Decision support for managing security complexity
in software development

by

Derek Kenneth Church
B. Sc., University of Victoria, 2000

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCES

in the Department of Computer Science

University of Victoria

Decision support for managing security complexity
in software development

by

Derek Kenneth Church
B. Sc., University of Victoria, 2000

Supervisory Committee

Dr. Jens Weber, Supervisor, Department of Computer Science

Supervisor

Dr. Micaela Serra, Department of Computer Science

Departmental Member

Dr. Daniel German, Department of Computer Science

Departmental Member

Dr. Stephen Neville, External Examiner, Dept. of Electrical and Computer Engineering

Outside Member

Supervisory Committee

Dr. Jens Weber, Supervisor, Department of Computer Science

Supervisor

Dr. Micaela Serra, Department of Computer Science

Departmental Member

Dr. Daniel German, Department of Computer Science

Departmental Member

Dr. Stephen Neville, External Examiner, Dept. of Electrical and Computer Engineering

Outside Member

Abstract

Security in software applications is a growing concern. This is evidenced by the increasing number of media articles, which detail money lost and the invasion of privacy that occurs, when the security vulnerabilities of a software application are exploited.

As a new and rapidly evolving field, the issue of security in software is still an open area of research. The most common methods consist either of A) a reactive survey where subjective determination is made of the level of security present in the software or B) an attempt to integrate security issues into the software development process. The least costly and arguably more effective approach is the latter. However, current approaches rely heavily on the presence of experts in the domain of security to both identify and resolve the issues. Such experts are not always available for each software development project, creating a problem for a project team needing to inject security into their process.

This thesis represents an approach for realizing the construction of a decision-support tool for injecting security into an existing process model. Current literature involving security is examined, and the information gleaned is used to construct a security ontology using grounded theory. A framework is then created that maps into existing software process models and the framework loaded with information from the ontology. A practical analysis using the framework is done by examining the TAPAS project in order to revise the tool, with the goal of increasing usability. This analysis is also used to determine if the tool can identify areas of opportunity with regard to security in the TAPAS project.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I wish to thank my parents, my daughter, my brother and his family, for their support, encouragement and love, without which I could not have made it this far. This work is as much theirs, as it is mine. I also wish to acknowledge Dr. Jens Weber, my professor, and Dr. Micaela Serra, my advisor, for their support, patience and guidance. It was a long road.

Dedication

To Kelsea – who made it all possible.

Chapter 1 - Introduction

1.1 Motivation

Information technology and the accompanying software are now ubiquitous. The explosive growth of computing in the last 30 years, along with the introduction of low cost world wide network communication, has placed microprocessors and software in devices ranging from cell phones to toasters. Following this trend in parallel has been growth of computer crime. In 2005 the Federal Bureau of Investigation completed a computer crimes survey[38] which states that dealing with viruses, spy-ware, PC theft and other computer related crime cost U.S. businesses $67.2 billion in 2005 (in comparison to this, loss due to telecommunication fraud is listed at only one billion dollars for the same year). The most significant part of this loss comes from worms, viruses and Trojan horses, followed by computer theft, financial fraud and network intrusion. The survey notes that this information does not include the current cost of both the labour and tools used to attempt to prevent these incidents, which would drive the true cost even higher. A question clearly generated by these numbers is "what is being done or can be done to reduce these losses?"

The worms, viruses and Trojan horses contributing the most to the cost of computer crime are examples of tools designed to take exploit vulnerabilities existing in software applications. These vulnerabilities range widely depending on the flaw to be exploited: from the buffer overflow exploit which takes advantage of a defect in software that allows an attacker to load their own code, to searching online databases attempting to discover holes in security through which a program can gain access, and the embedding of malicious code in benign software that consumers are about to use. The number of programs written to exploit these security holes in the software continues to increase. A study completed by Kaspersky Labs[25] on the problem found that by the end of the 2005 analysts were detecting 6368 malicious programs per month. Over 2005 alone there

was a 117% rise in the number of programs they detected on a monthly basis. These numbers suggest that there is a urgent need to do more to curtail security vulnerabilities in software applications.

The field of security in computing is still emerging as a discipline, much akin to the field of software development itself, which is arguably still in its infancy. Unlike other fields of engineering which can draw on decades or even centuries of experience with constant physical laws yielding a static environment, the engineering of software has undergone evolution again and again. The hardware on which it runs continually evolves, the programming languages change and the programming paradigm itself is altered by the constant introduction of new operating paradigms (i.e. the internet, object oriented programming and multiprocessing). This alters not just the operating environment of the program but also its behavior and how it will be used. This means that there are new issues emerging continually regarding security in the field of computing and also that techniques for dealing with security in software applications are still being developed.

Currently the two most common methods of achieving security in software operate at two different points in the software development lifecycle. The first can be generally categorized as a *reactive technique*. Forms consisting of lists of questions are completed for software in the final stages of development and this if done to ascertain whether or not the application can be considered secure. These questions often consist of high level abstractions that do not cite specific instances of vulnerabilities but rather ask the evaluator to make a recommendation on whether a specific policy or procedure has been adhered to in the software. Though useful, this can be a costly process both in terms of time and production as any issues identified can force large parts of the software project to undergo redevelopment. It is a truism in the field of software development that the earlier in the development a required change to code or design is detected, the lower the cost of fixing the problem. The second method is intended to address security by incorporating security concerns into all aspects of the software construction process. Two examples of this are the AEGIS[12] framework and the Microsoft Trustworthy Security Development Lifecycle[22]. In both a strong reliance is made on the presence of an expert in the domain of software security who participates in the process. These experts have years of experience which allows them to identify security concerns or

issues. Smaller software development houses consisting of a few or even one developer who do not have access to the aforementioned security experts cannot employ these methods. This is a problem for smaller development teams as security quickly becomes a major consideration in software applications. This need for security is evidenced by the ever-increasing number of media articles covering the topics. These articles detail the financial and personal cost of programs which have been compromised due to security vulnerabilities, and illustrate the need to incorporate security issues and concerns in every software project. However, development teams with limited resources cannot afford to stop work on software projects for an indefinite period as they gain the expertise and experience necessary to incorporate security considerations into their development processes. It is also unlikely that they can afford to purchase the services of an external expert in security to participate in the project.

Based on this assessment there is a clear need for a mechanism or methodology or both that allows developers to integrate security considerations into their current development process at a relatively low cost and effort. It is important to note that when speaking of cost we include the time taken to learn and apply the methodology as well as taking into account that, though the developers may be competent at the creation of software, they may also be unaware of the issues of security in software and so it would be inappropriate to assume their familiarity with the field of security. This must be taken into account by any proposed mechanism.

The intent of this thesis is to propose such a mechanism and methodology as a solution to this issue by constructing a decision support tool for managing security complexity in software applications. The problem domain where this methodology would best fit would be in the generalized domain of service oriented software applications (e.g. office productivity software, desktop publishing, etc.). In its current manifestation, it is not directed towards systems software (e.g. operating systems). It would be possible however to encode information for specialized problem domains, so it is not limited to general application.

1.2 Challenges presented

There are several challenges to the construction of the Software Security Decision support tool:

Integration into the Software Development Process: There is no one dominant method of developing software present today either in industry or academia. Any tool whose intent is to facilitate the injection of security concerns into the Software Lifecycle will need to map into a variety of different process models to prove effective.

Minimize the learning curve: As mentioned, most developers cannot stop making their software for long periods in order to receive training and gain experience in the application of security. This needs to be addressed as a practical concern and a means discovered to leverage the existing knowledge, skill or methodology possessed by developers today in order to minimize the learning needed to apply the tool.

Representation and Organization of Security Concerns: The decision support tool will need to: A) provide information on security issues B) provide a mechanism to navigate the issues to determine applicability and C) provide options and information for different security mechanisms. These security issues will also need to be gathered and organized for injection.

Extensibility and Flexibility: There are two dominant facts that are prevalent based on any survey of evidence in today's world of computer security. The first is that the field of security will continue to change and evolve at a rapid rate in the foreseeable future and the paradigms in which security issues are involved with software development will parallel this change and evolution. The second is that any tool developed must allow for the continual update to information and processes encoded within the tool and as also must be open to evolution in the usage of the tool to account for the introduction of new software development processes.

A decision support tool must provide mechanisms to account for all of these. It should provide wide coverage of security aspects in software, be easily updated as new exploits/techniques/information are made available, facilitate and highlight areas of concern in security and, where possible, indicate resources for further research. In addition it should allow a developer to make use of the tool at any stage of the development process, and provide incremental evaluation of security allowing the

developer to choose the level of detail which they want to achieve in the evaluation process. The tool should also highlight what is risked by not going further, so that the developer can make appropriate decisions.

1.3 Approach

A survey of the work in the field of computer security with regard to software is necessary in order to lay a foundation for the design and implementation of a decision support tool. The survey needs to take into account both the theory-oriented aspects as well as practical application of security techniques.

There is a variety of resources available dealing with the topic of security considerations in software development. The theoretical background and research is available through academic publications such as conference papers and textbooks, interviews with prominent members of the Security community that have been posted to the web and other information web sites covering current security flaws and issues and also current news articles posted by the general media. For a practical insight into the application of security in the development lifecycle a recently completed software project (TAPAS) is examined.

To validate the information collection and organization the technique of *grounded theory* is applied in the capture of the data. Grounded theory[15] advocates the collection of data in a research area first and then deriving theory from the data second. Using an open source visual modeling tool called CMAP, the data is entered on the map to construct an ontology (see Figure 1.1-to view a large version see Appendix A)

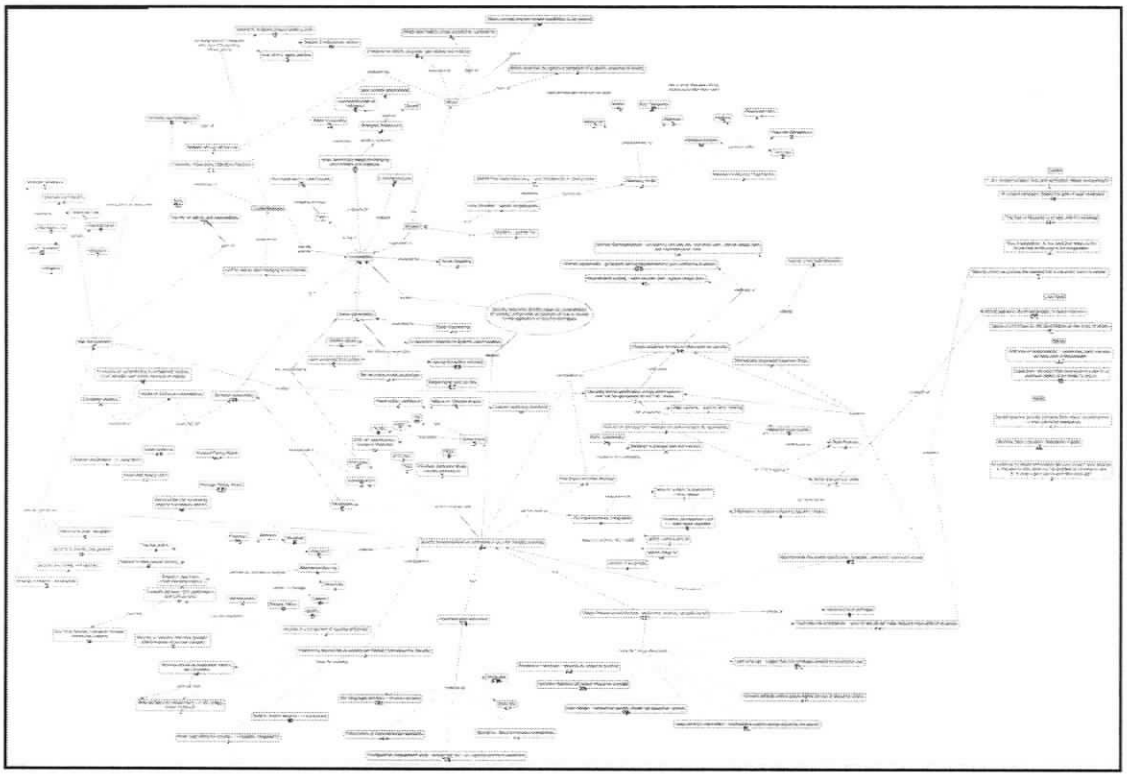


Figure 1.1 Ontology map constructed using the CMAP tool

The next step is to derive a methodology for security analysis for a software application based on the ontology. This first involves construction of a framework that will overlay a given software development process allowing for the identification of security issues at the appropriate point in the development lifecycle. The framework will then be injected with the relevant security issues identified by the ontology. This is used as a starting point for a practical analysis involving the TAPAS software project.

By analyzing the TAPAS project with more detailed passes the prior derived methodology is revised to reflect a framework that can be implemented in the decision support tool and be usable in the software development process. An added benefit of this is that attempting to use the tool for an analysis of the TAPAS project may find security issues in TAPAS that have yet to be identified.

1.4 Overview

The following chapters are structured as follows: Chapter 2 describes the related work which influence and direct this thesis. Chapter 3 comprises an introduction to the concepts and methodologies required, including decision support systems, software

development process models, security in computing and grounded theory. Chapter 4 covers the development of a process model named the *Security Spiral* and shows how it can map into current software process models. This chapter also covers the selection of software to support this tool and how the software can be used in the process. Chapter 5 introduces the use of grounded theory in the construction of ontology to represent a theoretical analysis of security with regard to software development, and presents a tool that is used to visualize the ontology. In Chapter 6 a practical analysis involving the TAPAS software project is explored and used to create a workflow process using the tool in a practical setting as well as to aid in identifying security issues in TAPAS. Finally, Chapter 7 gives a summary of this work and points to future research work as well as issues surrounding the decision support tool.

Chapter 2 - Related Work

2.1 Introduction

It should be noted that there are several overlapping efforts with this work that are intended to support security in the realm of software development. These efforts fall into a wide categorization, ranging from targeting specific areas of the software process to more generalized survey tools and in addition include methods for injecting security in the development lifecycle. The AEGIS framework is discussed in section 2.2; AEGIS is an engineering approach integrating security into the development process through the use of UML notation and stakeholder involvement. Section 2.3 introduces a software tool called ASSET developed by the National Institute of Standards and Technology intended to support self-assessment of security concerns in software projects. Section 2.4 discusses the *Security across the Software Development Lifecycle* Task Force report created by the National Cyber Security Partnership (NCSP). Section 2.5 describes the Trustworthy Computing Security Development Lifecycle, a process adopted by Microsoft. Section 2.6 details a tool, also developed by Microsoft, which is the work related closest to this effort. Lastly in section 2.7 we review the *RUP* software process, which illustrates work done on providing visualization and navigation of the process model during software development.

2.2 The AEGIS case study

AEGIS, described in [12], is a methodology for creating secure software systems through the modeling of asset's and contextual risk analysis. It incorporates security issues by adding UML constructs to the modeling process. This has a two-fold effect: it increases the visibility of security during development and also directly incorporates security concerns into the design documentation and workspace. Every decision involving security is weighed based on an analysis of the assets of the system, which focuses on the potential exposure of these assets. Meetings between the developers and the stakeholders involve a scenario play, where the developers foreshadow the compromise of an asset to the stakeholder and then investigate the potential consequences of this action with the stakeholder, deriving the security requirements from the stakeholders' response to the scenario. Once completed a risk analysis can be performed

to determine the cost-benefit of various responses, which can then again be reviewed with the stakeholder for determining which is considered an appropriate response, or conversely what the potential cost or exposure would be of not implementing a response. This methodology relies heavily on the presence of a third category of participant, that of the security expert. This role is critical in the risk analysis and security design phase in that all knowledge of security issues is assumed to come from this source, if the developers do not already have the requisite technical security knowledge.

2.3 NIST and ASSET

In the United States, the National Institute of Standards and Technology (NIST) host's the Computer Security Resource Center (CSRC) through their Computer Security Division. The purpose of the CSRC is to improve security in the field of information security by providing technical documents, vulnerability advisories and other resources that are pertinent to the field.

