issues and they may question or be suggested by other issues, positions, and arguments.

The results of applying IBIS are discussed in the companion case study article.

Joint Application Development (JAD)

JAD [Wood 95] is specifically designed for the development of large computer systems. The goal of JAD is to involve all stakeholders in the design phase of the product via highly structured and focused meetings. Typical participants in the session include a facilitator, end users of the product, main developers, and observers.

In the preliminary phases of JAD, the requirements-engineering team is tasked with fact finding and information gathering. Typically, the outputs of this phase, as applied to security requirements elicitation, are security goals and artifacts. The actual JAD session is then used to validate this information by establishing an agreed-on set of security requirements for the product.

Feature-Oriented Domain Analysis (FODA)

FODA is a domain analysis and engineering method that focuses on developing reusable assets [Kang 90]. By examining related software systems and the underlying theory of the class of systems they represent, domain analysis can provide a generic description of the requirements of that class of systems in the form of a domain model and a set of approaches for their implementation. The FODA method was founded on two modeling concepts: abstraction and refinement [Kean 97]. Abstraction is used to create domain models, as described above, from the specific applications in the domain. These generic domain products abstract the functionality and designs of the applications in a domain. The generic nature of the domain products is created by abstracting factors that make one application different from other related applications. The FODA method advocates that applications in the domain should be abstracted to the level where no differences exist between the applications. Specific applications in the domain are developed as refinements of the domain.

The FODA method has three phases:

1. **context analysis**
   Information required for various activities is gathered from various sources.
2. **domain modeling**
   Product line requirements are analyzed using a set of domain models. Common and variable requirements are identified using a technique called feature modeling. Feature models consist of diagrams that represent features in a hierarchical structure. These requirements are further analyzed using several structured system-analysis techniques such as data flow diagrams, entity relationship diagrams, and functional diagrams.

3. **architecture modeling**

   Domain models are used to create an architecture model. The architecture model can be instantiated to develop individual applications [Kuloor 02].

Critical Discourse Analysis (CDA)

CDA uses sociolinguistic methods to analyze verbal and written discourse [Schiffrin 94]. Sociolinguistics assigns special significance to the structure of speech and texts and provides methods for specifying the linguistic features of different types of discourse units and the way they are tied together into larger units of meaning [Alvarez 02].

Moreover, CDA concerns itself with examining social context along the lines of ideology, power, and inequality. Through discourse examination, topics of power inequalities usually along the lines of race, class, gender, sexuality, and occupation are exposed. Therefore, CDA demystifies what is taken to be common sense by "de-familiarizing" it and signaling its functions and consequences in sustaining the social order.

In particular, CDA can be used to analyze requirements elicitation interviews and to understand the narratives and "stories" that emerge during requirements elicitation interviews.

Accelerated Requirements Method (ARM)

The ARM process [Hubbard 00] is a facilitated requirements elicitation and description activity. There are three phases of the process:

1. Preparation Phase
2. Facilitated Session Phase
3. Deliverable Closure Phase

In addition, there are various assumed Successor Activities Phases.

During the Preparation Phase, planning and preparation are completed to ensure an effective session. During this activity, the overarching goals and objectives, and the preliminary scope of the effort are defined; key success measures are defined; key participants are identified; and the preliminary schedule is developed. The Preparation Phase typically has a duration of one to four days.

During the Session Phase, a trained—and content neutral—facilitator leads the selected participants through a structured process to collect the functional requirements of the project under consideration. The facilitated process employs defined scoping, brainstorming, and explanatory and prioritization techniques. This stage typically has a duration of three days.

During the Closure Phase, the key deliverables, such as a requirements collection, are polished, published, and disseminated, and the various following activities are planned.

The ARM process is similar to JAD but has certain significant differences from the baseline JAD method, which contribute to its uniqueness. For example, in this process, the facilitators are content-neutral, the group dynamic techniques used are different from those used in JAD, the brainstorming techniques used are different, and the requirements are recorded and organized using different conceptual models.

