Software Engineering Institute

How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods

Nancy R. Mead

August 2007

TECHNICAL NOTE
CMU/SEI-2007-TN-021

CERT Program

Carnegie Mellon

This work is supported by the Army Research Office through grant number DAAD19-02-1-0389 ("Perpetually Available and Secure Information Systems") to CMU's CyLab. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

List of Tables

Abstract

The Security Quality Requirements Engineering (SQUARE) method, developed at the Carnegie Mellon Software Engineering Institute, provides a systematic way to identify security requirements in a software development project. This report describes SQUARE and then describes other methods used for identifying security requirements, such as the Comprehensive, Lightweight Application Security Process, the Security Requirements Engineering Process, and Tropos, and compares them with SQUARE. The report concludes with some guidelines for selecting a method and a look at some related trends in requirements engineering.

1 Background: The Importance of Requirements Engineering

It is well recognized in industry that requirements engineering is critical to the success of any major development project. Several authoritative studies have shown that requirements engineering defects cost 10 to 200 times as much to correct once fielded than if they were detected during requirements development. Other studies have shown that reworking requirements defects on most software development projects costs 40 to 50 percent of total project effort, and the percentage of defects originating during requirements engineering is estimated at more than 50 percent. The total percentage of project budget due to requirements defects is 25 to 40 percent.

Requirements problems are the number one reason why projects

- are significantly over budget
- are significantly past schedule
- have significantly reduced scope
- deliver poor-quality applications
- are not significantly used once delivered
- are cancelled

Requirements engineering typically suffers from the following major problems:

- Requirements identification typically does not include all relevant stakeholders and does not use the most modern or efficient techniques.
- Requirements analysis typically is either not performed at all (identified requirements are directly specified without any analysis or modeling) or analysis is restricted to functional requirements and ignores quality requirements, other nonfunctional requirements, and architecture, design, implementation, and testing constraints.
- Requirements specification is typically haphazard, with specified requirements being ambiguous, incomplete (e.g., non-functional requirements are often missing), inconsistent, not cohesive, infeasible, obsolete, neither testable nor capable of being validated, and not usable by all intended audiences.
- Requirements management is typically weak, with poor storage (e.g., in one or more documents rather than in a database or tool) and missing attributes, and is limited to tracing, scheduling, and prioritization.

1.1 SECURITY REQUIREMENTS ISSUES

In reviewing requirements documents, we typically find that security requirements, when they exist, are in a section by themselves and have been copied from a generic set of security requirements. They tend to be general mechanisms such as password protection, firewalls, virus detection tools, and the like. The requirements elicitation and analysis that is needed to get a better set of security requirements seldom takes place. Even when it does, the security requirements are often developed independently of the rest of the requirements engineering activity and hence are not integrated into the mainstream of the requirements activities. As a result, security requirements that are specific to the system and that provide for protection of essential services and assets are often neglected.

Although data exists to support the benefits of requirements engineering in general, the data to specifically support the benefits of security requirements engineering is anecdotal. However, the National Institute of Standards and Technology (NIST) reports that software that is faulty in security and reliability costs the economy $59.5 billion annually in breakdowns and repairs [NIST 2002]. The costs of poor security requirements show that even a small improvement in this area would provide a high value. A recent study found that the return on investment when security analysis and secure engineering practices are introduced early in the development cycle ranges from 12 to 21 percent, with the highest rate of return occurring when the analysis is performed during application design [Soo Hoo 2001]. By the time that an application is fielded and in its operational environment, it is very difficult and expensive to significantly improve its security.

If security requirements are not effectively defined, the resulting system cannot be effectively evaluated for success or failure prior to implementation.

1.1.1 The Problem of Negative Requirements

Much requirements engineering research and practice has addressed the capabilities that the system will provide. So a lot of attention is given to the functionality of the system, from the user's perspective, but little attention is given to what the system should *not* do. In one discussion on requirements prioritization for a specific large system, ease of use was assigned a higher priority than security requirements. Security requirements were in the lower half of the prioritized requirements. This occurred in part because the only security requirements that were considered had to do with access control.

Current research recognizes that security requirements are negative requirements. General security requirements, such as "the system shall not allow successful attacks," are therefore generally not feasible, because there is no agreement on ways to validate them other than to apply formal methods to the entire system, including COTS components. We can, however, identify the essential services and assets that must be protected. We are able to validate that mechanisms such as access control, levels of security, backups, replication, and policy are implemented and enforced. We can also validate that the system will properly handle specific threats identified by a threat model and correctly respond to intrusion scenarios.

Many methods have been developed that facilitate this kind of requirements analysis and the development of security requirements. The objective of this report is to provide an overview of various security requirements engineering methods and to compare them with one developed at the Software Engineering Institute, the Security Quality Requirements Engineering (SQUARE) method.

2 Methods and Practices

Many requirements engineering research projects undertaken in recent years have resulted in the development of methods and processes that can be used in identifying security requirements. These are some of them:

- Security Quality Requirements Engineering (SQUARE) is a process aimed specifically at security requirements engineering.
- The Comprehensive, Lightweight Application Security Process (CLASP) approach to security requirements engineering [OWASP 2007] is a life-cycle process that suggests a number of different activities across the development life cycle to improve security. Among these is a specific approach for security requirements.
- Core security requirements artifacts [Moffett 2004] takes an artifact view and starts with the artifacts that are needed to achieve better security requirements.
- The Security Requirements Engineering Process (SREP) [Mellado 2007] is a nine-step process that is based partially on SQUARE but incorporates consideration of the Common Criteria and notions of reuse.
- Security patterns are useful in going from requirements to architectures and then designs [Haley 2007, Rosado 2006, Weiss 2007].
- Tropos is a self-contained life-cycle approach [Giorgini 2007]. It is very specific in terms of how to go about requirements specification.
- Other useful techniques are the use of attack trees in security requirements engineering [Ellison 2003] and misuse and abuse cases [Alexander 2003, Fernandez 2007, Sindre 2000]. Formal specification approaches to security requirements, such as Software Cost Reduction (SCR) [Heitmeyer 2002] have also been useful. The higher levels of the Common Criteria [CCEVS 2007] provide similar results.

These are described in more detail below.