The ASSET software application begins with the NIST special publication technical document, NIST SP 800-26. SP 800-26 is the Security Self-Assessment Guide for Information technology Systems. Created in consultation with government and Industry and building on the United States Federal IT Security Assessment Framework, this document provides a questionnaire that can be followed to provide an analysis of an existing software application with regard to security issues, and potentially identify areas of concern with regard to security. Building on this the contents of the SP 800-26 document were integrated into a software application that would facilitate use of the questionnaire. This tool is called ASSET (Automated Security Self-Evaluation Tool). Written in Java to be run on a machine with the Windows 2000 operating system as its base operating system, ASSET provides an interactive tool for a self-assessment of a software project. It allows for the creation and modification of reports based on the self-assessment and the criteria it lists from the SP 800-26 document. It is important to note that the tool does not perform an analysis of any kind, and its reliability is based solely on the information entered by the ASSET user.

2.4 Improving security across the Software Development Lifecycle NCSP task force report

Following the release of the 2003 White House National Strategy to Secure Cyberspace and the National Cyber Security Summit, the National Cyber Security Partnership (NCSP), a public-private partnership, was established to develop shared strategies and programs to better secure and enhance America's critical information infrastructure. Consisting of five task forces that included security experts across the IT industry their mandate was to develop work products to identify and assist with security concerns in the realm of computing. One particular taskforce, named *Security across the Software Development Lifecycle*, published a report April 1, 2004, detailing their findings, ideas and recommendations. This taskforce created four subgroups, each of which had the following mission[1]:

- enhancing the education and training of present and future developers to put security at the heart of software design and at the foundation of the development process (Education Subgroup)
- developing and sharing best practices to improve the quality of software, as well as the process so that systems are more resilient to attack (Software Process Subgroup)
- creating incentives that can create a culture of security awareness, and disincentives for malicious behavior (Incentives Subgroup)
- making the patching process simple, easy, and reliable (Patching Subgroup)

The recommendations of the subgroups were integrated into the report referencing short, mid and long term goals all designed to improve security in software development. It is important to note that these recommendations dealt with broad issues in the field as opposed to focusing on specific instances, for example, a short term recommendation from the Education subgroup stated "Adopt software development processes that measurably reduce software specification, design and implementation defects", without specifying which development process is preferred or what metrics can be developed to provide the aforementioned measurement.

2.5 Microsoft's SDL

Microsoft's SDL[22] (trustworthy computing Security Development Lifecycle) is a software process model that Microsoft introduced for use when developing software applications which were identified as being possible targets for malicious attack once deployed. It adds security checkpoints into each segment of the development process including written artifacts, security code reviews, the use of static analysis tools and the development of threat models. Each phase of the normal development process is paralleled by a similar phase of the 'security in the software' development process. A key part of this initiative is the role of the security advisor. Assigned to the project in the inception phase the security advisor co-ordinates and guides the project at each step of the development process, contributing insight and expertise into security issues and the steps necessary to resolve them. This places a heavy reliance on the presence, and participation, of a domain expert in the field of security in the role of the security advisor.

2.6 Microsoft's Threat and Analysis Tool

Microsoft has released a software product, named the Microsoft Threat Analysis and Modeling tool, which overlaps with some aspects with this work. It is targeted towards non-security experts for use in their software development projects, which is the same audience as targeted by the security spiral. The main focus of the software, however, is the construction and resolution of threat models in the domain of security in software application development. The tool itself is intended to run on Windows platforms and is not open for extension. It relies on forms for its data collection and interaction with users.

2.7 The RUP software product

The RUP (Rational Unified Process) is a software engineering process framework developed and marketed by Rational Software[21]. There are a variety of tools that fall under the RUP product umbrella, but the one most relevant to this work is the Rational Method Composer software product. Its purpose is to provide a tool that visually guides a developer through the Rational Unified Software Development process during the production of their software. It includes a wide variety of resources, both online and

referenced, which deal with all aspects of software development, and places these resources in the relevant timeline of whichever software process model is being applied. It allows for the tailoring of the provided process model so that it can be customized to a particular circumstance or project, and then further allows for integration with other products from the Rational suite so that the methodology created in the Method Composer product can be directly integrated with the actual development of the software project itself.

Chapter 3 - Background

3.1 Introduction

The focus of this research is to facilitate the integration of security issues into software development. The foundation needed to support this effort consists of five main parts. To begin a review of the role of security in computing is needed, in order to introduce the field of security and relevant terminology. Next an overview of the current paradigms of the software development is done, as this is the context into which we are injecting the security material. Key questions of how to place the security issues into the software development lifecycle, how these materials should be organized and presented are next. In order to assist in these decisions of placement and presentation a survey is done of the types of decision support structures and methodologies available. This, combined with the software development processes, provides the basis of the formulation of the decision support tool meta-structure and implementation that is performed in the following chapters. Finally a means of organizing the relevant security material prior to its injection motivates a section on Grounded Theory, which is the technique used to perform a systematic review and organization of the material. This ontology will then be used in the theory based phase of construction in Chapter 5.

3.2 Security in Computer Science

Security in Computer Science is not a new issue and in fact begins with the advent of computers themselves. Actual working physical machines representing the ancestral forbears of today's machines came into being during the Second World War and the decade that followed. The relevance of starting this far back in the history of computing comes with the observation that at the beginning security in any form was anchored firmly in the notion of physical access, and from this point on the evolution of computer systems and software created the emergent properties (i.e. problems of security) that represent such challenges in today's systems. In the 1940's and 1950's machines such as the UNIVAC I filled rooms to which physical access was controlled[10]. Programs were created on punch cards by programmers and dropped off to be executed by the machine operator. The results of the program run were then picked up later in an output box. In

these cases security involved recognizing that the output would require more protection than was offered by a pickup box. The machine operators could also introduce unauthorized changes if two-person control or separation of duties was not enforced as a security measure. In all these cases the computer in question would literally reload an operating system, the program to run and any required data for each job run executed by the system. No other programs would run on the system at the same time, and there was no direct access by anyone to the running program. The introduction of resident operating systems, multi-tasking, communication networks and shared resources in the 1950's and 1960's drove the emergence of security concerns beyond that of physical access.

The first recorded academic paper raising computer security[6] as an emerging problem was the R609-1 Rand Report published in February 1970 entitled *Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security.* It was produced by the Rand Corporation for the US government (though at the time it was rated confidential). The taskforce, created in 1967 and chaired by Willis Ware, then Chairman of the Rand Corporation, produced an overview of the field of security in computing and also provided a list of conclusions and recommendations. The seven stated conclusions were:

1. Providing satisfactory security controls in a computer system is in itself a system design problem. A combination of hardware, software, communication, physical, personnel, and administrative procedural safeguards is required for comprehensive security. In particular, software safeguards alone are not sufficient.
2. Contemporary technology can provide a secure system acceptably resistant to external attack, accidental disclosures, internal subversion, and denial of use to legitimize users for a *closed environment* (cleared users working with classified information at physically protected consoles connected to the system by protected communication circuits).
3. Contemporary technology cannot provide a secure system in an o*pen environment*, which includes un-cleared users working at physically unprotected consoles connected to the system by unprotected communications.
4. It is unwise to incorporate classified or sensitive information in a system functioning in an open environment unless a significant risk of accidental exposure can be accepted.
5. Acceptable procedures and safeguards exist and can be implemented so that the system can function alternatively in closed environment and in an open environment.

6. Designers of secure systems are still on the steep part of the learning curve and much insight and operational experience with such systems is needed.
7. Substantial improvement (e.g. cost, performance) in security controlling systems can be expected if certain research areas can be successfully pursued.

For the purposes of recommendations and guidelines in the report it was assumed they were dealing with the most difficult security control situation of the time: a time-sharing multi-access computer system serving geographically distributed users and processing sensitive information. Even at this early point in the history of computing the complexity of security in computing was noted, as the following citation from the report illustrates:

> *Thus, the security problem of specific computer systems, must, at this point in time, be solved on a case-by-case basis, employing the best judgment of a team of system programmers, technical hardware and communication specialists, and security experts[40].*

This one report spawned a flurry of activity in the 1970's for government sponsored research and recognition of security in computing. This laid the foundation for much of what we have now. David Bell and Len Lapadula's work with the Multics operating system[4] in the 1970's led to the creation of the Bell-Lapadula model which is still in use today. It defines a set of classifications for dealing with confidentiality as well as a set of properties that must be upheld in order for the classifications to be valid. Other avenues of exploration and work included secure operating systems, cryptography, evaluation criteria for trusted computer systems, threat modeling, vulnerability classification and analysis, to but name a few. All of these areas contributed to the following overview of the field.

3.2.1 Confidentiality, Integrity and Availability

At a basic level the relevant security concerns with regard to a computer system revolve around two areas. The information or data housed in a system and the resources provided by the system. Confidentiality, integrity and availability form a tripod of security services when considering the broadest desirable characteristics for these two things. Bishop in [7] refers to these three as the basic components of computer security.

Confidentiality in definition refers to the concealment of information or resources. Whether referring to secure military communications of the national importance, government databases holding the medical information of its citizens, or a computer data-

file which contains the diary of teenager, the need to protect the information from being viewed by anyone or anything unauthorized is plain.

Integrity refers to the validity of data or resources. This can be further refined into data integrity and origin integrity. Data integrity refers to our confidence that the information has not been modified in any unauthorized way, while origin integrity refers to the ability to verify that this information comes from where we think it comes. The last component is that of availability.

Availability refers to the ability to use the information or resource desired. Coupled into this is reliability, which encompasses availability, as it describes a level of confidence in availability.

Confidentiality, availability and integrity when categorized as security services are intended to counteract threats to the security of a system.

3.2.2 Threats

A threat as defined in RFC 2828, the Internet Security Glossary[35], is the potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm. This goes hand in hand with the definition of a vulnerability, which is an exploitable weakness in the computer system, be it based in hardware, software or people. It is relevant to note that the definition is careful not to assign intent, in that the behavior motivating the exploitation of the vulnerability is not by mandate malicious. This behavior could also be non-malicious, or even accidental. The type of threat can further divide into four categories: disclosure, deception, disruption and usurpation.

Disclosure as a threat category is the unauthorized access of information. It can take the form of either exposure, in which sensitive information is directly released to an unauthorized party, or interception, in which the unauthorized party actively intercepts information which is moving between destinations. Exposure itself can be the result of a deliberate release of information, such as a company employee publishing to the World Wide Web a company's internal memo which contains damaging information, or an unauthorized party 'scavenging' through a system to gain knowledge of sensitive information. It can also result from error- either human-based such as accidental release of information, or a hardware or software error that accomplishes the same result.

Interception, the other subset of disclosure, involves a deliberate act on the part of an unauthorized party to capture the information in route between two points. This act can be a physical hijacking of data such as intercepting a courier package, or can take the form of electronic surveillance where a data-line is monitored, and the information either directly taken from the line, or possibly by pattern analysis of the electronic signal, the information is inferred.

Deception refers to the acceptance of false data by an unsuspecting entity. In this an unauthorized party can take a number of roles. The party can 'masquerade' as an authorized entity in order to gain access to information or resources. This technique could also be used to supply misinformation which is termed 'falsification'. A third possibility is 'repudiation', where the unauthorized party denies responsibility for any acts they have committed.

Disruption, the third category, is the interruption or prevention of correct operation of some part of the system. This has three escalating levels of definition. The first is that of incapacitation, in which complete disruption of the system is achieved. The second is corruption, which degrades or adversely affects the system. The third is obstruction, which may intermittedly deny usage of the system.

The last threat category is that of usurpation, in which an unauthorized entity gains control of all or part of a system. This can involve either misappropriation, in which the entity uses a resource to which they are not entitled, or misuse, in which the entity uses the resource to cause some mischief in the system.

3.2.3 Policy and Mechanism

In order to support the security goals of confidentiality, integrity and availability, and preclude their compromise by the threats listed, there is a need to have both policies and mechanisms in place.

A security policy is a declarative statement of exactly what actions and/or situations are permissible, and what are not. Put another way, a policy elaborates by specifying what actions are considered to be secure and which are insecure, and unambiguously partitions these actions correctly. How this goal is accomplished is kept distinct by defining the security mechanism: a method, tool or procedure for enforcing a security

policy[7]. Mechanisms can be preventive in nature, or in cases where a threat cannot be prevented, a mechanism can be provided for detection, and may also specify recovery.

Inherent to the specification of security policy and mechanism are the assumptions made regarding what security is needed and the context in which it is applied. Taken together, these create assurance that the system is secure or create a level of trust in the security of the system. To capture and validate assurance and trust, further refinements are needed below the level of policy and mechanism. These are the specification, design and implementation.

The specification defines the desired functioning of a system. Its presentation can range from a formal mathematical description down to an informal written work. A system is said to satisfy a specification if the specification correctly states how the system will function[7]. The design of a system is the translation of the specification into a blueprint for a system that will meet the needs as detailed in the specification. Lastly, the implementation is the actual creation of the system itself.

When dealing with security, additional considerations come into play regarding the feasibility of affirming a security policy through a prescribed mechanism. As layering security considerations and concerns into the development process has a price in terms of time and labour, both a cost-benefit analysis and a risk analysis are required. There are a wide variety of questions to be asked at this point, for example: "What information or parts of a system require protection? What is the cost of implementing a prescribed security mechanism? If a cost is too high what are the potential risks to the system if it is implemented without the mechanism and what is the exposure or cost if the system is the compromised? What is the liability of the producer of the system if a compromise if achieved? What are the government regulations that affect the system or its requirements in the context of security?" These considerations and more weigh in on the application of security in the modern world.

3.3 Software Development Lifecycle

There are a wide variety of different methods and techniques for the creation of software. A *software process* refers to the manner in which work activities related to software development are scheduled, organized, coordinated, and performed at a certain

period of time in a given place, in order to produce a software product or service[23]. An example listing of these work activities would consist of but not be limited to the following[14]:

- System conceptualization
- System requirements and benefits analysis
- Project adoption and project scoping
- System design
- Specification of software requirements
- Architectural design
- Detailed design
- Unit development
- Software integration & testing
- System integration & testing
- Installation at site
- Site testing and acceptance
- Training and documentation
- Implementation
- Maintenance

In software engineering, the term software lifecycle is used to refer to the changes that occur during the life of the software. The software development lifecycle refers to the changes that are encountered during the development process, and these can be further broken down into phases such as requirements, planning, implementation, etc. all of which encompass the work activities listed above. A *process model* is used as an abstraction to coordinate these activities. Many process models are based on generic models of a software lifecycle process and then customized internally by the developers using them to meet particular goals or needs. We can categorize the software process models into three distinct but not necessarily disjoint classifications.

3.3.1 Ad-Hoc Development

Ad-hoc development refers to what can essentially be considered an *anti-process* model. In this type of development there is no overall formalized process model for coordination of activities, or workflow management, among developers. Instead, it is left up to the individual developers to perform activities based on their own experience and best judgment. This is described in the level 1 of the Capability Maturity Model[39], developed by the Software Engineering institute (SEI) of Carnegie Mellon University.

Higher levels of the CMM overlay a process model on the development. It should be noted that, though this type of development does not preclude successful completion of a software project, such success will be almost entirely determined by the resources, skills and talents of the individuals involved, with no reliance on a overlying process model.

3.3.2 Waterfall Process Model

The waterfall model derives its name due to the cascading effect from one phase to the other as is illustrated in the diagram below. Each phase has a well defined starting and ending point, with identifiable deliveries to the next phase as depicted below (this model is sometimes referred to as the linear sequential model).

