Global Information Assurance Certification Paper

Systems Engineering: Required for Cost-Effective Development of Secure Products

GIAC GSEC Gold Certification

Author: Dan Lyon, danlyon@mac.com
Advisor: Hamed Khiabani

Accepted: TBD

Abstract

This paper shows why and how system engineering should be applied to increase secure design of products. The paper examines research on how system engineering improves security and quality while also reducing development cost and schedule. The examination provides manufacturers with the incentive needed to invest in system engineering.

1. Introduction

Security of data and systems is critical to consider during development of a complex system, and by taking a systems approach, secure design can be achieved in a cost effective manner. This paper overviews a basic understanding of systems engineering, then links that understanding to the domain of information security, and shows the cost effectiveness of a systems approach.

Publications that describe information security along with systems engineering lack the link to the ability of system engineering to manage complexity. Some address this issue from a software security angle, such as Gary McGraw's *Software Security: Building Security In* (2006) and Michael Howard and Steve Lipner's *The Security Development Lifecycle* (2006); however, these publications do not explicitly address information security from a systems approach nor do they describe system engineering's ability to manage complexity. Anderson (2009) states that system engineering is insufficient because it overlooks malice. But malice is simply another perspective the system engineer should use to elicit requirements. This paper fills the gap between malice, complexity and system engineering.

According to Howard and Lipner (2006), security is a subset of quality, but there is no examination of what quality means. Without a true understanding of the definition of quality, how to best improve quality is unclear. This paper defines quality and explores research showing how systems engineering can improve quality. Because security is a part of quality, improving quality will also improve security.

Three studies are presented that showcase the benefit of systems engineering on product quality, schedule and cost. An empirical study by W. Forrest Frantz exhibits significant schedule decrease coupled with increased functionality by taking a systems approach (1995). Research by Eric Honour demonstrates a measurable increase in product quality by increasing Systems Engineering Effort and correlates this to cost and schedule (2004). The impact of this research is that by spending the appropriate amount

on systems engineering, quality and therefore security of the product can be improved most cost-effectively. Barry Boehm's research supports the economic benefit of fixing problems at the requirements and design stages rather than later in the development cycle (McGraw, 2006).

While security requirements are necessary, they are only one aspect of creating a secure system. Johnson (2006) describes how complexity in systems creates emergent behaviors, and McGraw (2006) states that security is an emergent property of a system. These relationships imply that because of complexity, emergent security-related behaviors will be discovered during system development. Because complexity drives the security behaviors, it needs to be managed – a fact affirmed by the International Council on Systems Engineering which found that systems engineering emerged as an effective way to manage complexity (INCOSE, 2010).

Practical application of systems engineering techniques is demonstrated and tied to information security. These techniques provide different views of the system to elicit requirements and manage complexity throughout the development process. Traditional methods are: use cases, interface specifications, data flow diagrams, architecture design and technical leadership (Honourcode, 2011). Each idea is applied to the information security domain.

The connection between complexity, security and quality shows the value systems engineering brings to development of a complex system. Therefore, systems engineering is the most cost effective discipline to ensure security is designed into a system.

2. Introduction to Systems Engineering

Systems engineering is defined by the International Council on Systems Engineering as "a profession, a perspective, and a process" (INCOSE, 2010, p. 7). Each of these qualities is examined and tied back to information security.

According to INCOSE (2010), the systems engineering profession is stated as:

> …an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing, and disposal. Systems engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs. (p. 7)

The definition above contains several key points that relate to information security. Those points are to discover requirements early, consider the complete problem, and provide a quality product that meets user needs.

According to Gary McGraw, designing security up-front is cheaper than when doing it later in the development cycle (2006). Systems engineering is the beginning of the product development process and is responsible for translating user needs into system requirements (Blanchard & Fabrycky, 1998). If systems engineering is the beginning of the development cycle, then including information security as a user need drives security to the front of the product development process.

Systems engineering is focused on solving the problem in its entirety by taking a systems approach (INCOSE, 2010). Such an approach is essential for creating a secure system and is illustrated by the following example. Most people are aware of encryption as a solution for ensuring confidentiality. However, what good is encryption if the private keys are not private? If the key management process is omitted as part of the system, then the entire problem is not being considered. The complete problem is solved only when the technical details of encryption algorithms, key lengths, policies and procedures for management of private keys have all been considered.

Delivering a quality solution that satisfies user needs is the objective of systems engineering (INCOSE, 2010). This point relates because quality is subjective and depends upon perspective (McGraw-Hill, 2005). Quality is more thoroughly elaborated

below, but the outcome is important: to ensure a quality solution, the solution must be considered from the perspective of all stakeholders.