2.1 OVERVIEW OF THE SQUARE PROCESS

Security Quality Requirements Engineering (SQUARE) is a process model developed at Carnegie Mellon University [Mead 2005a, Mead 2005b]. This process provides a means for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications. The focus of this methodology is to build security concepts into the early stages of the development life cycle. The model can also be used for documenting and analyzing the security aspects of fielded systems and for steering future improvements and modifications to those systems.

Subsequent to initial development, SQUARE was applied in a series of client case studies. Carnegie Mellon graduate students worked on this project during the summer and fall of 2004 and the summer of 2005. The case study results were published [Chen 2004, Gordon 2005, Xie 2004]. Prototype tools were also developed to support the process. The draft process was revised and baselined after completion of the case studies; the baselined process is shown in Table 1. In principle, Steps 1-4 are actually activities that precede security requirements engineering but are necessary to ensure that it is successful. Detailed discussion of the method can be found in *Security Quality Requirements Engineering (SQUARE) Methodology* [Mead 2005a].

Table 1: The SQUARE Process

Step	Input	Techniques	Participants	Output
1 Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements team	Agreed-to definitions
2 Identify security goals	Definitions, candidate goals, business drivers, policies and procedures, examples	Facilitated work session, surveys, interviews	Stakeholders, requirements engineer	Goals
3 Develop Artifacts	Potential artifacts (e.g., scenarios, misuse cases, templates, forms)	Work session	Requirements engineer	Needed artifacts: scenarios, misuse cases, models, templates, forms
4 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
5 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
6 Elicit security requirements	Artifacts, risk assessment results, selected techniques	Accelerated Requirements Method (ARM), 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

Step	Input	Techniques	Participants	Output
7 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
8 Prioritize requirements	Categorized requirements and risk assessment results	Prioritization methods such as AHP, Triage, Win-Win, etc.	Stakeholders facilitated by requirements engineer	Prioritized requirements
9 Requirements inspection	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan, peer reviews, etc.	Inspection team	Initial selected requirements, documentation of decision-making process and rationale

2.1.1 How to Apply SQUARE

The SQUARE process is best applied by the project's requirements engineers and security experts, in the context of supportive executive management and stakeholders. We believe the process works best when elicitation occurs after risk assessment has been done (Step 4) and when security requirements are specified prior to critical architecture and design decisions. Thus critical business risks will be considered in the development of the security requirements.

Step 1, Agree on Definitions, is needed as a prerequisite to security requirements engineering. On a given project, team members will tend to have definitions in mind, based on their prior experience, but those definitions will not necessarily agree [Woody 2005]. For example, to some government organizations, security has to do with access based on security clearance levels, whereas to others security may have to do with physical security or cyber security. It is not necessary to invent definitions. Most likely, sources such as IEEE and SWEBOK will provide a range of definitions to select from or tailor. A focus group meeting with the interested parties can enable the selection of a consistent set of definitions for the security requirements activity.

Step 2, Identify Security Goals, should be done at the organizational level and is needed to develop the information system. This provides a consistency check with the organization's policies and operational security environment. Stakeholders from different areas often have different goals. For example, a stakeholder in human resources may be concerned about maintaining the confidentiality of personnel records, whereas a stakeholder in a financial area may be concerned with ensuring that financial data is not accessed or modified without authorization. It is important to have a representative set of stakeholders, including those with operational expertise. Once the goals of the various stakeholders have been identified, they will need to be prioritized. In the absence of consensus, an executive decision may be necessary to prioritize goals.

Step 3, Develop Artifacts, supports all the subsequent activities. It is often the case that organizations do not have a documented concept of operations for a project, succinctly stated project goals, documented normal usage and threat scenarios, misuse cases, and other documents needed to support requirements definition. This means that either the entire requirements process is built on a foundation of sand or a lot of time is spent backtracking to try to obtain such documentation.

Step 4, Perform Risk Assessment, requires an expert in risk assessment methods, the support of the stakeholders, and the support of a requirements engineer. There are a number of risk assessment methods to select from. A specific method can be recommended by the risk assessment expert, based on the needs of the organization. The artifacts from Step 3 provide the input to the risk assessment process. The outcomes of the risk assessment can help in identifying the high-priority security exposures. Organizations that do not perform risk assessment typically do not have a logical approach to considering organizational risk when identifying security requirements but tend to select mechanisms, such as encryption, without really understanding the problem that is being solved.

Step 5, Select Elicitation Technique, becomes important when there are several classes of stakeholders. A more formal elicitation technique, such as the Accelerated Requirements Method [Hubbard 1999], Joint Application Design [Wood 1989], or structured interviews can be effective in overcoming communication issues when there are stakeholders with different cultural backgrounds. In other cases, elicitation may simply consist of sitting down with a primary stakeholder to try to understand that stakeholder's security requirements needs.

Step 6, Elicit Security Requirements, is the actual elicitation process using the selected technique. Most elicitation techniques provide detailed guidance on how to perform elicitation. This builds on the artifacts that were developed in earlier steps, such as misuse and abuse cases, attack trees, threats, and scenarios.

Step 7, Categorize Requirements, allows the requirements engineer to distinguish among essential requirements, goals (desired requirements), and architectural constraints that may be present. Requirements that are actually constraints typically occur when a specific system architecture has been chosen prior to the requirements process. This is good, as it allows assessment of the risks associated with these constraints. This categorization also helps in the prioritization activity that follows.

Step 8, Prioritize Requirements, depends not only on the prior step but may also involve performing a cost/benefit analysis to determine which security requirements have a high payoff relative to their cost.

Step 9, Requirements Inspection, can be done at varying levels of formality, from Fagan Inspections to peer reviews. Once inspection is complete, the organization should have an initial set of prioritized security requirements. It should also understand which areas are incomplete and must be revisited at a later time. Finally, the organization should understand which areas are dependent on specific architectures and implementations and should expect to revisit those as well.

2.2 THE COMPREHENSIVE, LIGHTWEIGHT APPLICATION SECURITY PROCESS

The following overview of the Comprehensive, Lightweight Application Security Process (CLASP) is extracted from the Build Security In Web site article "Introduction to the CLASP Process" [Graham 2006].

The CLASP Process is presented through five high-level perspectives called *CLASP Views*. These views allow CLASP users to quickly understand the CLASP Process, including how CLASP process components interact and how to apply them to a specific software development life cycle.

These are the CLASP Views:

- Concepts View

  This view provides a high-level introduction to CLASP by briefly describing, for example, the interaction of the five CLASP Views, the seven CLASP Best Practices, the CLASP Taxonomy, the relation of CLASP to security policies, and a sample sequence for applying CLASP process components.