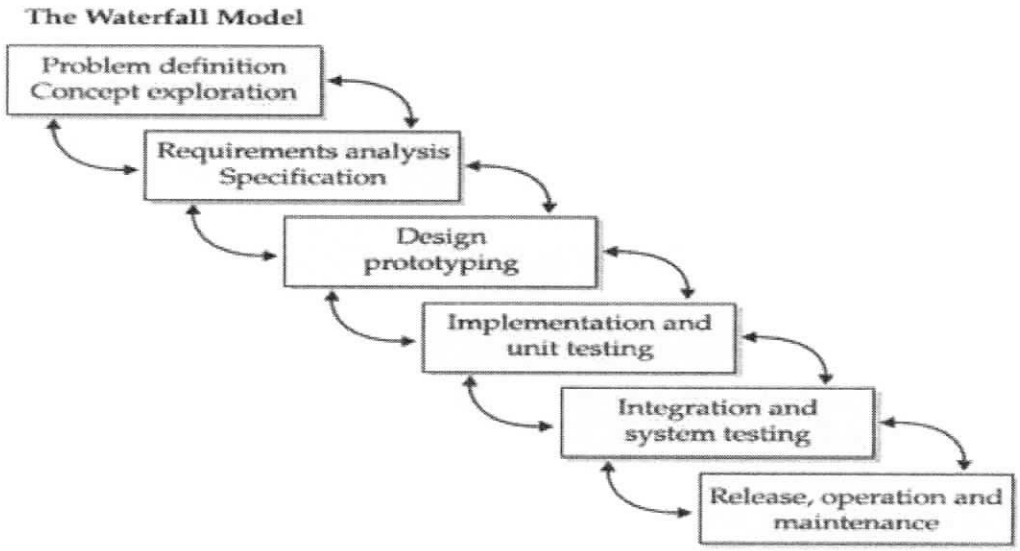


Figure 3. 1 - Waterfall process model[1]

The model consists of six distinct stages:

- Requirements analysis phase - the problem is specified along with the desired service objectives/goals and the constraints are identified.
- Specification phase - the system specification is produced from the detailed definitions defined in the first phase (this document should clearly define the product function).
- System and software design phase - the system specifications are translated into a software representation. The software engineer at this stage is concerned

[1] Picture source:
http://www.link.co.yu/StudentskePrezentacije/Aleksandar1003/radovi_1_files/image004.jpg

with the data structure, Software architecture, algorithm details and interface representations. The hardware requirements are also determined at this stage along with a picture of the overall system architecture. By the end of this stage the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces. Any faults in the specification should ideally not be passed 'down stream'.

- Implementation and testing phase - at this stage the designs are translated into the software domain, detailed documentation from the design phase can significantly reduce the coding effort and testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.
- Integration and system testing phase - all the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer for acceptance testing.
- Maintenance phase - usually the longest stage of the software. In this phase the software is updated to meet the changing customer needs, adapted to accommodate changes in the external environment, correct errors and oversights previously undetected in the testing phases and enhance the efficiency of the software.

As depicted, there are feed back loops to allow for corrections to be incorporated into the model. For example: a problem or update in the design phase requires a 'revisit' to the specifications phase. When changes are made at any phase, the relevant documentation is updated to reflect that change. This depiction of the waterfall model is a later variant of earlier applications of the model in which feedback loops were not allowed.

With this model there are some advantages: testing is inherent to every phase, it is an enforced disciplined approach, it is documentation driven, that is to say, documentation is produced at every stage, and the waterfall model is the oldest and the most widely used paradigm, so it may be considered the most mature of the software development processes.

However, many projects rarely follow its sequential flow. This is due to the inherent problems associated with its rigid format. Namely that it only incorporates iteration indirectly, thus changes may cause considerable confusion as the project progresses. In addition, at the start of a project there is usually only a vague idea of exactly what is required, and much of the clarification occurs later in the process. This model has difficulty accommodating the natural uncertainty that exists at the beginning of the project and, if the stages are strictly enforced in sequence, there is little opportunity to correct earlier misconceptions. Finally, the customer only sees a working version of the product after it has been coded. This may result in disaster if any undetected problems are presented at this stage.

3.3.2.1 Variations on the Waterfall Model

The inherent limitations of the Waterfall model have lead to some variations intended to overcome these limitations, such as the Prototyping model[39] and the Rapid Application Development Model (RAD)[39].

The Prototype model produces a working version of the product much earlier in the product lifecycle. This is because, in many instances, the client only has a general view of what is expected from the software product. In such a scenario, where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirements, the prototyping model is intended to help discover exactly what it is the customer needs, by allowing the client to interact and experiment with a working representation of the product. This increases the flexibility of the development process. The developmental process only continues once the client is satisfied with the functioning of the prototype. At that stage the developer determines the specifications of the client's real needs, and then moves into the developing the final system for the client.

The RAD model is essentially a high speed adaptation of the waterfall model, where the software development cycle shortened to two to three months. In order for this system to work however it is assumed that there are multiple teams of software development groups available to work on the same project, and furthermore that the software product required can be broken down into modules that each team can work on independently, to allow for the group to meet the shortened deadline.

3.3.3 Evolutionary Process Model

There are two main development strategies based on the evolutionary process model, that of the Incremental approach and that of the Spiral model.

The incremental approach takes the waterfall model, combines it with the prototyping model and then injects iterations into the process. Unlike the prototyping model, the software code produced at the end of an iteration of the incremental approach is intended to be fully functional, and part of the final solution. At the end of each increment the development process is restarted, and more modules and components are added. After the final iteration a complete functional product is presented. A advantage of this is at each iteration, the client can be consulted as to what they consider to be the next logic component to add to the construction, and also they can be consulted as to the validity of the solution so far. It also provides the client with a usable product at all levels of the process from the end of the first iteration onwards. It would be important to note that a potential flaw to the incremental approach is that there is no guarantee of a final usable system, though this is not a common result. As the system changes through each iteration, with added functionality driven by feedback from the customer, it is possible to discover during the last stage that a required feature, not implemented previously, is not achievable or even constructible, thereby invalidating the software product and leaving the customer with a fully functional piece of software that cannot do what they require of it. This is not limited to the incremental approach and may occur in other process models as well.

The spiral model, illustrated below, provides the potential for rapid development of incremental versions of the software by combining the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. The software is developed in a series of incremental releases, with the early stages being either paper models or prototypes. Later iterations become increasingly more complete versions of the product; though at anytime prototyping can be utilized to explore the validity of a concept.

In the model depicted there are 6 task regions consisting of:

1. Customer communication task – to establish effective communication between developer and customer

2. Planning task – to define resources, time lines and other project related information
3. Risk analysis task – to assess both technical and management risks
4. Engineering task – to build one or more representations of the application
5. Construction and release task – to construct, test, install and provide user support (e.g., documentation and training)
6. Customer Evaluation – product review by customer

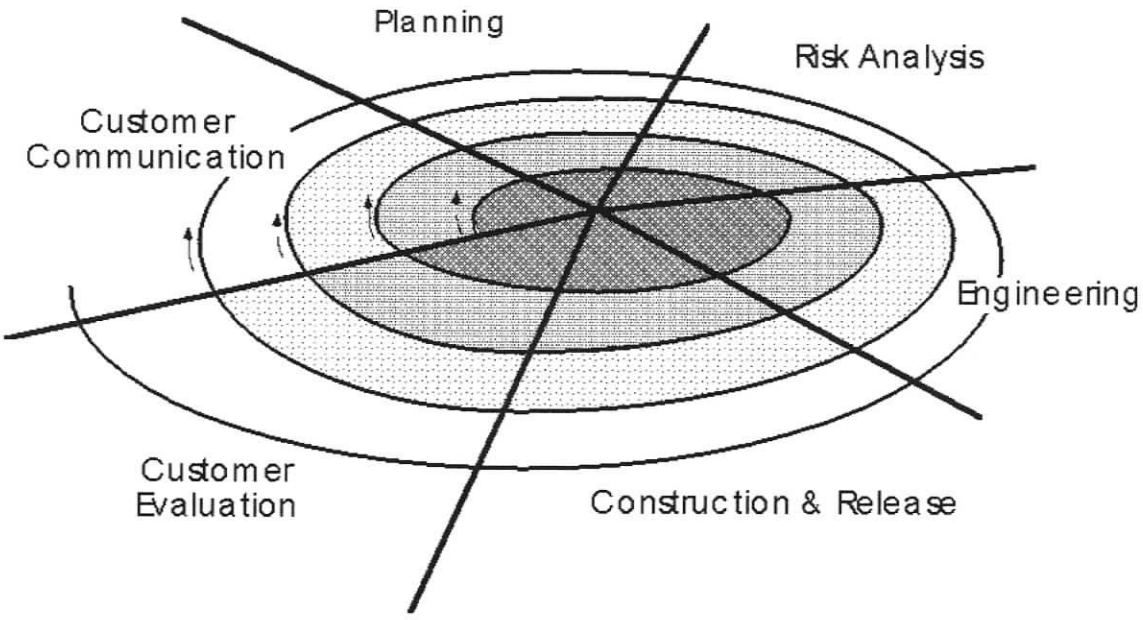


Figure 3. 2 – Spiral Process Model

The evolutionary process begins at the centre position and moves in a clockwise direction. Each traversal of the spiral typically results in a deliverable. For example, the first and second spiral traversals may result in the production of a product specification and a prototype, respectively. Subsequent traversals may then produce more sophisticated versions of the software.

An important distinction between the spiral model and other software models is the explicit consideration of risk. There are no fixed phases such as specification or design phases in the model, and it encompasses other process models. For example, prototyping

may be used in one spiral to resolve requirement uncertainties and hence reduce risks. This may then be followed by a conventional waterfall development.

The spiral model is a realistic approach to the development of large-scale software products because the software evolves as the process progresses. In addition, the developer and the client better understand and react to risks at each evolutionary level. The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development. If employed correctly, this model should reduce risks before they become problematic, as consideration of technical risks are considered at all stages.

There are some disadvantages of the Spiral Model[30]. It demands considerable risk-assessment expertise and, as it is a relatively new process model, it has not been employed as much proven models (e.g. the Waterfall model) and, hence, may prove difficult to 'sell' to the client.

3.3.1 Rational Unified Process

The Rational Unified Process (RUP) is an iterative approach that expands on the Unified Software Development Process described in [19]. It combines a subset of the development activities each time through an iteration; these activities may be repeated in subsequent iterations as well, or new activities added and others removed. It combines elements of both the spiral process model and the incremental process model. The static structure of RUP is comprised of four sequential phases: Inception, Elaboration, Construction and Transition. Each of these phases may undergo one or more iterations of a set of development activities. The Inception phase is intended to capture the scope of the project and determine among the stake-holders whether to proceed. The Elaboration phase creates the architecture for the proposed solution and deals with any areas of technical risk. The Construction phase produces the first operational version of the software product itself and the Transition phase deals with refinements to the project and successful migration of the software to the customer's domain. As the following figure[21] depicts, each pass through (a vertical slice in the diagram) touches on most of the development activities with the goal of exposing risks, both business and technical, as early in the process as possible. Furthermore, unlike the waterfall process model, each

development activity is visited multiple times to allow for revision and changes based on discoveries made during the last iteration.

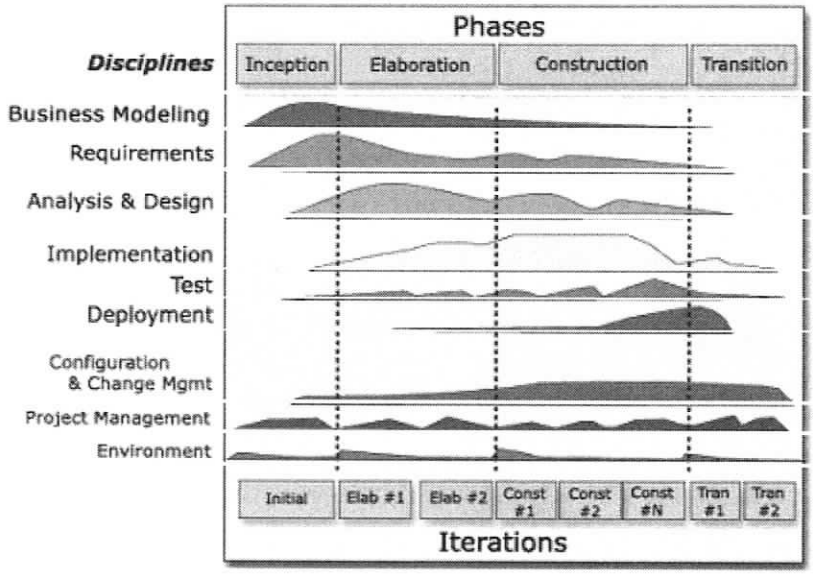


Figure 3. 3 - RUP process model[2]

3.4 Decision Support Systems

Decision support systems (DSS) encompass a wide category of different areas and can undertake many different forms. Decision-making itself is a cognitive process that is still an open area of research for psychologists. Techniques for making decisions can vary from a methodical approach of listing on paper all the advantages and disadvantages of the options available, and then comparing side by side all the possibilities until one path seems desirable above all others, to the simple flipping of coins to randomly select an option. With the advent of the computing age, the ability to create software to develop more complex tools to aid in the decision support process became possible, and subsequently such systems have been developed. In its simplest definition, a decision-support system can be considered a computer information system that aids the process of decision-making. This covers a variety of circumstance: from a spreadsheet application possessing an auto-recalculation feature which is used to create budget forecasts to aid in business decisions through to expert systems designed to aid with medical diagnosis.

[2] Picture Source: http://www-128.ibm.com/developerworks/rational/library/may05/brown/fig1brown.gif

Classifying the different categories of systems can depend on which researcher is interviewed, though they have several equivalencies. The one suggested by Power in [29] differentiates between communication-driven DSS, data-driven DSS, document-driven DSS, knowledge-driven DSS, and model-driven DSS as the five main categories.

A model-driven DSS emphasizes access to and manipulation of a statistical, financial, optimization, or simulation model. A model-driven DSS uses data and parameters provided by DSS users to aid decision makers in analyzing a situation, but they are not necessarily data intensive. The spreadsheet example mentioned prior would fall into this category. A communication-driven DSS supports more than one person working on a shared task, or put amore formally, an environment intended to support a collaborative effort. A data-driven DSS or data-oriented DSS emphasizes access to, and manipulation of, a time series of internal company data and, sometimes, external data. A document-driven DSS manages, retrieves and manipulates unstructured information in a variety of electronic formats and, finally, a knowledge-driven DSS provides specialized problem solving expertise, stored as facts, rules, procedures, or in similar structures.

In this work of constructing a decision support tool for the managing of security complexity in software application development, it would be reasonable to say that such a tool would be a combination of the last two categories mentioned. It should organize known security research consisting of documentation and fit this knowledge into an existing software development framework to assist in managing the problem of "security" with regard to software development.

It is important to note that, in the aforementioned definitions of decision support systems, there is an implication that the underlying tool will provide some type of automated mechanism which will assist in the cognition of the decision making process. In the case of this work, a Decision support system is defined to be a visually driven, knowledge-based mapping of security issues, potential responses and the consequences of enacting the responses. This supports the decision making process by creating a visual layout of the information that can be viewed, and grouping the key facts and relationships into a localized visual paradigm to facilitate cognition.

3.5 Grounded Theory

The term "Grounded theory" in its original form was proposed by Glaser and Strauss[15]. It proposes that rather than construct a theory and then search for data that either supports or invalidates the precepts of the theory, instead review all the data and then, from that information, inductively produce a theory or theories from the dataset. In this fashion there would then exist at least one set of data that fits the theory, since it was the data itself that produced it. The diagram below, from a paper by Dick, illustrates the process.

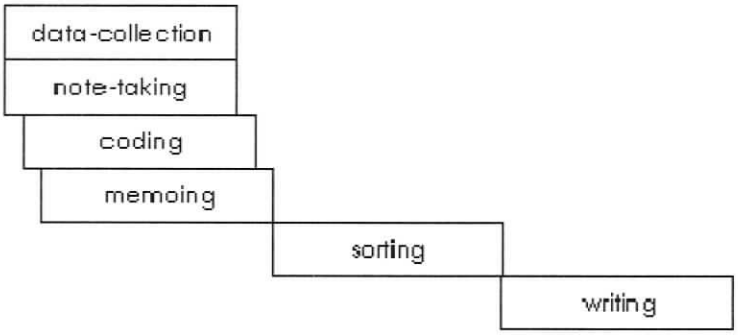


Figure 3. 4 - Phases of Grounded Theory[3]

The basic idea of the grounded theory approach is to read (and later review again) a textual database consisting of literature in the field, interviews with experts, media publications and essentially any other viable source of information that can be considered relevant and reliable and "discover" or label variables (called categories, concepts and properties) and their interrelationships. This is relevant in the data-collection and note-taking phases. The part of the analysis concerned with identifying, naming, categorizing and describing phenomena found in the text is referred to as "open coding". Essentially, each line, sentence, paragraph etc. is read in search of the answer to the repeated question "what is this about? What is being referenced here?" and then recorded as a label. These labels refer to things like hospitals, information gathering, friendship, social loss, etc. They are the nouns and verbs of a conceptual world. Also noted are the adjectives and adverbs- the properties of these categories. For example, about a friendship we might ask about its duration, and its closeness, and its importance to each party. Whether these properties or dimensions come from the data itself, from respondents, or from the mind of

[3] Picture Source: http://www.scu.edu.au/schools/gcm/ar/arp/grounded.html

the researcher depends on the goals of the research. From here the process of "memoing" begins. As coding proceeds, certain theoretical propositions will begin to appear when reviewing the information. These may be about links between categories, or about a core category: a category which appears central to the study. As the categories and properties emerge, they and their links to the core category provide the theory. To capture the link and the thought process behind it as well as any other observations a short document called a memo is created.