The importance of stakeholder perspective is shown by the problem of authentication in a system where access to a single user system is required at any time by a variety of individuals. Passwords are a common authentication measure in many systems, but passwords do not solve this issue effectively. If a shared password to the system were required, users would solve their need by writing the password down and taping it to the machine, thus ensuring availability of the system to all who required it. In this scenario, the security of the system is breached because user needs for system availability were not balanced with security. Systems engineering provides the framework to balance all stakeholder input to ensure a solution meets the needs of all users (INCOSE, 2010).

Systems engineering processes can be integrated into any development process model, including Waterfall, Spiral, and Vee (Blanchard & Fabrycky, 1998). The Vee process model depicts an example in Figure 1, but all models include the same fundamental concepts of the development lifecycle.

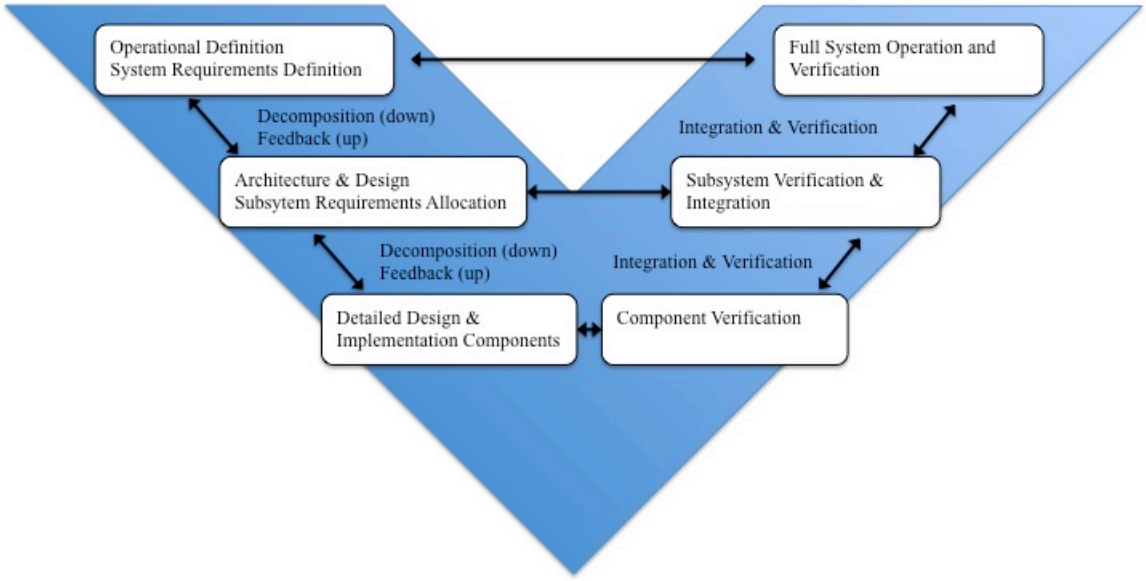

Figure 1 - Vee Process Model

Common attributes for development are depicted in the Vee process model. They are: requirements creation, architecture and design, implementation, testing and feedback loops (Blanchard & Fabrycky, 1998). The activities are worked in iterations to continuously make forward progress in the development of the system while at the same time providing valuable feedback to activities performed earlier (Honourcode, 2011).

3. System Engineering Increases Quality

As described in the previous section, the desired outcome of system engineering is to produce a quality product (INCOSE, 2010). Security is often referred to as a subset of quality (Howard & LeBlanc, 2003), and therefore improving quality can improve security. The relationship between quality, security, reliability and privacy can be viewed in Figure 2 - Quality Attribute Relationship (Howard & Lipner, 2006).

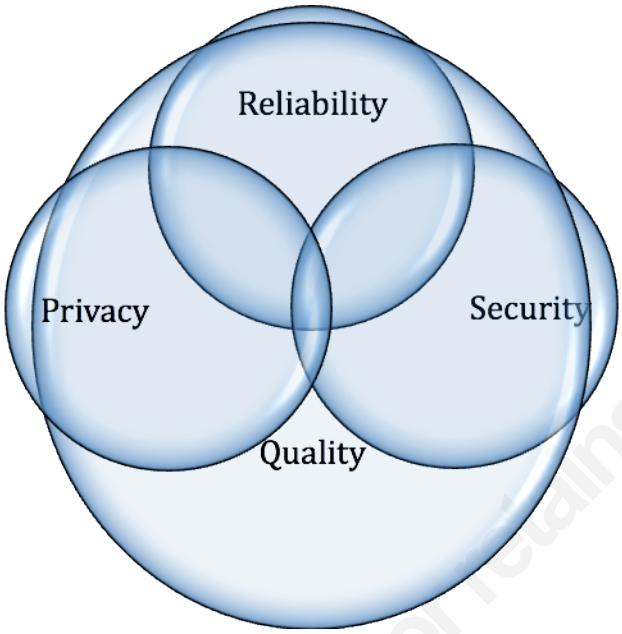

Figure 2 - Quality Attribute Relationship

Because security is a subset of quality, determining how system quality can be improved provides direction on how to increase security. While Howard gives effective explanations on how security is a subset of quality, his work lacks the definition of quality (Howard & LeBlanc, 2003; Howard and Lipner, 2006). What does it mean to have a quality product? To answer this question, the definition of quality must first be understood.