- Role-Based View

  This view contains role-based introductions to the CLASP Process.

- Activity-Assessment View

  This view helps project managers assess the appropriateness of the 24 CLASP Activities and select a subset of them. CLASP provides two sample road maps (legacy and new-start) to help select applicable activities.

- Activity-Implementation View

  This view contains the 24 security-related CLASP Activities that can be integrated into a software development process. The activities phase of the SDLC translates into executable software any subset of the 24 security-related activities assessed and accepted in Activity Assessment.

- Vulnerability View

  This view contains a catalog of the 104 underlying “problem types” identified by CLASP that form the basis of security vulnerabilities in application source code. CLASP divides the 104 problem types into five high-level categories. An individual problem type in itself is often not a security vulnerability; frequently, it is a combination of problems that lead to a vulnerability in source code.

  Associated with the Vulnerability View are the CLASP Vulnerability Use Cases, which depict conditions under which security services are vulnerable to attack at the application layer. The use cases provide CLASP users with easy-to-understand, specific examples of the relationship between security-unaware source coding and possible resulting vulnerabilities in basic security services.

Within a software development project, the CLASP Best Practices are the basis of all security-related software development activities—whether planning, designing, or implementing—including the use of all tools and techniques that support CLASP.

These are the CLASP Best Practices:

- Institute awareness programs.
- Perform application assessments.
- Capture security requirements.
- Implement secure development practices.
- Build vulnerability remediation procedures.
- Define and monitor metrics.
- Publish operational security guidelines.

2.3 CORE SECURITY REQUIREMENTS ARTIFACTS

Core security requirements artifacts provides a framework that includes traditional requirements engineering approaches to functional requirements and an approach to security requirements engineering that focuses on assets and harm to those assets. There are several fundamental ideas. One notion is that requirements engineering, including security requirements engineering, needs to focus on the *what* rather than the *how*. Another, following Michael Jackson's frameworks approach, is that security is not a feature of software alone, but also of the real world. This leads to the notion of different views. A third is that security requirements serve to constrain end-user functional requirements. Finally, the nature of arguments about security is discussed, with the idea that arguments about security must be grounded in the real-world situation in which security claims are made. This approach results in a specific process for security requirements engineering.

2.4 SECURITY REQUIREMENTS ENGINEERING PROCESS

As noted before, Security Requirements Engineering Process (SREP) incorporates notions of the Common Criteria and reuse. These are the SREP activities:

- Agree on definitions
- Identify vulnerable and/or critical assets
- Identify security objectives and dependencies
- Identify threats and develop artifacts
- Risk assessment
- Elicit security requirements
- Categorize and prioritize requirements
- Requirements inspection
- Repository improvement

2.5 SECURITY PATTERNS

The definition of a security pattern used in some sources is as follows: "A security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution. According to their level of abstraction, patterns can be divided into architectural patterns and design patterns" [Buschmann 1996, Rosado 2006, Weiss 2007]. Table 2 shows a mapping between security requirements, architectural patterns, design patterns, and security standards [Rosado 2006]. As you can see, security patterns can be extremely useful in an overall life-cycle approach, although they are likely to be most useful after security requirements are identified.

Table 2: Relationship Between Requirements, Patterns, and Standards

Security Requirements	Architectural Patterns	Design Patterns	Security Standards
Authentication	QoP Role-based security Assertion coordinator Data filter Check point SSO Cryptographic Direct authentication	Assertion builder SSO Delegation Sender authentication Authenticator Credential tokenizer	LibAlliance SASL based authentication service; SAML 2.0WS-Security + SAML 2.0 + Kerberos Token ProfileXML Key Management SystemWS-Security + XML Digital Signature SAML 2.0 + WS-Security + SAML Token Profile + XML Digital Signature SAML 2.0, Liberty Alliance Project ID-FF 1.1, WS-Federation
	Brokered authentication	Security token service, X.509 PKI, Kerberos	WS-Security + SAML 2.0 + Kerberos Token Profile
Authorization	PEP + PDP + PRP + PIP + PAP Data Filter Bodyguard Check point, Firewall	XML firewall filter Assertion builder Authorization RBAC Session	WS-Policy + WS-SecurityPolicy; XACML Profile; XrML ODRLWS-Authorization
Confidentiality	QoP, Encryption, Cryptographic, Layered security	Message inspector, Information secrecy, Secure pipe, Session	WS-Security + XML Encryption
Integrity	QoP Firewall Data filter Layered security	Message inspector, Secure pipe, Message integrity, Secure message router, Authoritative source of data multilevel security	WS-Security + XML Digital Signature
Audit	Check point Single access point	Audit interceptor Secure logger	

2.6 TROPOS

Tropos addresses four software development phases called Early Requirements, Late Requirements, Architectural Design, and Detailed Design. It uses a specific modeling approach, with graphical representation, to develop goal and plan models. Modeling activities include actor modeling, dependency modeling, production of actor diagrams, and plan modeling. Goals are decomposed into subgoals. Although Tropos has been in existence for some time, it was recognized that there was a need for Tropos to address security requirements. This led to the development of Secure Tropos.

Secure Tropos enables security constraints to be expressed throughout the development life cycle. In addition, concepts of trust, ownership, and delegation were introduced. Secure Tropos addresses the same four software development phases as Tropos, with security enhancements and modeling of specific security features. Secure Tropos has continued to be refined and developed and has been the subject of case studies. It has also been introduced in academic courses.

2.7 USE OF ATTACK TREES FOR MODELING AND ANALYSIS

The notion of attack trees as a method for modeling attacks has been described extensively in the literature [Schneier 2000]. The work by Ellison and Moore explores the use of attack trees in the development of intrusion scenarios, which can then be used to identify security requirements [Ellison 2003, Moore 2001]. A small attack tree example is shown in Figure 1.