Data collections, coding and memoing proceed until a sufficient saturation level is reached. This is noted when continual addition of codes from data sources simply repeats information already present and doesn't add anything new. At this point we move into the sorting phase to organize and analyze the memos and information collected. This sorting will yield the structure of the report, which is then done in the writing phase.

This approach fits with the construction of ontology's. In computer science, an ontology is a data model that represents a domain and is used to reason about the objects in that domain, and the relations between them. Since computer science as a whole is a relatively young field, only arising in the last half century, much of the work and information related to it is emergent in a wide variety of sources, without an overall organization or reference document detailing the domain. In order to organize information for research the approach of grounded theory can be used to build from the ground up an ontology based on the known facts and research which will categorize the data available and allow for the development of theory. It should be noted that, with the approach of Grounded Theory, there is an assumption of stability in the field under examination. This cannot be said for the field of Security in Computer Science. As previously noted, much of the field, theory and terminology is still emerging, and may be subject to change in the near feature as more is discovered. However, the main reason for using the Grounded Theory to construct the ontology is that this approach will create an un-biased conceptualization, not skewed in any one direction, or potentially subject to the author's own pre-conceptions. The ontology that is produced cannot be considered to be stable, and it should be recognized that in the near future, it may need to be revised to reflect changes in the field. Its creation however, as a temporary 'snap-shot' of current knowledge, will provide the necessary basis to continue work in the present, and may

even help to add stability to the field of security, as it can provide an anchor for future work..

Chapter 4 - Process Framework Construction

4.1 Introduction

The first step to constructing the decision support tool is to create a meta-structure that will represent the software development process and incorporate this structure into a software tool that will facilitate the decision process. This framework will be used as a roadmap on which to place the security issues that are described in Chapter 5 during the theoretical analysis and construction phase. In the first part of this chapter, a visual framework is derived for mapping into the software development process. Then, the properties needed for a software tool that can be used for this work are presented, followed by the decision and reasons behind selecting the Compendium tool. This chapter is then completed by a section illustrating how the meta-structure is represented in Compendium, and how the features of Compendium will map into the theoretical analysis and tool construction that is done in the next chapter.

4.2 Meta-Structure

One of the foundations of this work is the injection of security issues into an appropriate point in the software development process. In order to identify this point we need to have a representation of a software development process that can be used for the described placement. As discussed in the background (section 3.3) there are numerous process models to choose from, which give rise to the question: which model should be chosen to represent the framework? As previously discussed, there is no single process model that predominates the others in industry use, and even if a particular company adheres to a particular process model, it is still likely that some customization has occurred making the model unique in its context. In order for the tool to provide for the widest possible use there is a need to create a generic model which can be applied in the majority of circumstances. Of the three main themes of software development process models: the ad-hoc, the waterfall and the evolutionary, we first begin by recognizing that in the ad-hoc development process, there is no model in which to organize the relevant security issues, since it relies on individual experience. Therefore we make the assumption that the individual involved in an ad-hoc development will be able to

recognize security issues that should be dealt with for their project. Though this assumption may not be valid in all cases, it doesn't preclude said individual from making use of the security information encoded in the tool for the purposes of their software development, and therefore the tool can still be of benefit to them. This actually spawns one applicable criterion for a later section, which deals with the tool construction: that whatever structure is imposed, the tool should allow for an essentially non-hierarchal or flat read of the encoded security information, so that it can provide a list of security concerns that may be relevant in an ad-hoc development project. It would then, of necessity, rely on the experience of the developer (as the ad-hoc process does) to peruse the list and determine what is relevant to their project.

This leaves us with the remaining two process model categories for consideration, the waterfall and the evolutionary. For consideration the figure below depicts an example of the spiral development process, a subset of the evolutionary model:

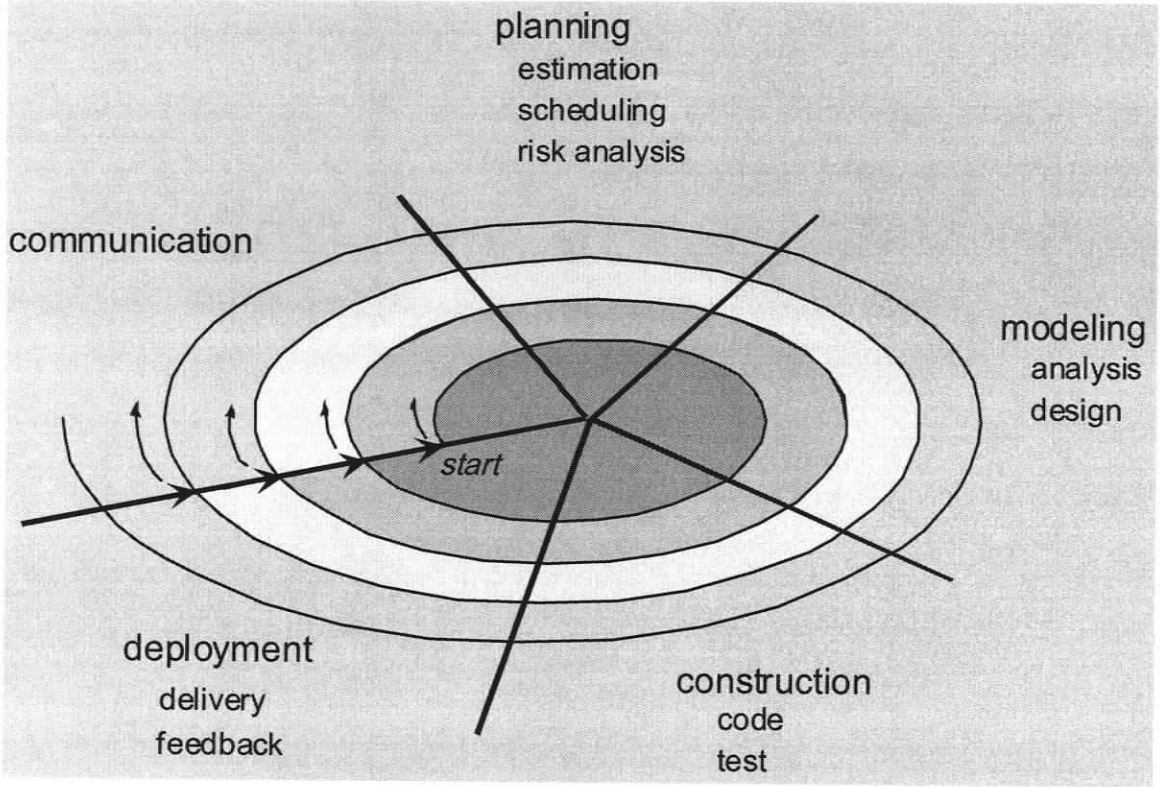


Figure 4. 1 - Spiral process model[4]

[4] Picture Source: R. Pressman, *Software Engineering, a practitioners' approach, 6th edition, 2005*

If we now take into consideration the six stages of the waterfall as described in section 3.3.1, and consider them in the context of the depiction of the spiral process, we can produce a mapping between the two. For example, the construction phase shown represents what is essentially a 'pie slice', taken from the spiral. The integration step from the waterfall model would map directly onto this slice. In other words, this spiral suggests that there will be several iterations through the construction phase. To translate this to the waterfall process, we would group together the considerations taken for each iteration through the construction phase to construct one complete 'slice' of the spiral, which then becomes the implementation stage in a software project that is adhering to the waterfall process. Similar mappings can be accomplished for the other five stages of the waterfall, by defining an appropriate slice of the spiral. The end result is that by using the spiral, it will not preclude application of the tool to a software project utilizing a waterfall model or one of its variations. In point of fact if we keep to providing the depicted spiral as visualization for mapping to the waterfall, by way of defining a slice that represents the waterfall stage, the mapping will be straightforward. This produces a second criterion for the tool: it should provide a way to represent the visual paradigm of the software process model, to facilitate mapping between it and other models. The diagram below illustrates how this mapping would work.

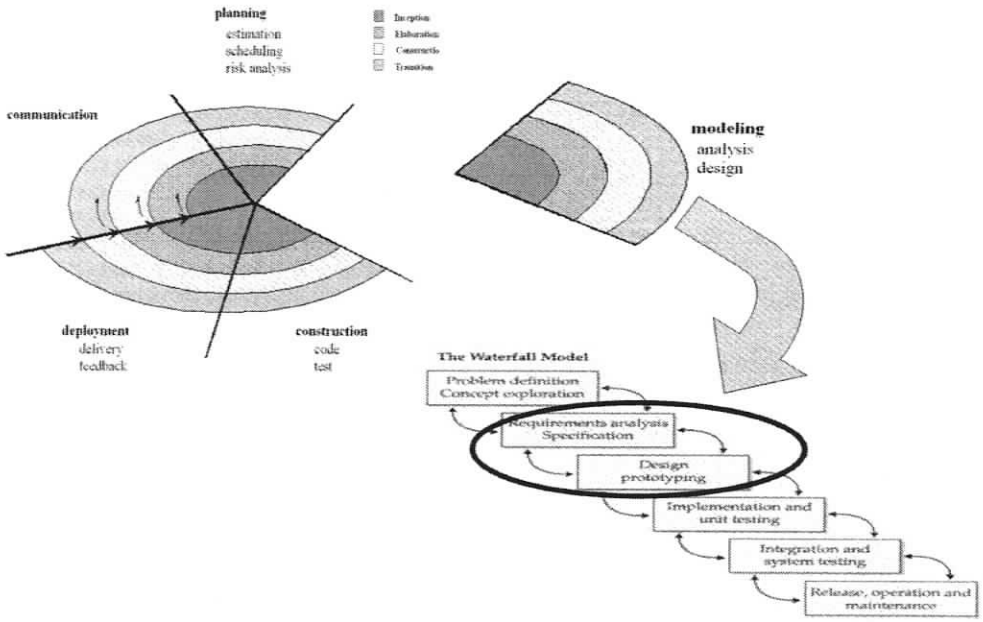


Figure 4.2 Mapping to the Waterfall model

Though we now have a mapping between ad-hoc, waterfall and the spiral there is still yet one more generic model to account for. The other subset of the evolutionary is the incremental process model. However, in order to connect the incremental to the others there already exists a process model that makes the mapping, the Rational Unified Process described in section 3.3.1. In order to make the connection then we need to incorporate the RUP into the spiral model, which is our current working background. The key to this is recognizing that each of the four static structural phases of the RUP, inception, elaboration, construction and transition, can be represented by one track around the spiral model. In cases where there is more than a single iteration through a particular RUP phase, this can be captured by simply making another pass in the same track. The visual representation of this is made in the figure depicted below. Each RUP phase is mapped to its appropriate track in the spiral, starting with the inception phase which is the innermost or starting track. The next track out from the spiral which is identified both by being a different color as well as adjoining the end of the spiral track named inception is the elaboration phase, and as shown this is continued on for both the construction and then transition phase. The end of the transition phase demarks the 'end' of the process. This does not, however, complete the mapping as the four static RUP phases only represented the one dimension of the RUP model. The other dimension that must be mapped consists of the working activities.

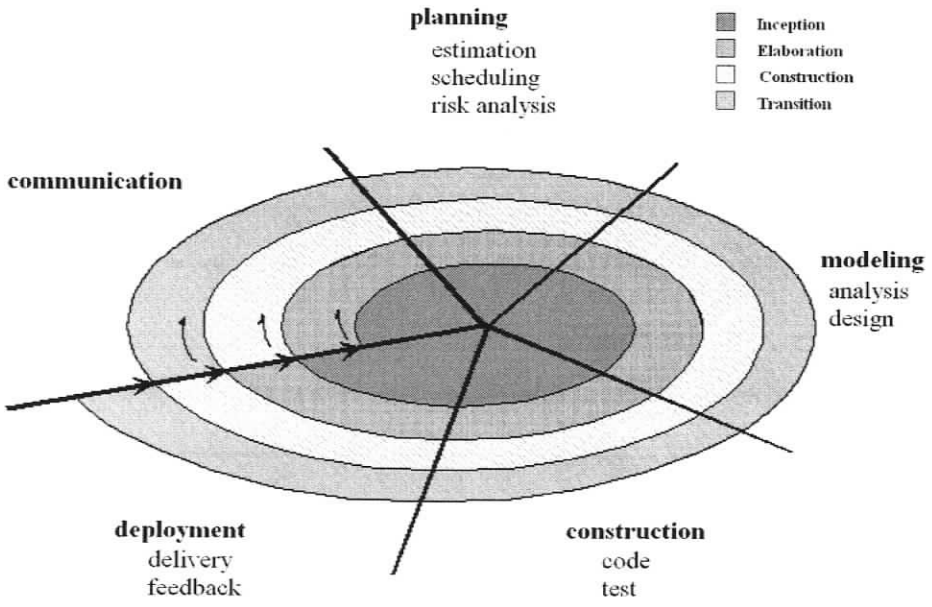


Figure 4. 3 - The Security Spiral

To create the mapping of the workplace activities we start with the diagram of the RUP process as shown and create a node or point that demarks the intersection of that workplace activity (the horizontal) with the static structure phase (the vertical):

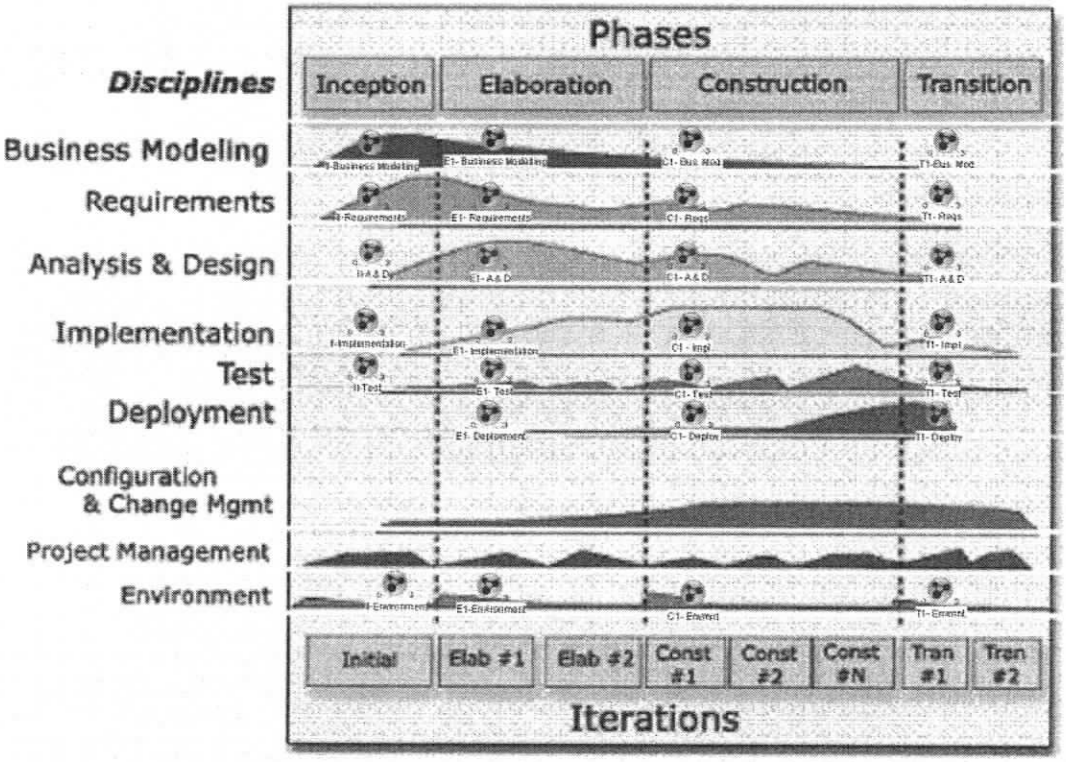


Figure 4. 4 - RUP process model with nodes[5]

The next step is to then transfer them to the combination Spiral/RUP. In the Spiral/RUP we start at the innermost spiral, and work clockwise towards the outside of the spiral. This is how time is represented in the development process. To move the process elements into the spiral development model, as was previously discussed, we copy the process element icons into one on the tracks of the model (recall each track was defined in terms of a phase of the RUP).