The ISO 25010 (2011) standard on system and software quality defines quality as contextual and subjective based on use of the system. An alternate definition on quality is provided from the domain of economics by E. Scott Maynes as "the subjectively weighted average of characteristics giving rise to utility" (as cited in Terleckyj, 1976, p. 530-531). Subjectivity and utility are important to defining quality because in Maynes research, the consumer decides usefulness of a product.

Further clarification on product quality is provided by three views in the McGraw-Hill Concise Encyclopedia of Engineering (2005). For clarification in this paper, definitions will be used to refer to the different views of the manufacturer and the user.

Manufacturing quality is the view of the manufacturer. This view is focused on the "design, engineering and manufacturing processes" used to create the product (McGraw-Hill, 2005, p. 559). The manufacturer's view is measured by adherence to specifications and standards, and quality improvement is focused on reducing production costs (McGraw-Hill, 2005). This view does not explicitly include the user perception, which leads to the second view of quality.

Performance quality is the view of the user and refers to the level that a product "satisfies their preferences and expectations" (McGraw-Hill, 2005, p. 559). The user will take perceptions into account that may not be related to the functionality of the product but are used in their assessment of quality (McGraw-Hill, 2005).

The view of the product as a system in use is a combination of the manufacturer and user's views (McGraw-Hill, 2005). This view of the system includes the traits that apply and contribute both to the operation and the functionality of the product.

Given the above definitions on user, performance and manufacturing quality, it is possible to have product quality mean different things for the manufacturer and the user. Manufacturing quality is focused on specifications, while the focus of user and performance quality includes subjective measures. The resulting contradictions in meanings present a problem that can only be solved by incorporating user needs, preferences and expectations into the specifications for the product.

If performance and user quality are included with manufacturer quality, then increases in manufacturer quality will benefit both manufacturer and user. By applying this same concept, security of the system in use can also be improved if the system engineering includes information security.

Because manufacturer quality improvement is often focused on reducing costs, it is important to address the cost impact of systems engineering on the product development process.

4. System Engineering Decreases Cost

According to Blanchard and Fabrycky (1998, p. 42-43), experienced engineers recognize that systems engineering reduces the cost of system design and development. Several examples are provided to illustrate the economic benefit of systems engineering.

The first example is provided by Barry Boehm's research on the cost of fixing software defects at various points in the development lifecycle starting with requirements. While this research was not solely in the systems engineering domain, the research directly applies because early generation of requirements is a fundamental activity of systems engineering (INCOSE, 2010). Boehm's data demonstrates that fixing bugs at the requirements and design stages is more efficient by an order of magnitude as opposed to fixing them at the testing and maintenance stages (Boehm as cited in McGraw, 2006). The results are summarized in Figure 3 - Cost To Fix Defect in Stage (Boehm as cited in McGraw, 2006).