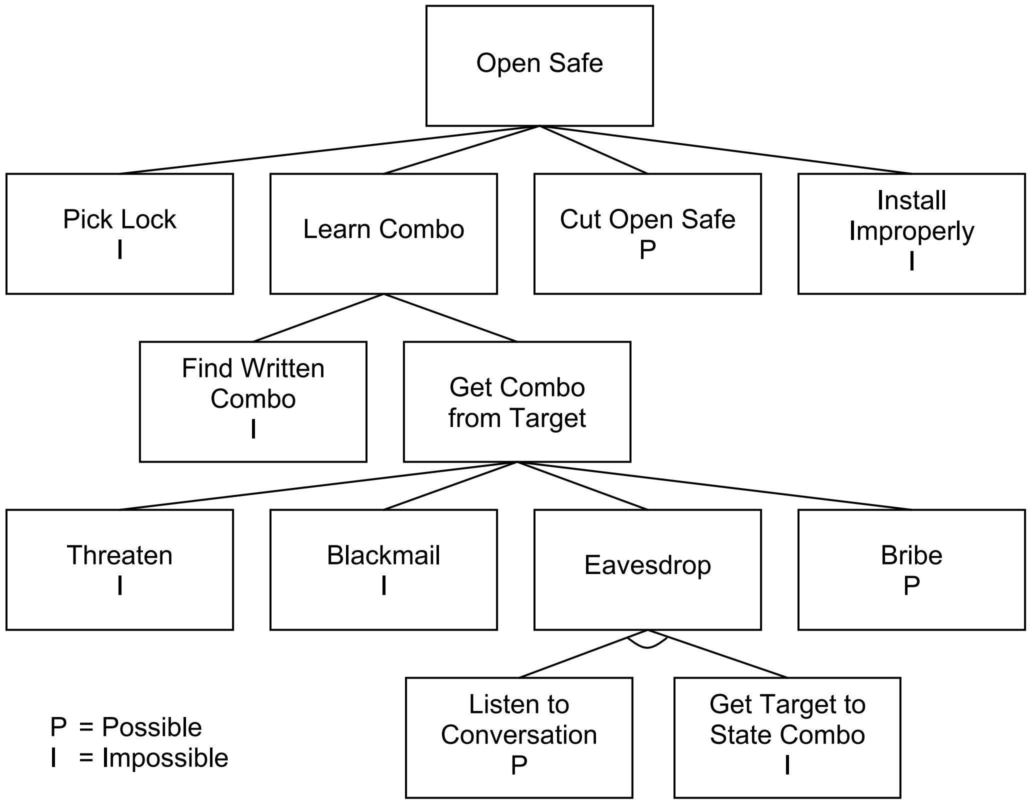


Figure 1: Attack Tree Example

Once fault trees have been used to model intrusions, they can also be used to help identify requirements for intrusion detection systems, as described by Ellison and Moore. Alternatively, fault tree analysis can be used to identify other security requirements, once the fault trees have been used to model intrusion behavior. Formal use of fault trees suggests the possibility of formal analysis, which could be a great advantage in developing a set of consistent and complete requirements.

2.8 MISUSE AND ABUSE CASES

A security *misuse* case, a variation on a use case, is used to describe a scenario from the point of view of the attacker [Alexander 2003, Sindre 2000, Sindre 2002]. Since use cases have proven useful in documenting normal use scenarios, they can also be used to document intruder usage scenarios and ultimately to identify security requirements or security use cases [Firesmith 2003]. A similar concept has been described as an *abuse* case [McDermott1999, McDermott 2001].

One obvious application of a misuse case is in eliciting requirements. Since use cases are used successfully for eliciting requirements, it follows that misuse cases can be used to identify potential threats and to elicit security requirements. In this application, the traditional user interaction with the system is diagrammed simultaneously with the hostile user's interactions. An example of this is shown in Figure 2 [Alexander 2003].

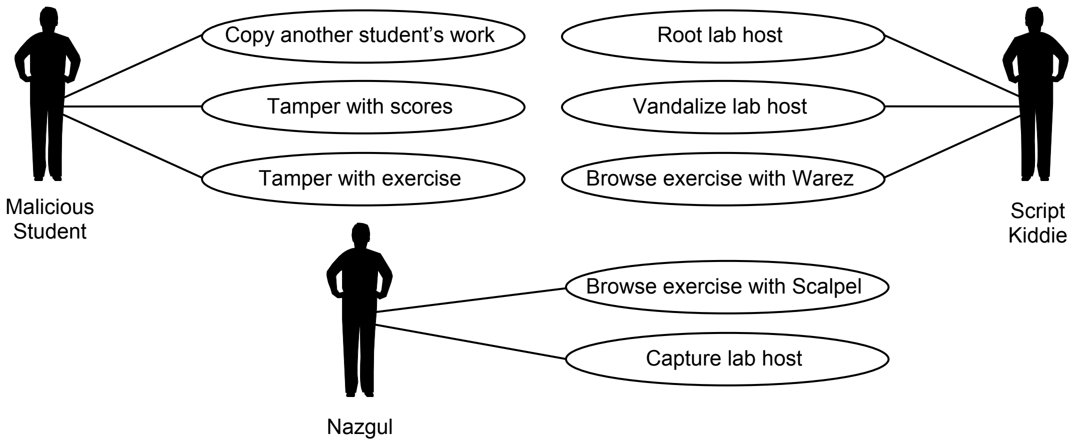


Figure 2: Abuse Case Diagram for an Internet-Based Information Security Laboratory

Alternatively, abuse cases tend to show the "abuse" side of the system, in contrast to traditional use cases. The contrast between use and abuse cases is shown in Table 3 [McDermott 1999].

Table 3: Contrast Between Use and Abuse Cases

Use Case	**Abuse Case**
• A complete transaction between one or more actors and a system • UML-based use case diagrams • Typically described using natural language	• A family of complete transactions between one or more actors and a system that results in harm • UML-based use case diagrams • Typically described using natural language. A tree/DAG diagram may also be used. • Potentially one family member for each kind of privilege abuse and for each component that might be exploited • Includes a description of the range of security privileges that might be abused • Includes a description of the harm that results from an abuse case

2.9 FORMAL METHODS

Formal methods are typically used in the specification and verification of secure systems. From a life-cycle viewpoint, the specification typically represents either formal requirements or a formal step between informal requirements and design.