Before this is done, it is important to note that process element icons were only placed in a particular phase if the visual representation indicated that work on that area was being done in that phase. In the figure of the prior page, in the inception phase, there is no entry for the deployment discipline visually, and hence we do not place an icon there.

[5] Picture Source: http://www-128.ibm.com/developerworks/rational/library/may05/brown/fig1brown.gif

Also of note in this process is the absence of icons for both the Project Management discipline and the Configuration and Change management discipline. Since both of these refer to organization techniques for the software process, their relevance in terms of decision support for security is difficult to determine. For the purposes of this work there are considered to be subsumed into the Requirements process, and unless an area is exposed in which they are specifically mentioned, they will not be included. This brings us to the figure on the following page, which shows the resulting visual model.

As depicted, each phase icon has mapped into a particular track of the spiral. Furthermore each track is color coded to represent an iteration of the Rational Unified process, beginning with the innermost starting track representing the Inception phase. This is then followed by the Elaboration, Construction and Transition phases in each consecutive spiral outward.

In almost all of the cases, the RUP node cleanly transferred to a relevant spot in one of the five quadrants from the Spiral model, , i.e. E- A & D standing for Analysis and Design in the Elaboration phase, clearly falls into the Modeling quadrant of the Spiral, in the track appropriate to the Elaboration phase. The only node element which lacked a clean target quadrant in the spiral was the Environment nodes. As information pertaining to this is usually solicited from the client, and based in large part on their needs, it was placed alongside the Business Modeling nodes in the Communication quadrant.

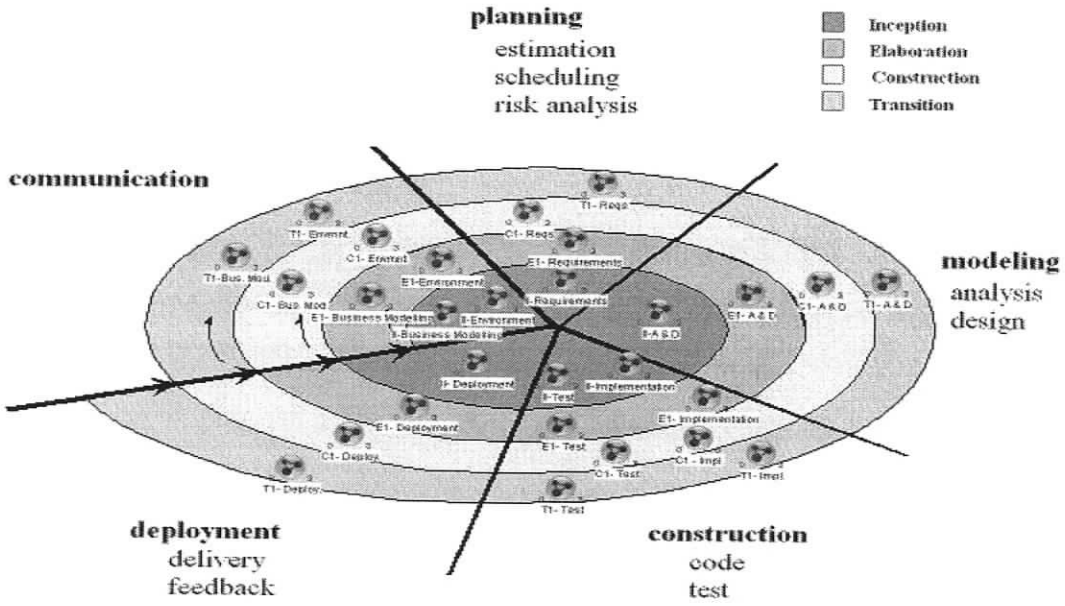


Figure 4. 5 - The Security Spiral with nodes

To re-affirm the context and summarize the construction so far, the spiral model and the Rational Unified Process have been merged to provide what are essentially anchor points in the software development process, which can also be used by mapping to projects which make use of the waterfall process. The next step is further refinement of these nodes.

4.2.1 Node Refinement

In order to be able to better place the security issues from the theoretical construction performed in the next chapter, each particular node from the RUP workplace activity model that was transferred to what we now define as the Security Spiral requires a brief analysis. This is to ascertain the relevant context of the security concern, with regard to software process, and facilitate deciding which of the 28 nodes in the Security Spiral it would be the most relevant to place a particular security issue in. In order to assist in this analysis, the following table briefly describes the context of the process node (discipline) in the overall development process. As mentioned, this will help anchor each issue of security to its relevant placement in the software development lifecycle. This information can then be incorporated into the tool for reference not only during the placement of the security note, but also later when the tool is used to support security decisions during software development. The majority of the information listed in the following table was derived from [21]. It is recognized that certain issues of placement, such as "when are non-functional requirements first considered?" can be located in more than one node. For example, non-functional requirements can be considered both in the Business Model node of the inception phase, or in the Environment node of the same phase, or in both. In these cases the issue under consideration is listed in both sections, with the understanding that this may cause a replication of information, if a security issue is identified as belonging to that overlapping criterion, and then it must be placed into both. The following listing is not comprehensive, as its intent is to provide a context to the node, to help with placement of security issues, and is it not intended to be a complete listing of all software development issues in that part of the process. In other words, its

intent is to anchor that node in the process model, for the purpose of placing security issues.

Inception Phase
Business Model • Understand the scope of the project required by the customer   o Produce a Vision Document detailing the product, expected functionality, users for whom the project is targeted and any known non-functional requirements

Requirements
• Detail expectations regarding project from the customers perspective and verify these expectations with the customer → Use case construction
  o Identify key system functionality
  o Identify target users and their interaction with the system (the Use case)
• Determine the cost, schedule and risks associated with the project

Analysis and Design • Establish scope of system • Construct a proposed architecture of the system, or at least a possible solution • Identify areas of risk from both the business perspective (mandatory functionality of the system) and the developers (areas of the software development that are either 1- know to be difficult to construct or implement or 2 – completely unknown and therefore considered high risk until the risk exposure can be determined by investigation)
Implementation • Identify high risk areas for either business concerns or developers and prioritize these areas first • Possibly implement a prototype to verify 'do-ability' of a poorly-understood or difficult concept → results from the Analysis and Design phase • Possibly implement a prototype/storyboard of User interfaces or user interaction with the system to clarify understanding by walking customers through the use case using the storyboards to demonstrate operation

Test

- Confer with Client to determine external testing requirements
- Identify key components for internal test harness development

Environment

- Ascertain all relevant points regarding the operating environment in which the project will operate, including hardware, software and network considerations.
    - List these as non-functional requirements
- Determine if the environment will evolve or change in the future; note the possible impact on the project and client expectations with regard to this, i.e. if expected to handle changing environment is this included in the requirements and accepted as an extra cost to the project

Elaboration Phase

Business Model

- Revise model (Vision Document) to represent all information discovered in the inception phase

Requirements

- Revise requirements in depth
- Complete the majority of the use cases (target is 80%)
    - Identify which use cases need to be addressed during the Elaboration phase and which can be postponed to the construction phase
    - Complete a customer walk-through with the user-interface prototype/storyboard to verify the use case
- Identify non-functional requirements(performance, stability) that will need to be dealt with in the construction phase

Analysis and Design

- Design the Baseline architecture
    - Identify all building blocks of the system including any commercial off the shelf (COTS) components
    - Identify architecturally significant Use cases (key system functionality) that should be part of the implementation in this phase

Implementation • Build prototype of the Baseline architecture
Test • Test prototype of the Baseline architecture o Validate that architecture will work o Verify that all major technical risks have been resolved o Verify that proper quality attributes are present: performance, scalability and cost o Test critical scenarios
Deployment • Perform integration and system testing for early deployment and feedback loops o Expose product to small user group to verify application is useful and provides desired behavior; identify areas of insufficient capabilities where revision required for next phase o Look for scenarios where software performs to requirements by requirements do not make sense in the user environment; earmark for revision in construction phase
Environment • Verify that the baseline architecture will be sufficient in the required operating environment.
Construction Phase
Business Model • Resolve issues that require clarification from the client that were identified in prior phases → Revise Vision Document and solicit stakeholder agreement from the customer
Requirements • Define the number of iterations required for the construction phase o Develop an integration build plan for each iteration o Work from high-risk to low risk through the iterations o Identify previously missed supporting components and add to the next

iteration; revise requirements and design accordingly • Refine requirements to necessary detail to complete all use cases
Analysis and Design • Finalize architecture and actively enforce the resulting design
Implementation • Perform incremental builds as per the iterations defined in the requirements • Focus each iteration on completing a set of components, subsystem or set of use-cases
Test • Compile and test all code; Fix issues as they become known • Apply unit tests and derive tests from use-case scenarios
Deployment • Determine whether software, the sites, and the users are all ready for application to be deployed → create a deployment plan to resolve any aforementioned issues • Perform a pre-release deployment, i.e. Beta deployment o Solicit feedback from users o Identify issues that need resolution(i.e. bugs, missed requirements) • Identify existing operational databases or programs that will need to be migrated to interact or be loaded on the new system
Environment • Confirm project meets previously described operating conditions
Transition Phase
Business Model • Confirm that clients expectations have been met • Finalize evaluation criteria from Vision Document • Resolve any outstanding issues with regard to the software project by either o Adding to current implementation required changes o Documenting why this issue will remain outstanding
Requirements • At this stage requirements should be subject only to minor clarifications or

corrections. No new requirements should be added at this point.
Analysis and Design • Each change request for bug fixes needs to be analyzed for impact on other components and a plan for implementation of the change request created. • New features may be added at this point but for this feature a Construction phase Iteration should be performed focusing on the feature to be added in order to identify impact on the overall project
Implementation • Implement any change requests according the plan produced in the Analysis and Design phase
Test • Beta-test to validate that users expectations are met o Capture, analyze and implement change requests (for defects) • Identify and perform regression testing • Perform acceptance testing
Deployment • Train users to use the system • Migrate existing systems or databases to the new system
Environment • Verification of the environment only.

Tabel 4. 1 - Workplace activity anchor descriptions

4.3 Tool Software Requirements

In the prior sections two criteria have been identified as important to the environment needed for the construction of the decision support tool. The first is the ability to extract the issues in a non-hierarchal listing for those whom are using an ad-hoc development process. This is considered a necessary component. The second is to provide a mechanism for visualizing the Security Spiral as it was developed and depicted, as this

visualization facilitates the mapping between the different software development process models. This is also considered a requisite. To these are added the following[6]:

1. The software tool employed should be free to download and use. This allows the research and construction of this tool to be continued by others, and also would allow for its use in the widest variety of circumstances if no license fee is required.
2. If possible, the tool should be cross platform. This would allow for the widest possible use without restriction.
3. The tool must allow for the export and import of the Decision support model once constructed, either as an inherent part of the tool or as a separate construct.
4. As new issues will arise in both security and software development, the tool must allow for future modification.
5. If possible, the tool should be open-source to allow for additional features to be added assuming that there will be areas of opportunity to extend the functionality of the tool to aid in its use in the security context.
6. If possible, the tool should not just have the required feature of the flat non-hierarchal printout of security issues as described previously, but also be able to provide this in a format that will facilitate the transfer of security information to another tool.
7. The tool must provide for a mechanism to annotate and extract relevant points once discovered.
8. If possible, the tool should a mechanism to support a shared environment so that more than one developer can use the tool concurrently for the same software project.

It is accurate to state that the most direct method to meet all the above criteria is to create a software tool from scratch, and define the aforementioned list as the software projects goal from initial inception. However, work has already been done on decision support and issue based information management software that may fulfill the needs as described, and to avoid re-inventing the wheel, a survey was done to first ascertain that

[6] If the requisite is considered a plus but not a necessity, its notation is prefaced with "If possible"

no product available could fulfill the criteria as listed, before beginning to develop our own tool.

Searching the World Wide Web for a free tool to implement a decision support system produced two software projects of note that warranted further investigation. The names of these two products were Compendium[7] and Dicodess[8].

Dicodess is an open source software project created by the Decision Support Systems group of the University of Fribourg, Switzerland. Though written in Java, the support page indicates that it is a product specific to the Windows platform due to its reliance on a Windows DLL (dynamic link library). As shown in the figure below, it provides a graphical interface for entering information into the decision support system as well as supporting the decision process, though the graphical interface is restricted to a tree structured outline view:

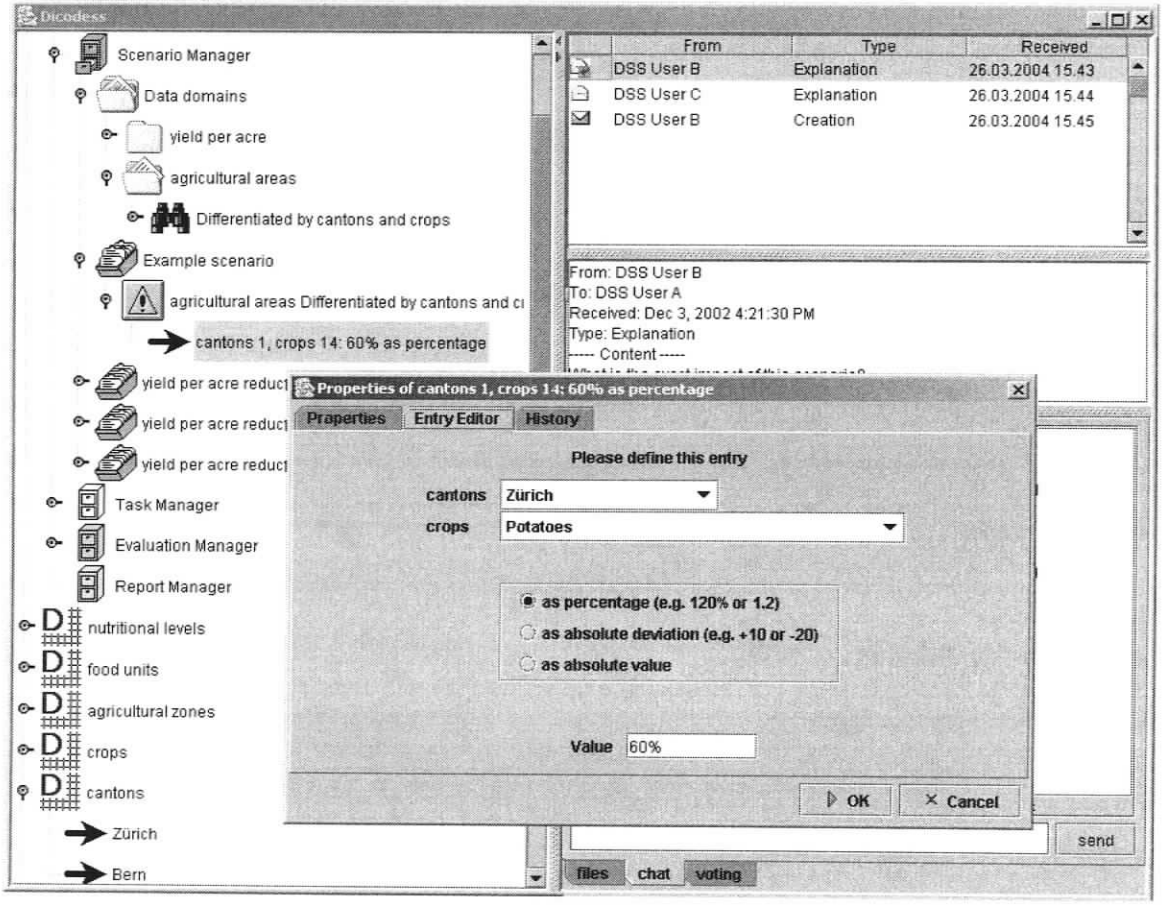


Figure 4. 6 - Dicodess Software application[9]

[7] http://www.compendiuminstitute.org/default.htm
[8] http://dicodess.sourceforge.net/

Compendium is a creation of the Compendium institute which is a collaboration of industry representatives (including NASA) and academic institutions (primarily the Knowledge management institute of the Open University). Also written in Java it does not have a reliance on a particular windows component and therefore meets the criteria for a cross-platform product. Unlike Dicodess, the Compendium product was not developed specifically for decision support; it is primarily developed in the realm of knowledge management, and it brings with it a rich visual environment, as shown in the figure below.