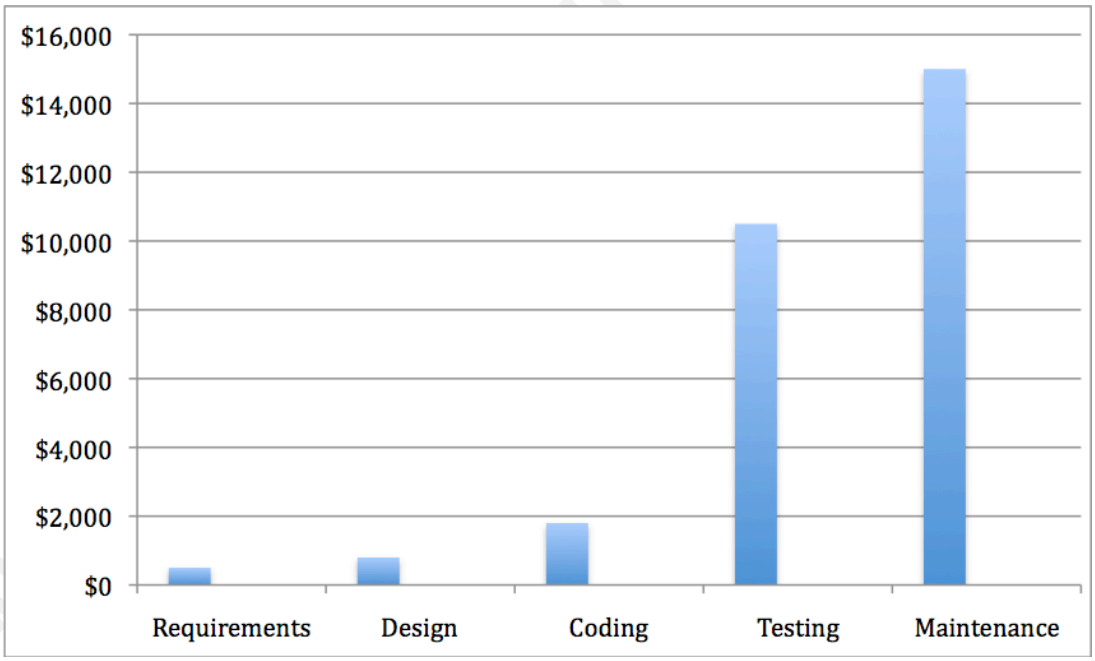

Figure 3 - Cost To Fix Defect in Stage

The second example is an empirical study by W. Forrest Frantz who shows that systems engineering can increase a system's quality and simultaneously reduce schedule

(1995). Frantz's research compared three similar complex projects where the only significant differences in development were the systems management activities (1995).

Each of the three projects developed Universal Holding Fixtures, referred to as UHF, that hold large airplane parts during the airplane manufacturing process. Frantz (1995) described the complexity of the UHF machines as:

> Each UHF can configure to any part shape within its family, float the part on air while it is being loaded, locate the part in X and Y via monuments that pin the part through tooling holes, locate the part in Z via actuated pogos with vacuum cups on end effectors that pull vacuum on the part at heights derived from Engineering CAD drawings, hold the part to less than +0.003", change the vacuum level, and respond to various process commands. (Background section, para. 2)

Other characteristics Frantz (1995) mentions are that each fixture weighed over 10 tons, had about 100 axis of motion, had interfaces to upstream data, required changes to related systems and processes, and cost millions.

Frantz investigated the systems management differences and compared the approaches from all three projects. Frantz' results (1995) show that project duration was 104, 48, and 36 weeks with the systems engineering effort increasing respectively. Two factors that Frantz points out as contributing to the increased success of the projects were the use of systems engineering to increase quality and the use of a systems approach to viewing and solving problems (1995).

Frantz measured quality by the "degree to which all customer requirements are met" (Frantz, 1995, Factors That Reduce Cycle Time section, para. 1). All three projects met the manufacturing process requirements. However, only the shorter duration projects included significant additions in functionality (Frantz, 1995). The shorter duration projects therefore have higher quality by surpassing the minimum set of requirements.

On the two fastest projects a systems approach coordinated requirements generation with all impacted organizations to quickly and more completely define requirements (Frantz, 1995). Frantz' data (1995) clearly shows that increased system engineering effort improved quality and a systems approach reduced schedule duration.

The third example is Eric Honour's "*Understanding the Value of Systems Engineering*". Honour's research shows an increase in quality by increasing systems engineering (Honour, 2004). Honour establishes a Development Quality (DQ) index to measure project quality:

$DQ = 1 / (½ * (Actual\ Cost/Planned\ Cost + Actual\ Schedule/Planned\ Schedule))$.

This formula results in a DQ of 1 for a project on-cost and on-schedule. When the data is plotted against the percentage of project total cost spent on systems engineering effort, an optimum of about 20% of project cost should be spent on systems engineering effort to achieve the best Development Quality (Honour, 2004). Honour then establishes the reported average of 3-8% of project cost spent on systems engineering (2004). The evidence indicates that by raising the amount spent on quality systems engineering effort, a higher quality system can be delivered with better cost, schedule, or both.

Systems engineering has been presented as a concept and shown to be a cost effective mechanism to increase quality and security of a system. The discussion now turns to show that security is a system property.

5. Security Is a System Property

Security as a system property is shown by defense in depth. The defense in depth approach consists of building layer-upon-layer of controls into a system. That way, if one control fails, another control is present to provide protection (Howard & LeBlanc, 2003). When defining the multiple layers for defense in depth, the layers start small, perhaps completely implemented in a given subsystem. The controls eventually cross subsystem boundaries and, by definition, become necessary to consider at the system level.

According to McGraw, security is an emergent behavior of the system (McGraw, 2006). Marsh states that emergent behaviors are produced when elements of a system are combined and a property not inherent in either element is created (2009). These emergent behaviors are displayed by complex systems (Marsh, 2009).