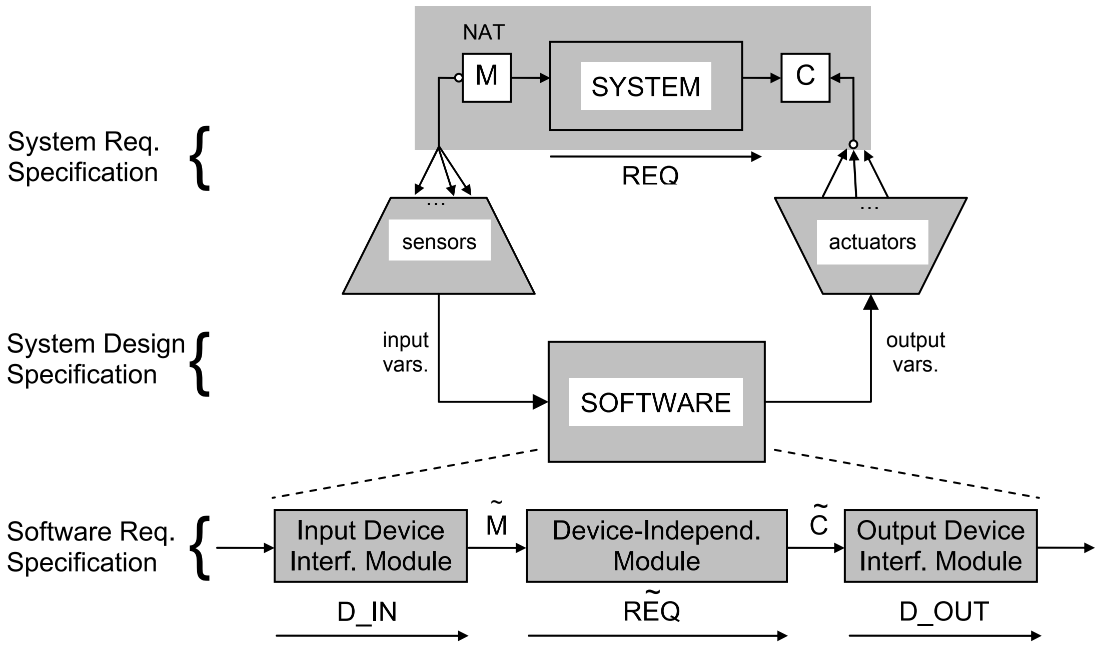
Some formal methods are applied to security standards, such as the Common Criteria. Organizational objectives are translated into the specification of all relevant security functions in a planned system. The subset of specifications to be implemented is identified and further assessment or risk analysis takes place [Leiwo 1999a]. The Common Criteria is used during the second, or evaluation, phase. The Kruger-Eloff process, based on the Common Criteria, is used for evaluation of information security. Another method focuses more generally on information security policy specification [Ortalo 1998]. A formal specification language is described, and in a case study the method is applied to the description of security requirements for a medium-size banking organization. This method provides flexibility and expression to correspond to specific organizational needs.

The B formal method is used specifically to support the design and validation of the transaction mechanism for smart card applications. The mathematical proofs provide confidence that the design of the transaction mechanism satisfies the security requirements [Sabatier 1999].

An interesting contribution is a model that focuses on modeling the organization in which information security is developed [Leiwo 1999b]. The organization is described in layers of abstraction. In addition, a notation for expressing security requirements is described, under a framework of harmonization functions and merging of requirements. A case study that focuses on the security requirements for sharing of patient data among hospitals and medical practitioners is described.

2.9.1 Software Cost Reduction

Software Cost Reduction (SCR) is a formal method based on a tabular representation of specifications and analysis of the requirements for complex systems. It was originally developed to document the behavior of the A-7E aircraft [Heninger 1980] and has been augmented with a tool suite and applied to many complex and safety-critical systems [Heitmeyer 2002]. Figure 3 shows the relationship between the System Requirements Specification (SRS), the System Design Specification (SDS), and the Software Requirements Specification (SoRS).



NAT
M
SYSTEM
C
System Req. Specification
REQ
sensors
actuators
System Design Specification
input vars.
output vars.
SOFTWARE
Software Req. Specification
Input Device Interf. Module
M̃
Device-Independ. Module
C̃
Output Device Interf. Module
D_IN
RẼQ
D_OUT

Figure 3: Relationship Between the SRS, the SDS, and the SoRS

This decomposition is commonly used in many large DoD and other government systems. The SCR notation is used for specification. According to Heitmeyer and Bharadwaj,

> *To specify the required system behavior in a practical and efficient manner, the SCR method uses terms and mode classes. A term is an auxiliary variable that helps keep the specification concise. A mode class is a special case of a term, whose values are modes. Each mode defines an equivalence class of system states, useful in specifying the required system behavior. In SCR specifications, we often use prefixes in variable names. In SCR specifications, we often use the following prefixes in variable names: "m" to indicate monitored variables, "t" for terms, "mc" for mode classes, "c" for controlled variables, "i" for input variables, and "o" for output variables.*
>
> *Conditions and events are important constructs in SCR specifications. A condition is a predicate defined on one or more state variables (a state variable is a monitored or controlled variable, a mode class, or a term). An event occurs when a state variable changes value* [Bharadwaj 2003].

Table 4 is an example of an SCR table.

Table 4: Condition Table Defining the Value of Term tRemLL

Mode Class = mcStatus			**Trac.**
Mode	**Condition**		
unoccupied	true	false	FM3
occupied	mIndoorLL > tCurrentLSVal	mIndoorLL ≤ tCurrentLSVal	FM1
temp_empty	mIndoorLL > tCurrentLSVal OR tOverride	mIndoorLL ≤ tCurrentLSVal AND NOT tOverride	FM1, FM6
tRemLL	0	tCurrentLSVal – mIndoorLL	FM1

For systems that require a rigorous specification method, SCR would seem to be a good choice. It is probably not as useful in the early requirements stages, for example during elicitation, and may have the most utility in the specification activity that tends to occur between requirements and design activities.

2.9.2 Common Criteria[1]

The Common Criteria enables an objective evaluation to validate that a particular product or system satisfies a defined set of security requirements. Although the focus of the Common Criteria is evaluation, it presents a standard that should be of interest to those who develop security requirements.

The Common Criteria (CC) was developed through a combined effort of six countries: the United States, Canada, France, Germany, the Netherlands, and the United Kingdom. This effort built on earlier standards, including Europe's Information Technology Security Evaluation Criteria (ITSEC), the United States' Trusted Computer System Evaluation Criteria (TCSEC), and the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC) [Caplan 1999]. A Common Cri-

[1] Much of the material in this section is drawn from the report *International Liability Issues for Software Quality* [Mead 2003]. A definitive source of current information about the Common Criteria is the CCEVS Web site [CCEVS 2007].

teria evaluation provides an objective way to validate that a particular product satisfies a defined set of security requirements. The focus of the Common Criteria is the evaluation of a product or system rather than the development of requirements. Nevertheless, its evaluation role makes it of interest to those who develop security requirements. The Common Criteria allows for seven Evaluation Assurance Levels (EALs), which will be described further.