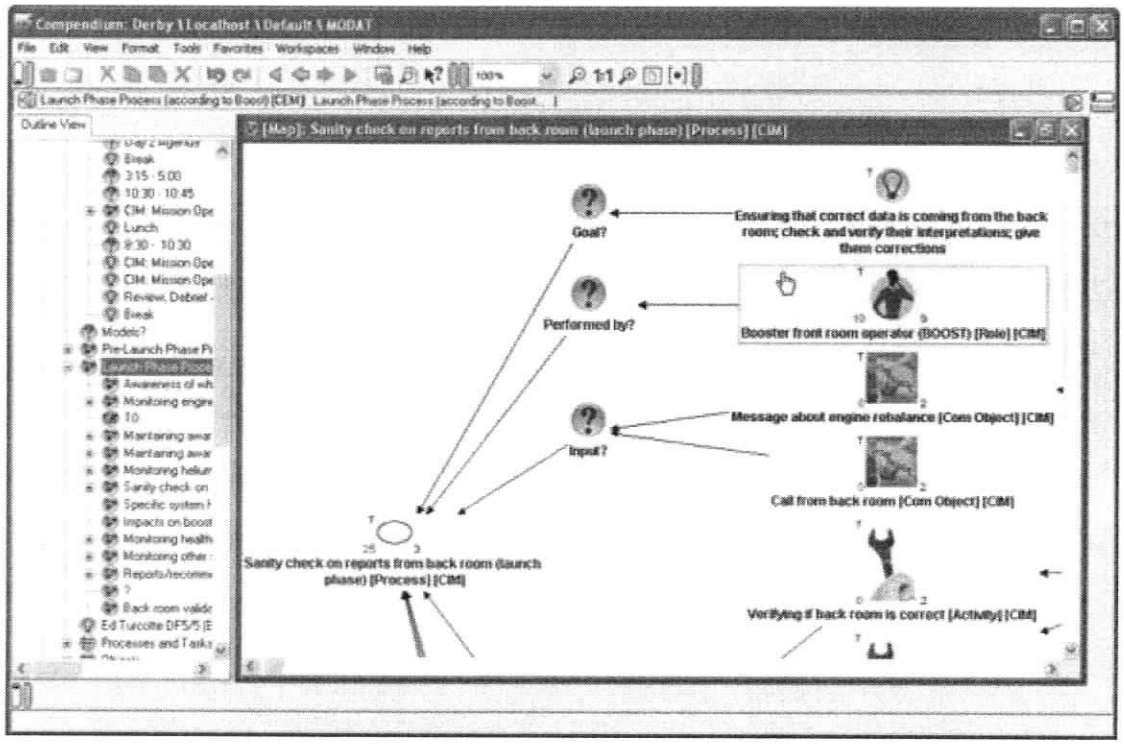


Figure 4.7 - The Compendium tool[10]

The table on the following page summarizes the criteria points that were previously laid out and how both Compendium and Dicodess compare.

[9] Picture Source: http://dicodess.sourceforge.net/images/screenshots/entry_definition.png
[10] Picture Source: http://www.compendiuminstitute.org/community/showcase.htm

Supports	Dicodess	Compendium
Hierarchal extraction (ad-hoc)	Yes	Yes
Visual Security Spiral	No	Yes
Free download/use	Yes	Yes
Cross platform	No-but future intended	Yes
Export/Import Model	Yes	Yes
Open source	Yes	Yes
Future Model modification	Yes	Yes
Easy transfer of Security Information	Yes	Yes
Annotations Support	Yes	somewhat
Collaborative Environment	Yes	Yes

Tabel 4. 2 - Compendium vs. Dicodess

As the table implies, both products meet not only the required criteria but most of the preferred but not required list as well. As both tools are available the option of constructing the decision support tool from scratch is now beyond consideration, as it would be replicating work that is already done. Further to this point the intent of this thesis is the construction of a mechanism to support security decisions in software development, not to construct software to implement a decision support tool as such. Given that, the decision now becomes which tool is preferable.

The type of decision making process needed is almost entirely ensconced in document driven decision support, with a small overlap into the area of knowledge and communication driven decision support. The Dicodess product is intended primarily for model driven decision support where data can, in most cases, be given mathematical weight which is used to help make decisions. Compendium is built to provide far greater variety in terms of visual mapping and navigation of issues, and though not specifically tailored for decision support, will function in this regard. This leads to the conclusion that Compendium is the preferable environment to attempt to implement the tool. It is relevant to this argument to note that the security ontology described in Chapter five was actually produced prior to this decision in the timeline of this thesis. Based on the

experience gained cataloging the different papers on security and construction of the ontology, it was clear that the resulting decision support mechanism would be document driven.

The next step is to determine how the Security Spiral can be implemented using Compendium, to support the decision process.

4.4 Compendium and the Security Spiral

The main strength of using the Compendium tool comes about as a result of its ability to provide visual mapping and navigation through information that has been entered. Using the picture of the Security Spiral as displayed in figure 4.2, a background picture displaying the same information is imported into Compendium. The next step is to create a representation of the meta-structure by using the representations available. It is important to note that one of the major strengths of using the Compendium software is that the tool provides for navigation through the workspaces in essentially the same manner as one would navigate documents on the World Wide Web.

4.4.1 Mapping the Security Spiral

The start point of the process involves creating what is essentially the 'root' node in a tree of visual navigation. This starting point will represent a navigable link to the visual representation of the Security Spiral. In order to maintain a link to the RUP visual representation, a second node is added that will link into the chart displaying the RUP process that was previously discussed in Section 4.2. The resulting display is shown in the figure below:

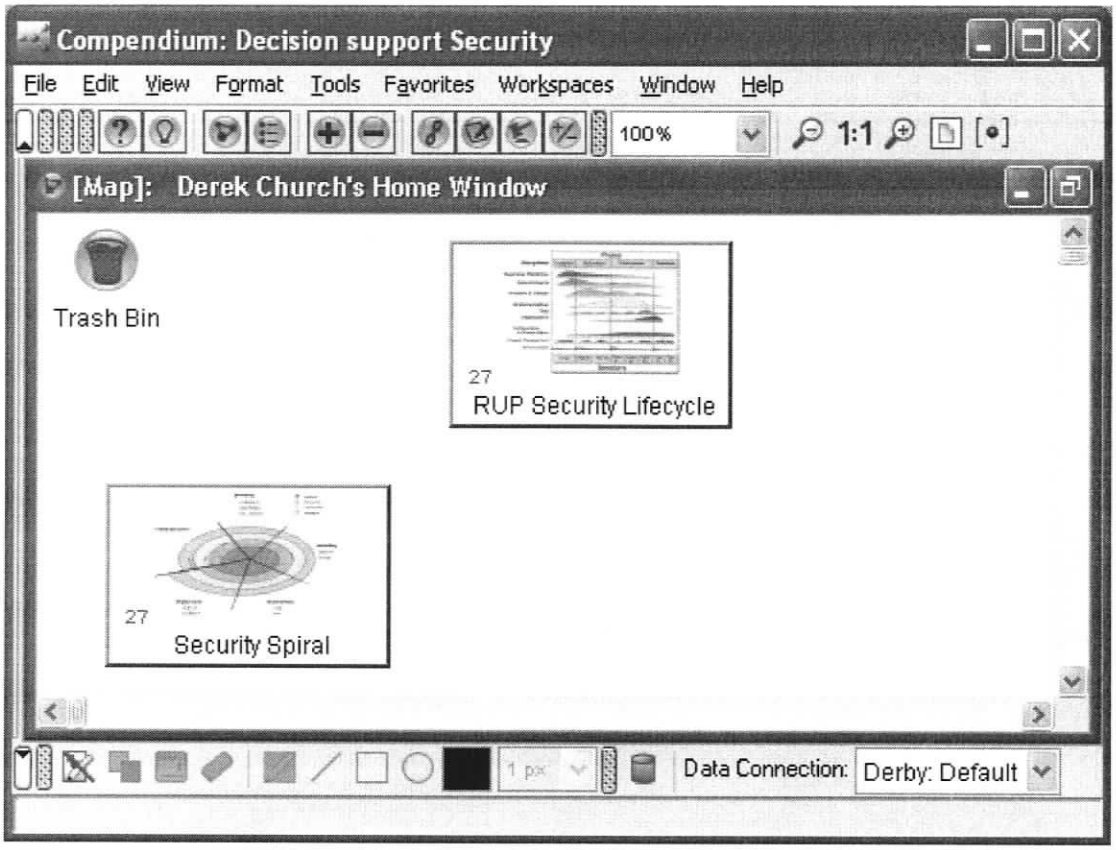


Figure 4. 8 - Main page of Decision support tool

Both the Security Spiral icon and the RUP Security Lifecycle icon are now linked through a mouse click to another workspace. This type of node in the Compendium tool is referred to as a *Map View*. In simplest terms a *Map View* represents the creation of another workspace inside the current one, which can be accessed through the use of traditional means of navigation (i.e. doubling clicking on the icon with the mouse). The default image for pictorially representing a map node consists of a purple circle, in which there are three smaller circles connected in a triangle. The tool, however, allows the icon to be changed to another image, which was done in the above example to more clearly indicate the purpose of the link.

The next step is to click into one of these workspaces and begin the construction of a navigable Security Spiral construct. This is done by importing the image of the Security Spiral, constructed previously, as a background and then creating *Map View* nodes representing the different phases of the RUP lifecycle and their location as previously defined. The resulting view is depicted below:

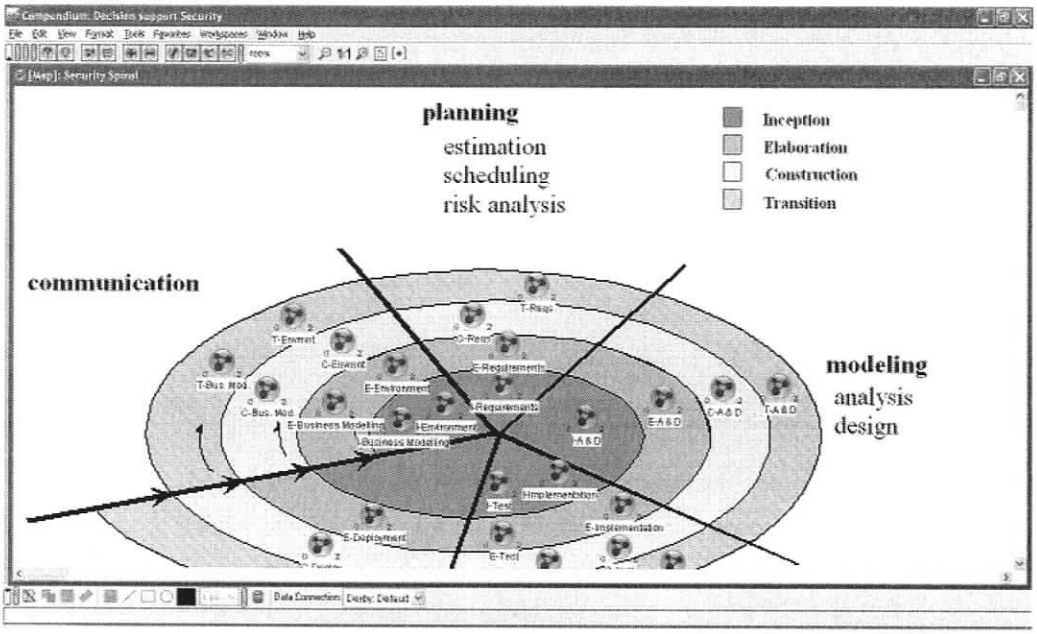


Figure 4. 9 - Embedding the Security Spiral

Each one of the nodes is now linked to another workspace where information can be placed. Compendium allows for a node to be present in multiple maps, all linked to the same workspace. This allows us to duplicate a similar representation of the RUP lifecycle to providing an alternate navigable view, as shown below:

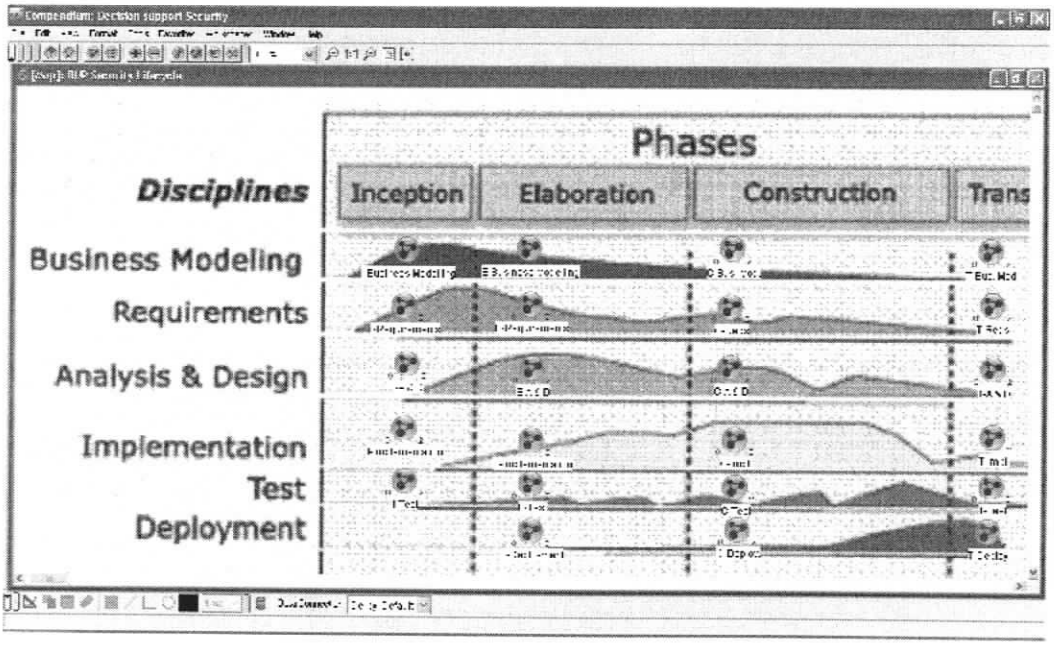


Figure 4. 10 - Embedding the RUP process model

Given that there is now a map that will lead into the different phases of the workplace activities of the software development lifecycle, the next step is to examine how the tool can provide for the representation of Security information.