Daniel Geer states that complexity is the enemy of security (2008). If complexity is the enemy, then a higher level of security can be achieved by managing complexity to reduce insecure emergent behaviors. INCOSE says systems engineering "emerged as an effective way to manage complexity and change" in project development (INCOSE, 2010, p. 14). In the ancient Chinese text on warfare, *The Art of War*, Sun Tzu states that to win battles, one must know the enemy as well as the self (Tzu & Giles, 2005). To extend Tzu's thought to the current topic, to win the battle of designing a secure system, one must know security and complexity. The systems engineering profession contains the necessary perspectives and processes to address the emergent behavior of security precisely because the profession already manages complexity and emergent behaviors not related to security.

Combining the systems engineering domain with the information security domain allows security to be designed in and emergent behaviors managed.

6. Impact

At the SANS Rocky Mountain 2012 Conference 2012, John Strand explained his view about how the current state of information security is broken, and new approaches are needed for information security. Many current practices for achieving information security are applied after a product has been developed. Examples such as firewalls, intrusion detection, intrusion prevention, and antivirus are all external systems to what organizations use to conduct business. One different approach is to provide manufacturers economic incentive to adopt a better development process.

The data presented shows that systems engineering both improves quality and reduces cost and schedule. McGraw states that security cannot be "bolted on" or "patched in" to create a secure system (2006, p. 209). However, current security practices include tools that do exactly that. Manufacturers and users will benefit from secure system engineering on products.

Dan Lyon, danlyon@mac.com

7. Systems Engineering Techniques

Concrete examples of applied systems engineering are described to show how traditional techniques can be used to increase security of a product throughout the development process.

Discussion of system engineering techniques starts at the earliest point in the development lifecycle with requirement generation. The goal is to generate correct and complete requirements. If the requirements are correct, complete and proven through verification or validation, then the system under design is shown to meet stakeholder needs.

In addition to being correct and complete, a requirement must be testable and unambiguous (Honourcode, 2011). This concept is illustrated by the realistic but hypothetical requirement "The system shall be secure". The requirement as written is not testable; even a subject matter expert cannot reliably and repeatedly prove this requirement. This lack of proof becomes even more of an issue as the domain knowledge in information security is reduced. A non-security expert who may be doing the actual implementation has a different understanding of what a secure system is than does an experienced penetration tester.

The hypothetical requirement could be rewritten to "The system shall be certified to FIPS 140-2". This wording change produces a requirement that is then verifiable and unambiguous. The test method is clear, and the results are not open to interpretation.

A technique for discovering and specifying requirements is use cases, and Alistair Cockburn describes a use case as something that describes a system's behavior under various conditions as the system responds to a stakeholder request (2000). The same technique is easily used to represent a malicious goal by changing perception and is termed an abuse case by McGraw (2006). By incorporating the attacker as an additional actor in the use case requirements elicitation process, more complete requirements can be generated as use and abuse cases often interact (Alexandar, 2003).

To model abuse cases successfully, McGraw states that a security subject matter expert should be included up-front in the requirements process (2006). This method

brings the discovery into the most cost-effective time of the development cycle (McGraw, 2006).

The following example depicts a few abuse cases for an online banking system where a customer desires to manage their money. Normal use cases are represented by white ovals and abuse cases are represented by black ovals. Derived system requirements included are in blue, and represent additional requirements on the system. The goal of the user is to "Manage Money" and this goal includes both "Log In To Website" and "Keep Money". The goal of the attacker is to "Get Money", which can be done in a variety of ways such as "Steal UID/Password", "Steal Identity" or "Sell Cracked Passwords".