Common Criteria Overview

The Common Criteria contains a grouping of 60 security functional requirements in 11 classes [Abrams 2000]. This grouping allows specific classes of requirements to be evaluated in a standard way to arrive at an Evaluation Assurance Level.

A package is an intermediate combination of requirements components that allows expression of a set of functional or assurance requirements that meet a subset of security objectives. A Protection Profile (PP) is an implementation-independent set of security requirements for a class of Targets of Evaluation (TOEs) that meet specific consumer needs. A TOE is basically an IT product or system, together with its documentation and administration, that is the subject of a CC evaluation. A PP allows security requirements to be expressed using a template in an implementation-independent way, and is thus reusable. This provides benefits when implementing a family of related products or a product line. A Security Target (ST) contains a set of security requirements that can be stated explicitly. An ST includes detailed, product-specific information. It can be viewed as a refinement of the PP and forms the agreed-on basis for evaluation.

Functional and assurance security requirements are the basis for the Common Criteria. There are seven Evaluation Assurance Levels (EALs). The higher the level, the more confidence you can have that the security functional requirements have been met. The levels are as follows:

- EAL1: Functionally Tested. Applies when you require confidence in a product's correct operation but do not view threats to security as serious. An evaluation at this level should provide evidence that the target of evaluation functions in a manner consistent with its documentation and that it provides useful protection against identified threats.
- EAL2: Structurally Tested. Applies when developers or users require low to moderate independently assured security but the complete development record is not readily available. This situation may arise when there is limited developer access or when there is an effort to secure legacy systems.
- EAL3: Methodically Tested and Checked. Applies when developers or users require a moderate level of independently assured security and require a thorough investigation of the target of evaluation and its development, without substantial reengineering.
- EAL4: Methodically Designed, Tested, and Reviewed. Applies when developers or users require moderate to high independently assured security in conventional commodity products and are prepared to incur additional security-specific engineering costs.
- EAL5: Semi-Formally Designed and Tested. Applies when developers or users require high, independently assured security in a planned development and require a rigorous development approach that does not incur unreasonable costs from specialist security engineering techniques.

- EAL6: Semi-Formally Verified Design and Tested. Applies when developing security targets of evaluation for application in high-risk situations in which the value of the protected assets justifies the additional costs.
- EAL7: Formally Verified Design and Tested. Applies to the development of security targets of evaluation for application in extremely high-risk situations, as well as when the high value of the assets justifies the higher costs.

3 Comparing Methods

It should be noted that in comparing SQUARE to the other methods we've described, we are to some extent dealing with apples and oranges. Some of the methods apply to the entire life cycle, not just requirements engineering. Some of them are processes specifically aimed at security requirements engineering, in a similar fashion to SQUARE. A third category encompasses specific methods that could be applied within a variety of processes, including SQUARE.

The Tropos material by Giorgini et al. is a self-contained life-cycle approach. It's not likely that an engineer would use both Tropos and SQUARE. If he or she were using Tropos, he or she would use it throughout. As noted earlier, CLASP is a life-cycle process that suggests a number of different activities across the development life cycle in order to improve security.

The Core Artifacts approach is not inconsistent with SQUARE, in that the goals and some of the process steps are similar, but it is a different process for arriving at security requirements.

Fernandez's use of misuse cases and attack patterns is consistent with SQUARE, as we have used misuse cases and attack trees as part of the process. However, there is less detail on how to use these specifically in the requirements area than the SQUARE process provides. Weiss's material on security patterns is consistent with SQUARE and could be used as part of the SQUARE process. Specifically, security patterns could be used to help identify and document security requirements. The security patterns described by Rosado fall into the architecture domain and would be most useful once requirements are in place.

SREP is quite similar to SQUARE. The following is a comparison of SREP and SQUARE.

SQUARE steps:

1. Agree on definitions
2. Identify security goals
3. Develop artifacts to support security requirements definition
4. Perform risk assessment
5. Select elicitation techniques
6. Elicit security requirements
7. Categorize requirements
8. Prioritize requirements
9. Requirements inspection

SREP activities:

1. Agree on definitions (SQUARE step 1)
2. Identify vulnerable and/or critical assets (not called out in SQUARE)
3. Identify security objectives and dependencies (overlaps SQUARE step 2)
4. Identify threats and develop artifacts (overlaps SQUARE step 3)
5. Risk assessment (SQUARE step 4)

6. Elicit security requirements (SQUARE step 6)
7. Categorize and prioritize requirements (SQUARE steps 7 and 8)
8. Requirements inspection (SQUARE step 9)
9. Repository improvement (not part of SQUARE)

As noted earlier, formal methods can be used in the specification and verification of requirements for secure systems. From a life-cycle viewpoint, the specification typically represents either formal requirements or a formal step between informal requirements and design. The elicitation approaches used in SQUARE do not lead directly to formal specifications, but approaches such as SCR or the higher levels of the Common Criteria could be used as a follow-on to the SQUARE process when formal specifications are called for.

We recommend that organizations follow a systematic approach in selecting a requirements engineering method. For example, when we evaluated requirements elicitation methods (one of the SQUARE steps) in a case study, we developed a table of desired attributes and assessed how well we thought each method satisfied the attributes. This could also be done with weights assigned to the attributes. An example is shown in Table 5.

Table 5: Comparison of Elicitation Techniques

	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
Client Acceptance	2	2	2	2	3	2	1	3	3
Complexity	2	2	1	2	3	2	1	1	2
Graphical Output	2	2	1	1	2	1	2	2	3
Implementation Duration	2	2	1	1	2	1	2	2	3
Learning Curve	3	1	2	1	3	2	1	1	1
Maturity	2	3	3	3	2	3	2	2	1
Scalability	1	3	3	3	2	3	2	1	2

Scale: 3 = very good, 2 = fair, 1 = poor.

For requirements engineering methods in general, no doubt the attributes would differ somewhat.

4 Conclusions and Trends in Security Requirements Engineering

In short, there is no single right answer when it comes to security requirements engineering. A lot depends on the processes that are already in place in a particular organization. Some organizations may prefer a detailed, specific method, whereas other organizations may prefer an approach that allows them to select methods to incorporate into existing processes. Another factor is the extent to which the project or organization is mission critical. This can dictate the level of formality used in requirements engineering and the need for assurance levels such as those provided by the Common Criteria.