ELICITATION EVALUATION CRITERIA

The following are example evaluation criteria that may be useful in selecting an elicitation method, but certainly there are other criteria that you could use. The main point is to use criteria and to have a common understanding of what they mean.

adaptability: The method can be used to generate requirements in multiple environments. For example, the elicitation method works equally as well with a software product that is near completion as with a project in the planning stages.

computer-aided software engineering (CASE) tool: The method includes a CASE tool. (The Software Engineering Institute defines a CASE tool as "a computer-based product aimed at supporting one or more software engineering activities within a software development process" [SEI 04].)

- **stakeholder acceptance**: The stakeholders are likely to agree to the elicitation method in analyzing their requirements. For example, the method isn't too invasive in a business environment.
- **easy implementation**: The elicitation method isn't overly complex and can be properly executed easily.
- **graphical output**: The method produces readily understandable visual artifacts.
- **quick implementation**: The requirements engineers and stakeholders can fully execute the elicitation method in a reasonable length of time.
- **shallow learning curve**: The requirements engineers and stakeholders can fully comprehend the elicitation method within a reasonable length of time.

- **high maturity**: The elicitation method has experienced considerable exposure and analysis in the requirements engineering community.
- **scalability**: The method can be used to elicit the requirements of projects of different sizes, from enterprise-level systems to small-scale applications.

Note that this approach presumes that all criteria are equally important. If some criteria are more important than others, a weighted average can be used. For example, availability of a CASE tool might be more important than graphical output. A typical weighting scheme could consider criteria to be “essential” with weight 3, “desirable” with weight 2, and “optional” with weight 1. The elicitation methods can then be ranked using a tabular form, such as the example shown in Table 1. This is not intended to be an actual recommendation to use a specific method. You can develop your own comparison criteria and ratings.

Table 1. Comparison of elicitation methods

3 = Very Good, 2 = Fair, 1 = Poor

	Misuse Cases	SSM	QFD	CORE	IBIS	JAD	FODA	CDA	ARM
Adaptability	3	1	3	2	2	3	2	1	2
CASE Tool	1	2	1	1	3	2	1	1	1
Stakeholder Acceptance	2	2	2	2	3	2	1	3	3
Easy Implementation	2	2	1	2	3	2	1	1	2
Graphical Output	2	2	1	1	2	1	2	2	3
Quick Implementation	2	2	1	1	2	1	2	2	3
Shallow Learning Curve	3	1	2	1	3	2	1	1	1
High Maturity	2	3	3	3	2	3	2	2	1
Scalability	1	3	3	3	2	3	2	1	2
Total Score	18	18	17	16	22	19	14	14	18

In our case studies, we decided to use JAD, ARM, and IBIS on three different projects. These three methods were subjectively ranked to be the most suitable candidates for the case studies, given the time and effort constraints that we were working with. We considered not just the total score. The learning curve was an important factor, and the team attempted to select methods that were not too similar to one another, to have some variety. In our case studies, we had the most success using ARM to identify security requirements. Detailed results for all three methods can be found in Requirements Elicitation Case Studies Using IBIS, JAD, and ARM.

ADDITIONAL CONSIDERATIONS

It is possible that a combination of methods may work best. You should consider this as part of the evaluation process, assuming that you have sufficient time and resources to assess how methods may be combined and to actually combine them. You should also consider the time necessary to implement an elicitation method and the time needed to learn a new tool that supports a method. Selecting an elicitation method that meets the needs of a diverse group of stakeholders aids in addressing a broader range of security requirements.

RECOMMENDATIONS

Organizations need to do a better job of identifying security requirements. It is not sufficient to list obvious requirements, such as strong passwords, encryption, and access control mechanisms, and declare victory. A systematic approach is needed to ensure that security requirements are captured. Otherwise the resultant system is likely to contain many avoidable security flaws. We recommend that organizations take the time to select an elicitation method using a systematic tradeoff analysis approach, such as we have outlined here.

REFERENCES

[Alvarez 02]

Alvarez, R. "Discourse Analysis of Requirements and Knowledge Elicitation Interviews." *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35).* Big Island, HI, January 7-10, 2002 (2002).