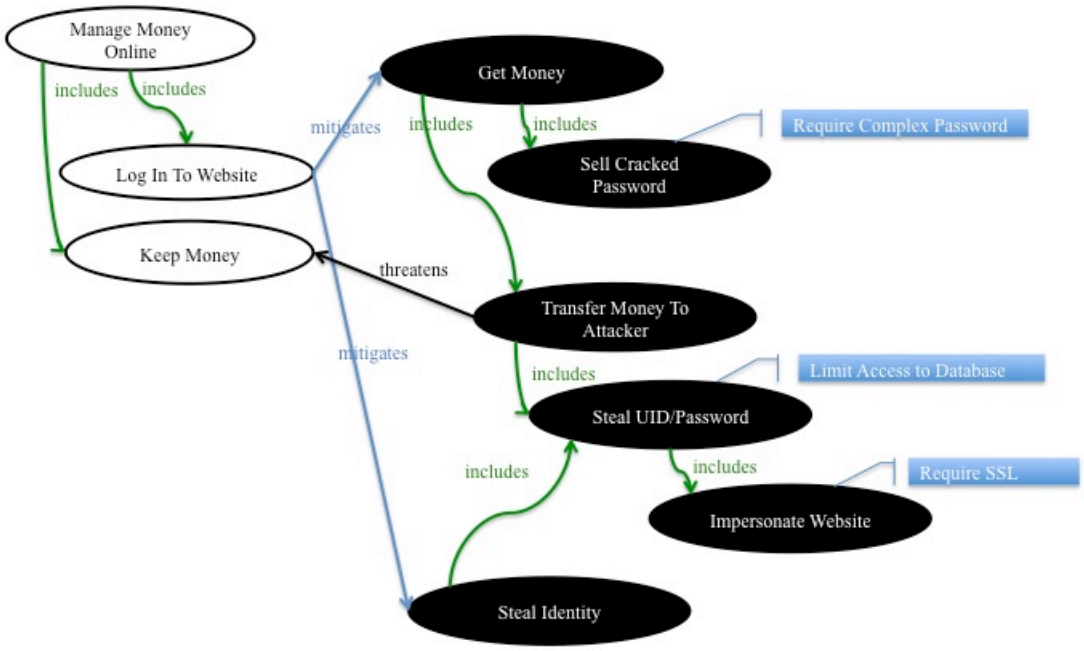

Figure 4 - Use Cases, Abuse Cases and Derived System Requirements

This example is used to illustrate the application of four system's engineering concepts.

Dan Lyon, danlyon@mac.com

First, the example highlights the notion of changing perception. Systems engineers are required to routinely change perception of the system as they perform requirements analysis (Honourcode, 2011). System engineers need only change perception of the system from normal to malicious to account for abuse cases.

Second, the example shows how a system is viewed in relation to a user function. System engineers view the system in terms of meeting a user need (INCOSE, 2010). The users of this system expect to keep their money. If the users do not keep their money, they will not use the system.

The third aspect the example reveals is that user needs must be balanced across the system. System engineers strive to balance requirements (INCOSE, 2010). One sure way the bank could satisfy the user need to keep money would be to not allow online account management, but this restriction conflicts with the primary user need.

The fourth illustration the bank example provides is that as Cockburn (2001) notes, use cases are some but not all of the requirements. Additional requirements exist outside of what the user can do with the system as shown with the example of "Require Complex Password". Most bank users are not security experts and cannot express the desire for complex passwords or authentication protocols. The function of the systems engineer must address the user desire to keep money, and create additional requirements that satisfy that user need.

Use cases can be as formal or as informal as needed (Cockburn, 2001). Abuse cases can be represented in the same manner as use cases, although some additional information may be desirable as described by Sindre and Opdahl. The main benefit for abuse cases is the brainstorming with stakeholders and subject matter experts (McGraw, 2006). This brainstorming allows the security considerations to be accounted for and balanced like any other system need.

Another tool frequently used in systems engineering is a data flow diagram (Honourcode Inc., 2011). The data flow diagram can show how data flows through a system. Figure 5 - Simple Data Sharing Across Internet illustrates a simple data sharing service across the Internet. In this example data flows from the user's computer on the left to the user's computer on the right using a server in the middle. This view highlights

where the data exists, therefore, allowing questions about it to help elicit requirements. Is the data encrypted across the Internet? Is the data encrypted on disk and in memory? Exploring the data flow also reveals the interfaces between subsystems.