Many organizations are realizing that security requirements need to be addressed early in the life-cycle process. It is a very active research area, with a wide variety of methods and tools under development. Some organizations, such as Microsoft, already have security requirements engineering methods incorporated into their life-cycle processes. At present, there is no consensus on a single best approach to security requirements engineering. However, many organizations intuitively feel that attention to this area will pay off in supporting their business goals.

There are a number of conferences and workshops that have been held over the last few years on the subject of security requirements. This is a trend that is likely to continue, as additional workshops of this type are already showing up on the calendar.

Another trend, which is somewhat unfortunate, is that many industrial organizations feel that their internal processes give them a competitive edge, so they are unwilling to publish or discuss the details. It was surprising to find that many organizations seem to have established processes for engineering security requirements when so few have published their methods.

Sources and Funding

Many of the techniques described in this report appear on the Build Security In Web site, which is sponsored by the U.S. Department of Homeland Security and developed by the SEI. Others appear in the book *Integrating Security and Software Engineering*, edited by Mouratidis and Giorgini. Much of the SQUARE material is drawn from various reports, papers, and book chapters authored by Mead, notably the BSI Web site articles and a chapter in the Mouratidis and Giorgini book. The section on the Common Criteria is extracted from the Build Security In Web site article "The Common Criteria" [Mead 2006].

This project is financially supported by the Software Engineering Institute, CyLab, and the Heinz School at Carnegie Mellon University.

References

URLs are valid as of the publication date of this document.

[Abrams 2000]
Abrams, M. D. & Brusil, P. J. "Application of the Common Criteria to a System: A Real-World Example." *Computer Security Journal 16*, 2 (Spring 2000): 11-21.

[Alexander 2003]
Alexander, I. "Misuse Cases: Use Cases with Hostile Intent." *IEEE Software 20*, 1 (January-February 2003): 58-66.

[Bharadwaj 2003]
Bharadwaj, R. "How to Fake a Rational Design Process Using the SCR Method," 3-4. *SEHAS'03 International Workshop on Software Engineering for High Assurance Systems*. Portland, OR, May 9-10, 2003. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/community/sehas-workshop/.

[Buschmann 1996]
Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; & Stal, Michael. *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester, NY: John Wiley & Sons, 1996 (ISBN 0-471-95869-7).

[Caplan 1999]
Caplan, K. & Sanders, J. L. "Building an International Security Standard." *IEEE IT Professional 1*, 2 (March/April 1999): 29-34.

[CCEVS 2007]
The Common Criteria Evaluation and Validation Scheme. http://www.niap-ccevs.org/cc-scheme/ (2007).

[Chen 2004]
Chen, P.; Mead, N. R.; Dean, M.; Ojoko-Adams, D.; Osman, H.; Lopez, L.; & Xie, N. *System Quality Requirements Engineering (SQUARE) Methodology: Case Study on Asset Management System* (CMU/SEI-2004-SR-015, ADA431068). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004.
http://www.sei.cmu.edu/publications/documents/04.reports/04sr015.html.

[Ellison 2003]
Ellison, R. J. & Moore, A. P. *Trustworthy Refinement Through Intrusion-Aware Design* (CMU/SEI-2003-TR-002, ADA414865). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
http://www.sei.cmu.edu/publications/documents/03.reports/03tr002.html.

[Fernandez 2007]
Fernandez, E.; Larrondo-Petrie, M.; Sorgente, T.; & Vanhilst, M. "A Methodology to Develop Secure Systems Using Patterns," 107-126. *Integrating Security and Software Engineering*. Edited by H. Mouratidis and P. Giorgini. Hershey, PA: Idea Group Publishing, 2007 (ISBN 1-599-04147-2).

[Firesmith 2003]
Firesmith, D. G. "Security Use Cases." *Journal of Object Technology 2*, 3 (May-June 2003): 53-64. http://www.jot.fm/issues/issue_2003_05/column6.

[Giorgini 2007]
Giorgini, P.; Mouratidis, H.; & Zannone, N. "Modelling Security and Trust with Secure Tropos," 160-189. *Integrating Security and Software Engineering*. Edited by H. Mouratidis and P. Giorgini. Hershey, PA: Idea Group Publishing, 2007 (ISBN 1-599-04147-2).

[Gordon 2005]
Gordon, D.; Stehney, T.; Wattas, N.; Mead, N. R.; & Yu, E. *System Quality Requirements Engineering (SQUARE) Methodology: Case Study on Asset Management System, Phase II* (CMU/SEI-2005-SR-005, ADA441304). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05sr005.html.

[Graham 2006]
Graham, Dan. "Introduction to the CLASP Process." *Build Security In*, 2006. https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/548.html.

[Haley 2007]
Haley, C.; Laney, R.; Moffett, J.; & Nuseibeh, B. "Arguing Satisfaction of Security Requirements," 16-43. *Integrating Security and Software Engineering*. Edited by H. Mouratidis and P. Giorgini. Hershey, PA: Idea Group Publishing, 2007 (ISBN 1-599-04147-2).

[Heitmeyer 2002]
Heitmeyer, C. "Software Cost Reduction." *Encyclopedia of Software Engineering*, 2nd ed. Edited by John J. Marciniak. New York, NY: John Wiley and Sons, 2002 (ISBN 978-0-471-37737-6).

[Heninger 1980]
Heninger, K. L. "Specifying Software Requirements for Complex Systems: New Techniques and their Application." *IEEE Transactions on Software Engineering SE-6*, 1 (January 1980): 2-13.

[Hubbard 1999]
Hubbard, R. "Design, Implementation, and Evaluation of a Process to Structure the Collection of Software Project Requirements." PhD diss., Colorado Technical University, 1999.

[Leiwo 1999a]
Leiwo, J. "A Mechanism for Deriving Specifications of Security Functions in the CC Framework," 416-425. *10th International Workshop on Database and Expert Systems Applications*. Florence, Italy, Sept. 1-3, 1999. Berlin, Germany: Springer-Verlag, 1999.