4.4.2 Issue management and Navigation

In order to provide visual clues for the navigation of the different issues, there are a variety of node types that can be employed. These nodes will be clickable in the same manner as hyperlinks on the World Wide Web. The result of clicking on one of the links will be either the workspace representing the node if it is a map will open, or if it is any other node type then the properties window of the node will be displayed in the application. By doing this we leverage a developers existing knowledge of navigating the World Wide Web and the fact that they can apply this same knowledge to navigating the framework of the tool. Using icons that provide visual clues, as well as the background images that help map the current location of the user in the tool to whichever phase of the software lifecycle model that they are in, also minimizes the learning that the developer must undergo in order to make full use of the tool. The following depiction is captured from the Compendium help files to illustrate:

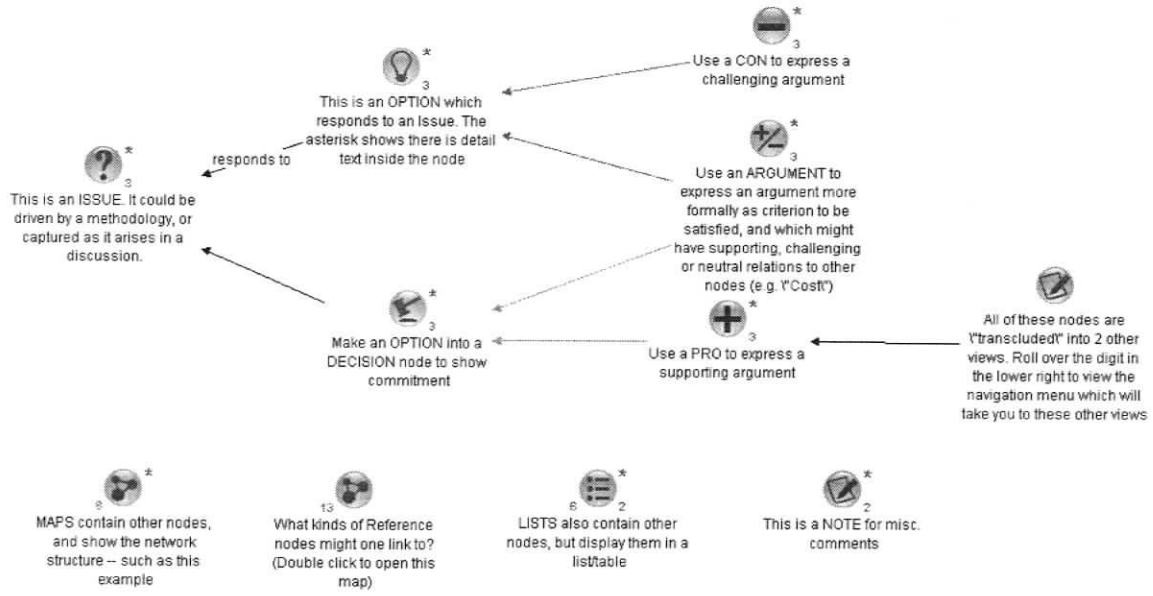


Figure 4. 11 - Node icons in Compendium[11]

[11] Picture Source: http://www.compendiuminstitute.org/Default.htm

Each of the nodes depicted will have a different meaning. The statement or presence of a security concern can be described by the issue node (containing a question mark). Responses to a particular issue node is made in terms of a listing of the responses, and pros and cons of a response, using the option node (represented by the light bulb) for the former and the plus and minus nodes for the latter. Not depicted above but available in the tool is the reference node, which allows for the creation of a hyper-linked reference to a resource that is external to the Compendium tool. The node icon for this is determined by the file extension and is usually represented by the common icon for the particular file type, such as a word document or Adobe pdf file. In cases where the link is to a resource on the World Wide Web, a globe of the earth is used with the three letter acronym 'www' emblazed on the globe.

Unlike a map node which will link into and open another workspace (and the *List View* node which will be discussed later), when any of the non-map nodes are double-clicked a properties box containing more information regarding that node is presented. These properties are divided into four categories: contents, properties, views and tags. The contents pane allows additional information to be entered regarding the node that was not displayed in the map. This allows for a large body of information to be represented in the node, and then hidden until the node is clicked on to reveal its contents. The views pane lists all other maps in which this node is present or duplicated, and allows for direct navigation to that view. The properties pane lists basic information regarding the node, such as creation date, last modification date and author. The last, the tags pane, allows for additional information to be added that will be searchable by the tool. For example, if a security issue is identified that involves the Java programming language; a tag labeled 'java' can be added to that node. A search can then be run later listing all nodes that contain the tag 'java'. The main benefit of this is that it will be possible that a security issue identified in one workplace phase, such as the requirements phase of the construction iteration, may have a bearing on the work done in the next phase, analysis and design. If the information in the node proves relevant, then it can be tagged with the name of the next phase that is relevant. When the developer enters the analysis and design section of the construction phase, they can then perform a search of all the nodes that are relevant to their workplace activity, and the nodes that have been

marked by the developer with the appropriate tag will be displayed. For this consideration, the basic tag set built into the tool (action-item, closed, communication, group, knowledge, location, network, object, open, open-issue, opportunity, problem, requirement, resource, role and task) will be augmented with the names of the 27 workplace activities, so that any given issue or reference can be marked to receive attention in another node, simply by running a search for the name of the node once you enter that node.

To further help anchor each of the 27 workplace activities the short lists of bullet points that were developed in section 4.2.1 was entered into an image file, and then imported into a background picture for each one of the workplace activities. This means that when a developer navigates into a particular activity, they will see a short summary of the relevant position of this activity in the overall context of the software development process.

Another relevant issues is the consideration of HCI (Human Computer Interaction) principles. Norman, in [28], makes a statement, "Don't think technology: think person, task, situation". Based on this, during the construction phases there will be several goals to apply. To begin, the user interface will focus on being simple and intuitive. Icons will be used for visual clues, and navigation through hyperlinks between maps and icons will be obvious. Colors will be kept to a minimum so that they do not 'drown' out the visualization with their presence, and navigation through the hierarchy of notes will be clearly marked, and follow a top-down, left-to-right approach that is common to most users.

A necessary component to the successful use of this tool is the ability to generate lists of information that is meaningful to the next phase, which is explored next.

4.4.3 Report Generation

In order to support development, a necessary part of the tool will be the ability to identify key issues of concern, and then extract them for use or application. The Compendium tool provides several possibilities for doing this. The most common method will be to use the tags that can be applied to each node to markup whatever pieces of information are relevant in that phase. Once done a report node can be created by first creating a list view node. Then using the search function to select all nodes with

a particular tag definition, and then selecting the option to insert this information into the view node. Once done, the contents of this node can be exported as either a web page including graphics, a straight xml format, or an outline format in html with no graphics. These report nodes can be left in place in the view in which they are generated, to annotate issues that need resolution. Removing the tag from a particular node will not remove it from the list, but if any other lists are generated using the same tag variable it will not appear in them. The allows for multiple reports to be generated to capture issues at a specific moment and time, and then for that report to be 'saved' at its particular location in the view in which it was generated for possible future reference, without impeding the generation of future reports that show what modifications have been done (as the issue has had its tag removed) and what are still left to do. Beyond marking the nodes with a tag indicating the relevant phase that the tag should come in the node can be marked with additional tags indicated an open issue, or an issue that has been dealt with, or to paraphrase state information. This can be used for markup and tracking of information as well.

Chapter 5 - Theoretical Construction

5.1 Introduction

In order to begin loading material into the decision support tool, a survey of the appropriate literature is needed. In this phase the purpose is to use research papers to create an ontology of facts and terms in the field of security as it applies to software. This ontology will be used in the initial construction of an analysis framework, which is subsequently applied to a practical analysis of a software project named TAPAS. Experiences from this practical application will serve as feedback, which will be used to revise the framework so that both theoretical and practical analyses are incorporated into its construction.

To identify this literature the process began with the ACM digital library. ACM (Association for Computing Machinery) provides an on-line digital library[12] for organizing and cataloguing research papers relevant to the field of computing in general. As well, ACM deals with publication of papers and organization of annual conferences on topics in computer. Using the digital library, a list of the conference proceedings and conference titles was examined. Two of the conferences, the workshop on New Security Paradigms (NWSP) and the Conference on Computer Communications and Security (CCS) were identified as sources. Another conference that was used for source material was the IEEE Computer Security Applications Conference (CASC). Examining the proceedings of the conferences any papers whose title was indicative of material covering the application of security in the software design process or identification of security faults or flaws was added to a list of potential source material. 32 papers were identified using this process. The papers were then printed out to be reviewed for source material. If the paper listed a citation that was not on the original list but contained relevant information the citation was used to source out and include that paper in the review process.

[12] http://www.acm.org

5.2 Ontology Construction

Following the approach of grounded theory (*see Section 3.5*) each paper was reviewed for terms, observations, case studies and any other relevant information. This is referred to as the data collection phase in grounded theory. A necessary part to the construction of the ontology is the organization of the material. A fellow researcher had suggested the use of a product named ATLAS.ti[13] to provide a tool to visually enter and code the information, as its purpose is to support the application of Grounded Theory. Upon investigation of this tool, however, it was discovered that the free demo version had built-in limitations, and would only allow for the coding of up to 50 facts as well as other restrictions. A preliminary estimate based on the papers reviewed so far indicated that whatever tool was used it would require more than 50 facts of information in the coding phase, and therefore ATLAS.ti would not suffice, unless funds were spent to purchase a license. As a pre-dominant element of the security decision support tool is the need to be adaptable and easily modifiable by future users, open source software or at least free versions of software for constructing the ontology and the decision support tool was considered a mandate in this research by the author. Further searching on the World Wide Web lead to the discovery of a tool called CMAP[14] that met the open source criteria. It should be noted that there are other tools that meet this criteria such as FreeMind[15] that could suffice for this phase equally well. CMAP was simply the author's preference.

5.2.1 CMAP as tool for representation

CMAP is a knowledge modeling tool developed, supplied and supported by the Institute for Human and Machine Cognition[16]. Though not specifically geared for ontology construction using grounded theory, it does provide a visual workspace for entering and collecting information as well as for using visual paradigms to infer relations between them, as depicted in figure 5.1.

---

[13] http://www.atlasti.com
[14] http://cmap.ihmc.us
[15] http://freemind.sourceforge.net/wiki/index.php/Main_Page
[16] http://www.ihmc.us

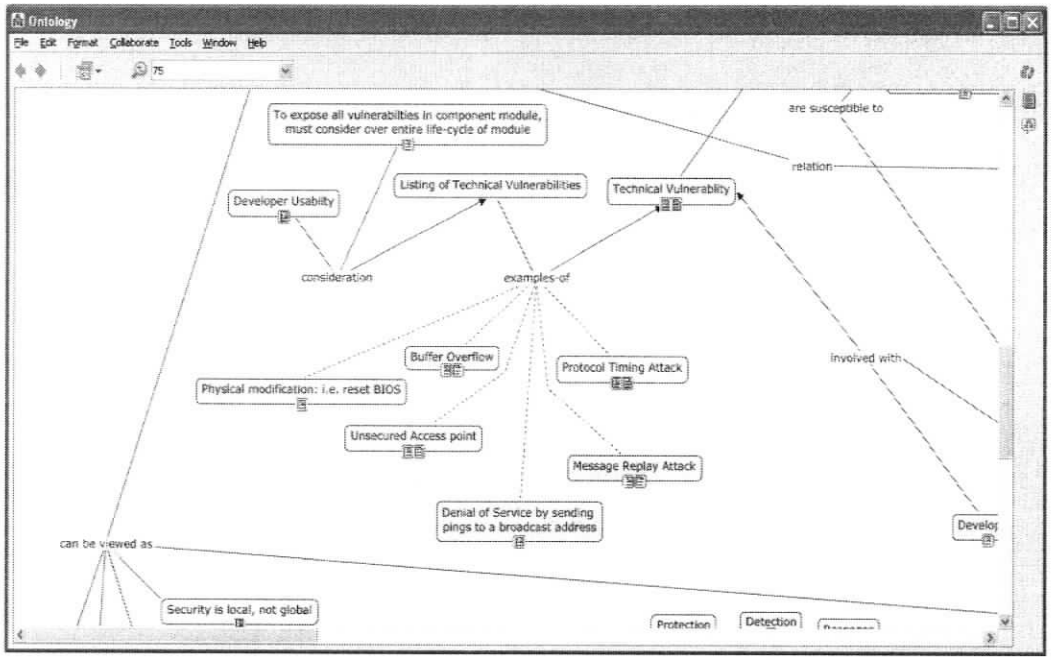


Figure 5. 1 - CMAP presentation depiction

Using the survey of papers relevant points are encoded onto the CMAP knowledge map using rectangles with curved corners. The tool also allows for insertion of links to the source documents which are imported into a library managed by CMAP, and indicated by an icon placed in the bottom center of the rectangle. Association arrows can then be created and named to link like points together, and in cases where there were examples the arrow line was changed from solid to dash as an indicator (see Figure 5.1). Appendix A contains the knowledge derived from these articles to represent the first iteration of the ontology meant to support the theoretical analysis and construction phase. The following sections detail the information encoded on the map.

5.3 Derivation of an Initial Analysis Framework

The next phase of the process uses the ontology map to construct an initial framework of analysis that we can apply to the TAPAS project. Upon inspection of the map, it is possible to overlay three distinct themes in the grouping of information if we draw boundaries as shown in figure 5.2.

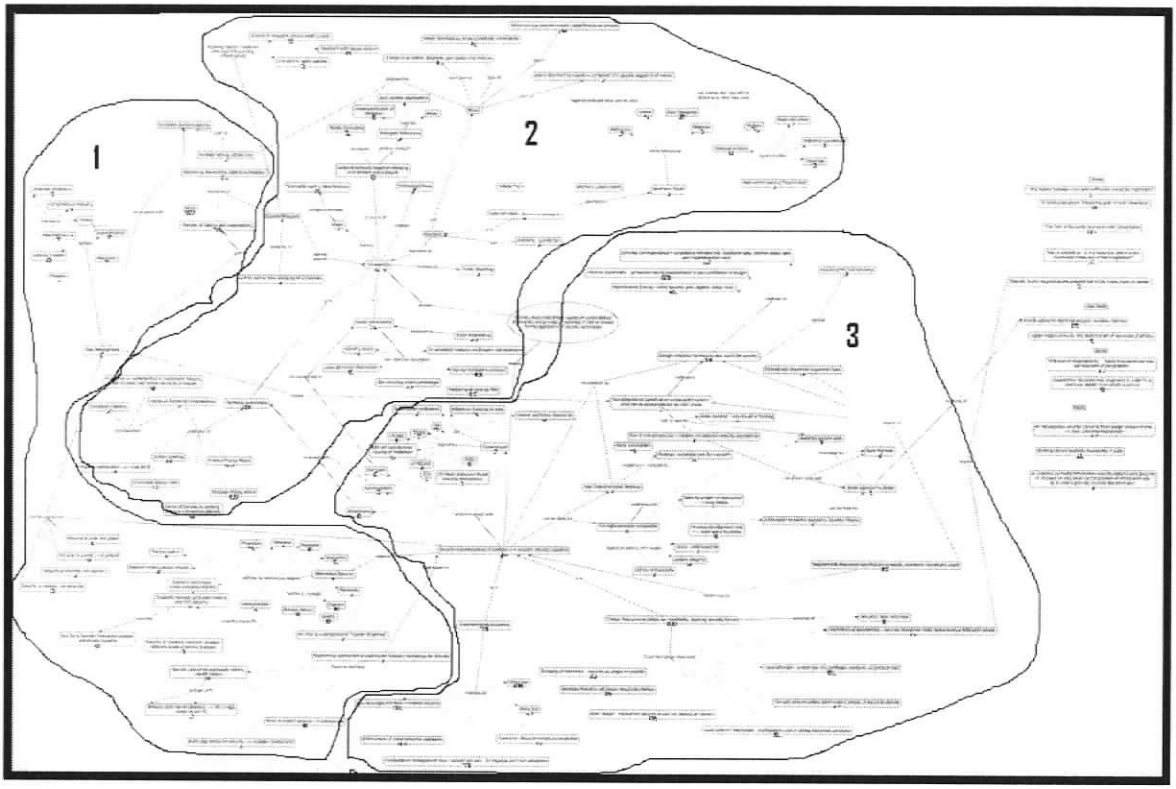


Figure 5. 2 - Ontology partition theme – See Appendix A for a large version

The overlying theme of these three areas shown is 1 – User-Driven Concerns, 2 – Vulnerability Concerns and 3 – Formalized Security Procedures. As the ontology map shows none of these areas are completely disconnected from one another. It is a better visualization of the process if we were to think of these three themes in terms of the mathematical tool of a Venn diagram as shown below.