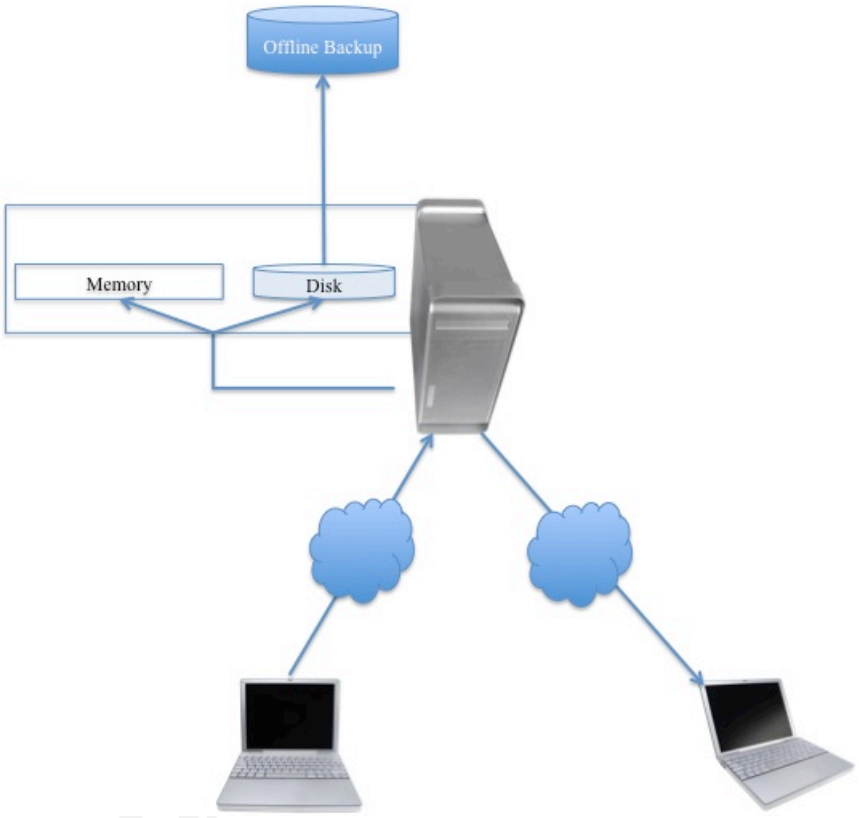

Figure 5 - Simple Data Sharing Across Internet

Interface specifications are commonly used in complex systems where components or subsystems interact (Honourcode, 2011). Interface specifications provide the opportunity for requirements on how subsystems react to external events and can be combined with the idea of chokepoints.

Chokepoints are defined as those points in the system where data has crossed a trust boundary and must be validated (Howard & LeBlanc, 2003). Chokepoints can be combined into the interface specification, thus creating security requirements for data input validation as well as the responses to improper data. These chokepoints provide not

only protection from bad data, but also the opportunity to specify proactive controls for a subsystem by using higher system level knowledge of behavior and interactions.

The following example illustrates how chokepoints improve security systems. A web service accepts user input. The interface specification could be written with requirements that define only the characters allowable. The interface requirements could also specify that if disallowed characters enter the system from the same IP address more than 3 times, then accept no further connections from that IP address. This provides an opportunity for products to define normal behavior and subsequently react to abnormal behavior, rather than relying on an external system like a firewall.

System architecture definition is another activity of systems engineering (Honourcode, 2011) driven by customer needs (Blanchard and Fabrycky, 1998). This can be illustrated in the common functionality of sharing a computer screen across the Internet. The following example from Figure 5 - Simple Data Sharing Across Internet illustrates two potential solutions that address the user need to share data across the internet.

Potential Solution	**User Need**	**Potential Architecture**
Solution A	User wants to show screen to remote party.	Users run software to connect through server to share data. Users share a generated token via external mechanism (e.g. – phone or chat)

Solution B	User wants to show screen to remote party. Server owner wants to restrict access.	Users are required to login. Users run software to connect through server to share data. Users share a generated token via external mechanism.

Table 1 - Description of Potential Solutions to Simple Data Sharing Across the Internet

Solution A and B both allow the user to accomplish their goal of sharing data. However, Solution B identified an additional user need of restricting access. Identifying this user need allows that conceptual design to evaluate various options.

Honourcode (2011) describes technical leadership as a process used to plan and guide a product's development as well as to manage relationships. The process of technical leadership spans the entire development process, as shown in Figure 6 with the "Vee" process model.