[Leiwo 1999b]
Leiwo, J.; Gamage, C.; & Zheng, Y. "Organizational Modeling for Efficient Specification of Information Security Requirements," 247-260. *Advances in Databases and Information Systems: Third East European Conference, ADBIS'99*. Maribor, Slovenia, Sept. 13-16, 1999. Berlin, Germany: Springer-Verlag, 1999 (Lecture Notes in Computer Science Vol. 1691).

[McDermott 1999]
McDermott, J. & Fox, C. "Using Abuse Case Models for Security Requirements Analysis," 55-64. *Proceedings of the 15th Annual Computer Security Applications Conference*. Scottsdale, AZ, Dec. 6-10, 1999. Los Alamitos, CA: IEEE Computer Society Press, 1999.

[McDermott 2001]
McDermott, J. “Abuse-Case-Based Assurance Arguments,” 366-374. *Proceedings of the 17th Annual Computer Security Applications Conference*. New Orleans, LA, Dec. 10-14, 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[Mead 2003]
Mead, N. *International Liability Issues for Software Quality* (CMU/SEI-2003-SR-001, ADA416434). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03sr001.html.

[Mead 2005a]
Mead, N. R.; Hough, E.; & Stehney, T. *Security Quality Requirements Engineering (SQUARE) Methodology* (CMU/SEI-2005-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
http://www.sei.cmu.edu/publications/documents/05.reports/05tr009.html.

[Mead 2006]
Mead, N. R. “The Common Criteria.” *Build Security In*, 2006. https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/239.html.

[Mellado 2007]
Mellado, D.; Fernandez-Medina, E.; & Piattini, M. “A Common Criteria Based Security Requirements Engineering Process for the Development of Secure Information Systems.” *Computer Standards & Interfaces 29*, 2 (February 2007): 244-253.

[Moffett 2004]
Moffett, J.D.; Haley, C.B.; & Nuseibeh, B. *Core Security Requirements Artefacts* (Technical Report 2004/23, ISSN 1744-1986). Open University, 2004.

[Moore 2001]
Moore, A. P.; Ellison, R. J.; & Linger, R. C. *Attack Modeling for Information Security and Survivability* (CMU/SEI-2001-TN-001, ADA388771). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
http://www.sei.cmu.edu/publications/documents/01.reports/01tn001.html.

[NIST 2002]
National Institute of Standards and Technology. “Software Errors Cost U.S. Economy $59.5 Billion Annually” (NIST 2002-10). http://www.nist.gov/public_affairs/releases/n02-10.htm (2002).

[Ortalo 1998]
Ortalo, R. “A Flexible Method for Information System Security Policy Specification,” 67-84. *5th European Symposium on Research in Computer Security – Proceedings*. Louvain-la-Neuve, Belgium, Sept. 16-18. Berlin, Germany: Springer-Verlag, 1998. (Lecture Notes in Computer Science Vol. 1485.)

[OWASP 2007]
OWASP CLASP Project. http://www.owasp.org/index.php/Category:OWASP_CLASP_Project (2007).

[Rosado 2006]
Rosado, David G.; Gutiérrez, Carlos; Fernández-Medina, Eduardo; Piattini, Mario. "Security Patterns and Requirements for Internet-Based Applications." *Internet Research 16*, 5 (2006): 519-536.

[Sabatier 1999]
Sabatier, D. & Lartigue, P. "The Use of the B Formal Method for the Design and Validation of the Transaction Mechanism for Smart Card Applications," 348-368. *FM '99: World Congress on Formal Methods, Vol. I*. Toulouse, France, Sept. 20-24, 1999. Berlin, Germany: Springer-Verlag, 1999. (Lecture Notes in Computer Science Vol. 1708.)

[Schneier 2000]
Schneier, B. *Secrets and Lies: Digital Security in a Networked World*. New York, NY: John Wiley & Sons, 2000.

[Sindre 2000]
Sindre, G. & Opdahl, A. "Eliciting Security Requirements by Misuse Cases," 120-130. *Proceedings of TOOLS Pacific 2000*. Sydney, Australia, Nov. 20-23, 2000. Los Alamitos, CA: IEEE Computer Society Press, 2000.

[Sindre 2002]
Sindre, Guttorm; Opdahl, Andreas L.; & Brevik, Gøran F. "Generalization/Specialization as a Structuring Mechanism for Misuse Cases." *Proceedings of the 2nd Symposium on Requirements Engineering for Information Security (SREIS'02).* Raleigh, NC, October 15-16, 2002. http://www.sreis.org/old/2002/finalpaper6.pdf.

[Soo Hoo 2001]
Soo Hoo, Kevin; Sudbury, Andrew W.; & Jaquith, Andrew R. "Tangible ROI through Secure Software Engineering." *Secure Business Quarterly 1*, 2 (2001).

[Weiss 2007]
Weiss, M. "Modelling Security Patterns Using NFR Analysis," 127-141. *Integrating Security and Software Engineering*. Edited by H. Mouratidis and P. Giorgini. Hershey, PA: Idea Group Publishing, 2007 (ISBN 1-599-04147-2).

[Wood 1989]
Wood, Jane & Silver, Denise. *Joint Application Design: How to Design Quality Systems in 40% Less Time*. New York, NY: Wiley, 1989 (ISBN 0-471-50462-9).

[Woody 2005]
Woody, C. *Eliciting and Analyzing Quality Requirements: Management Influences on Software Quality Requirements* (CMU/SEI-2005-TN-010, ADA441310). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn010.html.

[Xie 2004]
Xie, N.; Mead, N. R.; Chen, P.; Dean, M.; Lopez, L.; Ojoko-Adams, D.; & Osman, H. *SQUARE Project: Cost/Benefit Analysis Framework for Information Security Improvement Projects in Small Companies* (CMU/SEI-2004-TN-045, ADA431118). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tn045.html.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE August 2007	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods	5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Nancy R. Mead	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2007-TN-021
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS	12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

The Security Quality Requirements Engineering (SQUARE) method, developed at the Carnegie Mellon Software Engineering Institute, provides a systematic way to identify security requirements in a software development project. This report describes SQUARE and then describes other methods used for identifying security requirements, such as the Comprehensive, Lightweight Application Security Process, the Security Requirements Engineering Process, and Tropos, and compares them with SQUARE. The report concludes with some guidelines for selecting a method and a look at some related trends in requirements engineering.

14. SUBJECT TERMS information security improvement, misuse cases, requirements engineering, system survivability, requirements elicitation, security requirements, system security, SQUARE	15. NUMBER OF PAGES 35
16. PRICE CODE	

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102