[Anton 01]

Anton, A. I.; Dempster, J. H.; & Siege, D. F. "Deriving Goals from a Use Case Based Requirements Specification for an Electronic Commerce System," 10-19. *Proceedings of the Sixth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2000)*. Stockholm, Sweden, June 5-6, 2000. London, England: Springer-London, 2001.

[Checkland 90]

Checkland, P. *Soft System Methodology in Action*. Toronto, Ontario, Canada: John Wiley & Sons, 1990.

[Christel 92]

Christel, M. & Kang, K. *Issues in Requirements Elicitation* (CMU/SEI-92-TR-012, ADA258932). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.

[Chung 06]

Chung, L.; Hung, F.; Hough, E.;Ojoko-Adams, D. *Security Quality Requirements Engineering (SQUARE): Case Study Phase III* (CMU/SEI-2006-SR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.

[Finkelstein 92]

Finkelstein, A. "TARA: Tool Assisted Requirements Analysis," 413- 432. *Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development*. New York, NY: John Wiley & Sons, 1992.

[Hickey 03]

Hickey, A.; Davis, A.; & Kaiser, D. "Requirements Elicitation Techniques: Analyzing the Gap Between Technology Availability and Technology Use." C*omparative Technology Transfer and Society 1*, 3 (December 2003): 279-302.

[Hickey 04]

Hickey, A. & Davis, A. "A Unified Model of Requirements Elicitation." *Journal of Management Information Systems 20*, 4 (Spring 2004): 65-84.

[Hubbard 00]

Hubbard, R.; Mead, N.; & Schroeder, C. "An Assessment of the Relative Efficiency of a Facilitator-Driven Requirements Collection Process with Respect to the Conventional Interview Method." *International Conference on Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press, June 2000.

[Jacobson 92]

Jacobson, I. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Boston, MA: Addison-Wesley, 1992.

[Kang 90]

Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novack, W. E.; & Peterson, A.S. *Feature-Oriented Domain Analysis Feasibility Study* (CMU/SEI-90-TR-021, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[Kean 97]

Kean, L. "Feature-Oriented Domain Analysis." *Software Technology Roadmap*. (1997).

[Kuloor 02]

Kuloor, C. & Eberlein, A. *Requirements Engineering for Software Product Lines* (2002).

[Kunz 70]

Kunz, W. & Rittel, H. *Issues as Elements of Information Systems, Working Paper 131*. Berkeley: Institute of Urban & Regional Development, University of California.

[McGraw 06]

McGraw, G. *Software Security: Building Security In,* Boston, MA: Addison-Wesley, 2006, pp. 205-222.

[Mullery 79]

Mullery, G. P. "CORE: A Method for Controlled Requirements Specification," 126-135. *Proceedings of the 4th International Conference on Software Engineering (ICSE-4)*. Munich, Germany, September 17-19, 1979. Los Alamitos, CA: IEEE Computer Society Press, 1979.

[QFD 05]

QFD Institute. *Frequently Asked Questions About QFD* (2005).

[Rumbaugh 94]

Rumbaugh, J. "Getting Started: Using Use Cases to Capture Requirements." *Journal of Object-Oriented Programming 7*, 5 (September 1994): 8-23.

[Schiffrin 94]

Schiffrin, D. *Approaches to Discourse*. Oxford, England: Blackwell Publishers Ltd, 1994.

[SDS 86]

Systems Designers Scientific. *CORE—The Method: User Manual*. London, England: SD-Scicon, 1986.

[SEI 04]

Software Engineering Institute. *What is a CASE Environment?* (2004).

[Sindre 00]

Sindre, G. & Opdahl, A. L. "Eliciting Security Requirements by Misuse Cases," 120-131. *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages (Tools 37-Pacific 2000)*. Sydney, Australia, November 20-23, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Wood 95]

Wood, J. & Silver, D. *Joint Application Development, 2nd ed.*, New York: Wiley, 1995.

[Zowghi 05]

Zowghi, D. & Coulin, C. "Requirements Elicitation: A Survey of Techniques, Approaches, and Tools", Book Chapter in *Engineering and Managing Software Requirements*. Edited by Aybuke Aurum and Claes Wohlin. Heidelberg, Germany: Springer-Verlag, 2005.