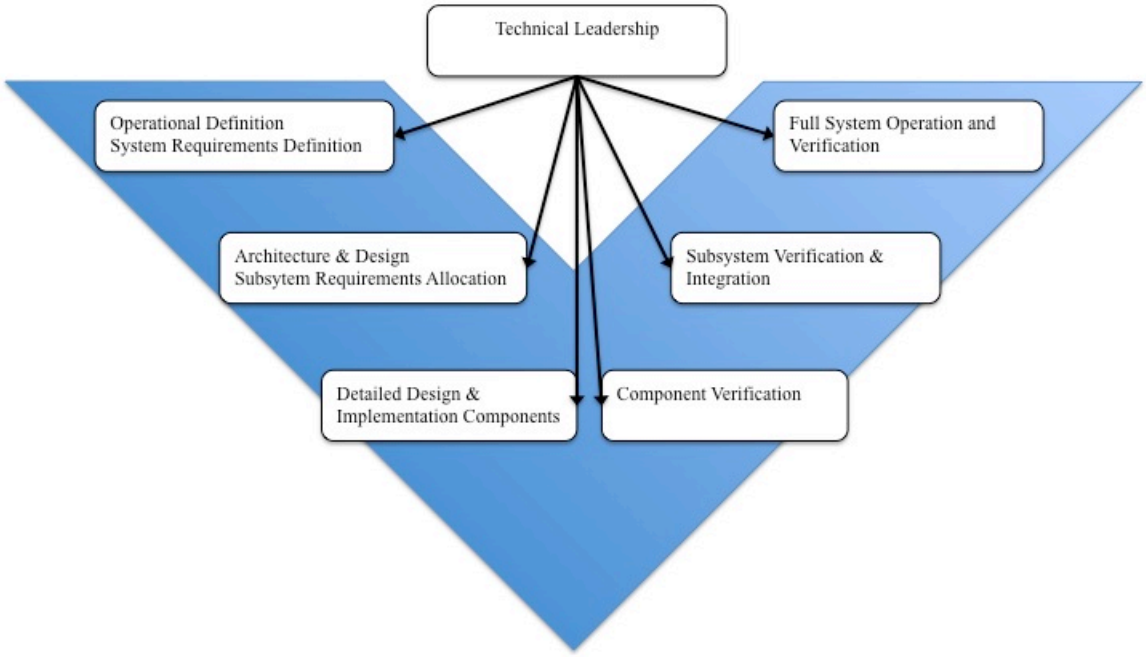

Figure 6 - Technical Leadership Spanning the "Vee" Process Model

Motivating the development team is essential to correct implementation of requirements, and this is part of technical leadership (Honourcode, 2011). Anderson also notes that it is essential to get the team caring about security (2009). This is fertile ground for security considerations where engineers have varying degrees of familiarity and concern with information security. The job of the systems engineer is to be a technical leader in order to communicate the importance of information security controls and motivate the team to implement the controls correctly.

Further technical leadership activities are the skills of conflict resolution and developing relationships (Honourcode, 2011). These soft skills are important to ensuring the overall security of the system because security is an attribute that can be viewed as unnecessary. If that is the view of a manager responsible for cost and schedule, then to that manager, the impact of security is to increase cost and schedule without providing

value. The systems engineer must be able to effectively educate and work with those unaware of the benefits of including security into product design.

8. Conclusion

This paper describes the connection between complexity, security and quality and the benefits that system engineering provides. It also shows the value systems engineering provides. By investing in system engineering, manufacturers can create higher quality products for less time and money. Manufacturers must adopt system engineering that includes information security to create secure systems.

9. References

Alexandar, I. (2003). *Misuse cases use cases with hostile intent.* Retrieved from http://easyweb.easynet.co.uk/iany/consultancy/misuse_cases_hostile_intent/misuse_cases_hostile_intent.htm

Anderson, R. (2009). *Security engineering, a guide to building dependable distributed systems*. (2nd ed.). Indianapolis: Wiley.

Blanchard, B. and Fabrycky, W. (1998) *Systems Engineering and Analysis*, 3rd Ed. Prentice Hall International Series in Industrial and Systems Engineering, USA.

Cockburn, A. (2000). *Writing effective use cases*. (1 ed.). New York, NY: Addison Wesley.

Frantz, W. Forrest (1995) "*The Impact of Systems Engineering On Quality and Schedule *Empirical Evidence** Retrieved May 2, 2012, from http://libsys.uah.edu/library/incose/Contents/Papers/95/9513.pdf

Geer, D.J. (2008). *Complexity is the enemy.* Retrieved June 24, 2012 from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04753682

Honour, E.C. (2004) *Understanding the value of systems engineering*. Retrieved May 2, 2012 from http://www.incose.org/secoe/0103/ValueSE-INCOSE04.pdf

Honourcode, Inc. (2011) *Applied systems engineering, a practical course in the discipline of defining and building complex systems*

Howard, M., & LeBlanc, D. (2003). *Writing secure code*. Microsoft Pr.

Howard, M., & Lipner, S. (2006). *The security development lifecycle*. Redmond, WA: Microsoft Press.

INCOSE. (2010). *Systems engineering handbook* v3.2

ISO/IEC 25010. (2011). *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.*

Johnson, C. W. (2006). *What are emergent properties and how do they affect the engineering of complex systems?* Informally published manuscript, Department of Computer Science , University of Glasgow, Glasgow, Scotland. Retrieved from

http://www.dcs.gla.ac.uk/~johnson/papers/RESS/Complexity_Emergence_Editorial.pdf

Marsh, G. E. (2009). *The demystification of emergent behavior*. Retrieved from http://arxiv.org/abs/0907.1117

McGraw, G. (2006). *Software security, building security in*. Addison-Wesley Professional.

McGraw-Hill (2005). *McGraw-hill concise encyclopedia of engineering*, McGraw-Hill.

Sindre, G., Opdahl, A. L. (n.d.). *Templates for misuse case description*. Retrieved May 25, 2012 from website http://swt.cs.tu-berlin.de/lehre/saswt/ws0506/unterlagen/TemplatesforMisuseCaseDescription.pdf

Maynes, E.S. (1976). The concept and measurement of product quality. In Terleckyj, N (Ed.), *Household production and consumption* (pp. 529-584). Retrieved from http://www.nber.org/books/terl76-1

Tzu, S., & Giles, L. (2005). *The art of war*. El Paso Norte Pr.