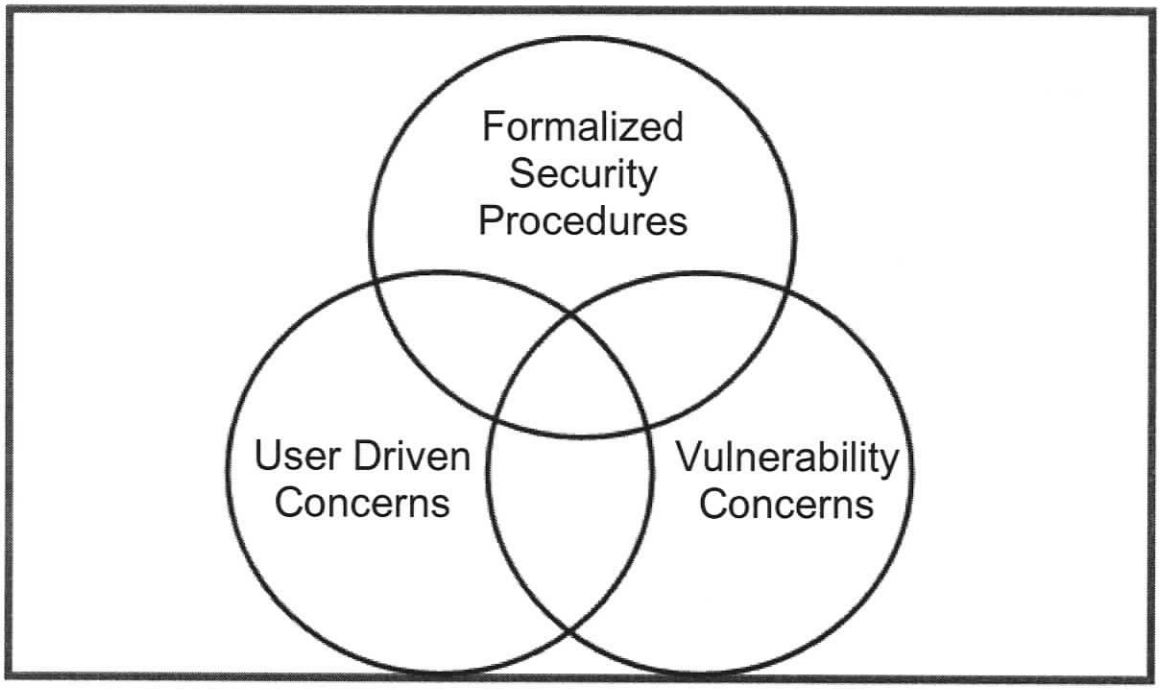


Figure 5. 3 - Venn diagram of Ontology partition

The areas where the themes overlap indicate concerns that bear a relation to both, for example, the area represented by the overlap between User-Driven Concerns and Vulnerability Concerns would represent in part the aspect of social vulnerabilities, a specific example being that of an attacker utilizing social engineering techniques to data-mine a user for information, in order to compromise a system.

It is also important to note that each of these themes is purely organizational in nature, as a way to briefly describe a category. There is far more meaning and definition to the named theme then is implied by the title, and the sections that follow this one delineate the meaning of each of these themes in detail, as well as laying out a preliminary foundation for an analysis framework.

Having identified these themes as a starting point, the next step is to break down each of these themes listing the pertinent notes, and providing an analysis of their relevancy in the context of the decision support tool.

5.3.1 Vulnerability Concerns

When dealing with the concept of vulnerabilities in the security paradigm, it is important to note that an underlying premise is the determination of whether malicious or non-malicious intention exists. This premise works hand in hand with the two different classifications of vulnerabilities, social and technical, that are described in [12].

Social vulnerabilities are tightly coupled to the user in the context of software. These can include physical parameters as mentioned in [12], such as leaving the computers unlocked or more specifically leaving the computer terminal logged in and the user then walking away, leaving the terminal open to use by any person who happens to walk by.

> Context: The physical parameters need to be screened for their relevancy to the software developer. Guaranteeing whether or not a room is locked is likely beyond the power of most developers when they have no physical control over the third parties that have purchased their software or the buildings/rooms in which it will be used. More relevant to the issue is the note regarding whether a computer is left unattended and logged in. This vulnerability would suggest that during the inception phase at the start of the project the customer should be questioned regarding what their expectations are in the prescribed scenario: should the software automatically revert to a login prompt after a certain period of inactivity? Also if there are expectations that a client using the software will be constantly pulled away from their desk for whatever the reason should there be a button or perhaps "magic' key combination that with one action (i.e., depressed mouse button or key press) can put the user back into a login screen thereby expediting the process. An expansion on this notion of physical parameters would also be the size of the font used to display information, and potentially the color scheme. A large font with sharp contrast between the foreground and the background color would make legibility of the information from a distance much easier for someone walking by another's' terminal. Though this would seem admittedly somewhat paranoid, depending on the nature of the information being dealt with, it could be a consideration. These considerations would need to be

identified in the inception phase. Using the framework developed in Chapter 4 we identify the question of vulnerability in physical exposure would fall into the Environment map node of the inception phase as non-functional requirements. As shown below, this information is then entered into the map for this node:

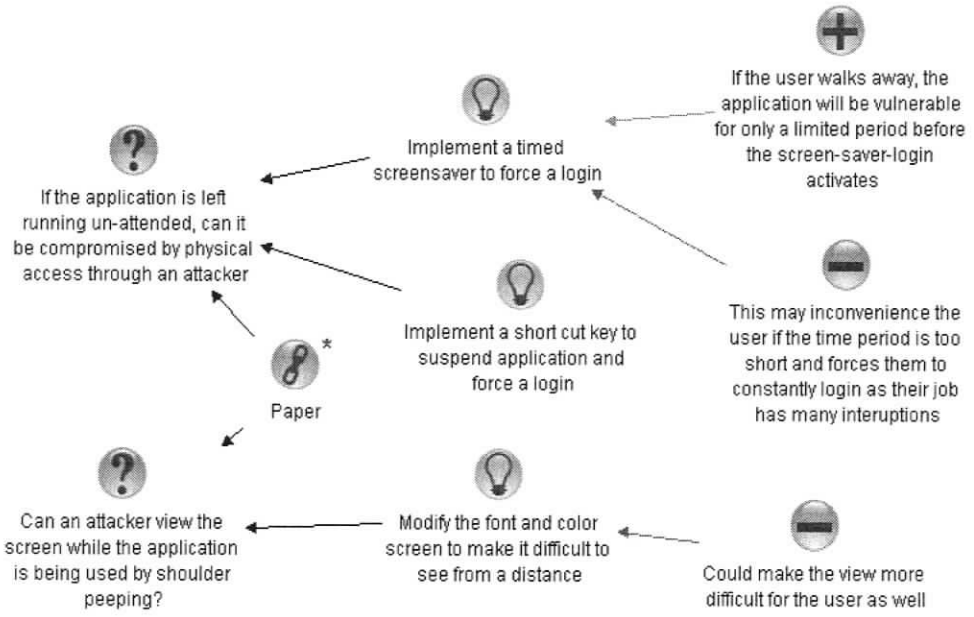


Figure 5. 4 - Representing information using Icons

Each vulnerability can be represented as a question node. The possible resolutions to this vulnerability are depicted as option nodes, with the positive and negative elements to consider shown in the plus/minus nodes. The paper from which this information was derived[12] is then added as a link node, with the bibliographic information entered into the node. If the user places the mouse over the Paper node the bibliographic information will appear in a message box at the bottom of the screen or in a 'Tool-Tip' pop-up text box if they move the mouse over the asterisk in the upper right corner of the icon. Alternatively the user can double-click on the paper node which will pop up a window containing the same information.

There are also usability issues in terms of user-perceived obstructions to the implementation of security [11, 12]. This notion of usability can be decomposed in terms

of the level of knowledge a user is expected to have in order to successfully use a product, or what specific task they are trying to accomplish and whether they consider the extra overhead of utilizing the security features worthwhile. This is not restricted to end-users who lack knowledge or patience. It can also include inadvertent mistakes on the part of systems administrators when they are implementing security features.

Context: The notion of usability plays a large role in all phases of the software development process. To help break this down it is useful to list scenarios:

1. The end user of a product is usually comforted by the presence of security features and it helps to instill confidence in the software product. An example of this is the pop-up message box in a web browser that indicates to the user that they are about to be redirected to a secure web site, perhaps one at which they will be required to enter their credit card information if they are purchasing an item. By having the box in place in the browser software, confidence in the product is instilled for the user. This could quickly fall apart however, if in the next window the user was then asked for enter their public key certificate in hexadecimal or binary which would likely confound the average user. The question of how 'visible' or transparent does the security need to be in order to instill consumer confidence, or the business value, is one that should be placed in the Business Modeling node of the inception phase. Regarding the usability level required of the security features and their presence: this is appropriate in the Requirements node of the inception phase
2. The example in (1) can be translated to the experience of a system administrator or even a third party programmer making use of commercially purchased software components. The documentation surrounding security features and code, how it is set up and used and the ease of use all need to be examined. This point can be added as a note in response to the note made

regarding the use of help files and also be related to the question regarding what roles are to be played in the system, both of which are located in the Business modeling node of the inception phase.

Beyond the physical system containing the software there are also the aspects of social engineering which are exploited by attackers attempting to gain access to a system to execute malicious behavior. Examples of these can range from a phone call to a targeted user in which the attacker attempts to acquire information on a system by tricking the user into revealing details of the system. Another common but less elegant technique is that of Dumpster diving - where an attacker will literally comb through mounds of refuse in order to find information on a system they wish to compromise.

Context: Like the physical parameters that were previously discussed, social engineering is a difficult topic for a software developer to address, as it oftentimes doesn't involve their software product specifically, but instead is intended to compromise the user. There are some relevant aspects that can be considered:

1. With regard to 'Dumpster Diving' it can perhaps be circumvented somewhat by attempting to limit what information can be printed out, and perhaps by even adding headings to sensitive information on each page indicating that it must be sent to shredding rather than disposed of in a standard garbage or recycling bin. Perhaps also in the case of sensitive information a unique number can be printed at the top of each page which internally would identify the user to the company (the user's login id would not be appropriate in this case). If this were done and pages of sensitive information surfaced in a venue in which it did not belong, then it would be possible to track back to who was responsible for the leak and take corrective action. It may also be appropriate to prefix a header on each page stating 'Confidential material – handle with care-must be shredded upon disposal". In both these cases scenarios should be identified

during the Environment node of the inception phase, and then marked for discussion with the customer during the Business Modeling node in order to ascertain what approach should be implemented.

2. Another possibility is that, with regard to password security, an attackers attempt to "information surf" during a phone conversation, in order to gain information to compromise a password. Potentially warnings could be place in software, such as pop-up information windows, which indicate or inform the users about the potential threat giving information out indiscriminately, which could compromise a system. It would be unclear how effective this would be however, as common experience indicates that most users will ignore such warning boxes (like the EULA, End Users License Agreement, for installing software). An option might be to list, as a course of action, a suggestion to whatever company is purchasing the software to advertise or advise their internal users as to this threat, which would then mitigate the risk, or at least transfer it to the customer's responsibility. In order of placement in the tool, it would be reasonable to produce documentation, either hard copy or electronic in some form, that would make end users aware of the potential risks of an attacker. It could even go as far as a presentation intended to educate. The construction of this document would take place in the Deployment node of the transition phase, as this is the point in development where not only would all of these issues would have been identified from earlier phases, but also where the customers are involved in taking ownership of the software project.

Before discussing the details of attackers, it is worth considering what they tend to use to break into the system once they have sourced out or acquired the relevant information. This brings us to the classification of the technical vulnerabilities or holes in the security

of a system, either physical or present in software that can be used to compromise the system. There are many examples of these flaws, a sample of which are listed in the following table:

Technical Vulnerability	Description
Buffer overflow	Overwrite a programs executing code with your own by feeding more information into the program that it was meant to handle in a variable field. [3] discusses both Stack and Heap based attacks.
Message Replay attack	Eavesdropping in a network environment on the information packets and then resending some of the captured packets to compromise a system
Denial of Service	Flooding a designated target machine with packets in order to slow its performance
Protocol timing attack	Using the logic of a program to learn more information about a system by measuring its response time
Unsecured access point	Finding an access point into a system (such as an unsecured wireless router) that will bypass the standard security protocols
Physical modification	Reset the BIOS by removing power to terminate the administrator's password protection

Table 5. 1 - Listing of Technical Vulnerabilities

All of the above technical vulnerabilities represent targets that can be exploited by attackers.

Context:

1. Buffer overflows are a common vulnerability in operating systems, stand-alone software applications and even network communications. During the Analysis and Design node of the inception phase the hardware, operating system and programming language(s) that will be involved in the project should be

identified. From here a list of all known vulnerabilities in these three areas should be constructed and then tagged to appear in a list node during the implementation phase: any area's which may be vulnerable should be checked and tested. In addition if the list of vulnerabilities in the Analysis and Design node is sufficiently large based on some pre-determined thresh-hold decided by the development team, it may be prudent to examine if choosing a different component for the hardware, operating system or programming language may produce fewer vulnerabilities that would then need to be dealt with in the implementation phase. This should also be a question of consideration in the Requirements node in the Inception phase as parts of the risk analysis, i.e., what platforms are vulnerable and can we choose a different operating environment in an attempt to reduce the risk? Of consideration as well is the fact that the operating systems, programming languages and hardware may be beyond the ability of developer to control if they are pre-determined by the requirements of the customer. In these cases techniques for dealing with injection of code intent on exploiting a buffer-overflow attack need to be addressed in the Analysis and Design node as these types of attacks account for over half the vulnerabilities reported each year[3]. These techniques can be either proactive in terms of actively trying to detect if code injection has occurred[3] or reactive in terms of creating a mechanism for self-monitoring of execution code through the use of check-summing[13].

2. The message-replay attack is exposed as part of a defect in the security protocols implemented in a network. As part of the Analysis and Design node of the Inception phase it should be noted that if the software involves or potentially could involve network communication at some point a protocol should be

chosen that will not be vulnerable to this attack. Further to this it should be investigated whether that protocol has been implemented correctly and that the algorithm for the protocol has undergone sufficient public scrutiny to ensure its security. It should be noted that the conceptual theory of this attack does not necessarily need to be constrained to a network scenario. If the application communicates with a third party component such as a database installed on the same system, the theory behind the message-replay attack can be applied to that scenario as well.

3. The Denial of Service attack (DDOS) is common in a network environment where a server provides a service, and due to packet flooding the service becomes unavailable. There are techniques available to combat this in the Analysis and Design node of the Inception phase if a network service is indicated as being required. A plan should be implemented and developed to prevent this attack based on existing knowledge/literature.
4. A protocol timing attack is dependent on both the environment in which it operates as well as the software it is attempting to compromise. The environment may not be part of what the software developer can affect if the company has pre-existing systems on which the software must run, however the software most definitely can be adjusted. The protocol timing attack relies on the difference in time that different paths of software execution can take. A simple example would be a login software module that consists of both a user id and a password. If the code is written as follows: first check the user id against a database, and then if the user id is valid proceed to the next section to check the password against a database and if not, return an invalid login attempt error. An attacker measuring response times for attempted logins might see a significant difference in time between a valid user id and an invalid one, and thereby be

able to determine between a valid user id and an invalid one. During the Implementation node of each phase all algorithms that deal with any security validation should be examined and then coded so that the response times are the same. This will ensure that there is no potential 'leak' of information to an attacker.

5. An unsecured access point can be introduced at any time into a system, and could therefore allow unauthorized access into the system. The most common circumstance of this occurs when a wireless router is added to an existing network by an end-user. The Environment node of the Inception phase should identify this vulnerability, and propagate the information to the Business modeling phase for discussion with the customer on an appropriate response. Preventative steps to take against this will be more challenging and will range from simply notifying a customer to enforce among their employee's that no third party hardware devices can be connected to the other extreme, which would include all network traffic being monitored to look for suspicious patterns, all network ports are locked down to a single MAC address and IP, machines are allowed to communicate only on specified ports, and complete paranoia, the assumption that all new things are security risks until proven otherwise, is strictly enforced with regard to any new machines that are connected into the system. In the former case a note is added to the end user security training documentation that is listed in the Deployment node of the Transition phase in case this needs to be included in the training document. In the latter case a note identifying this problem is added into the Environment node of the Inception phase clearly indicating that if this vulnerability is a possibility then this information node associated with this issue should be tagged for propagation for discussion with the customer in the Business Modeling node of the Inception phase.

6. Physical modifications, like the physical parameters mentioned earlier, are not solely the realm of the developer, however in such cases where this level of security is required in the system, the software can be set to attempt to access the BIOS using the specified password, and if unsuccessful can be set to issue a warning or trigger some type of failsafe. It would be appropriate during the Requirements node of the Inception phase to ascertain from the customer if this level of assurance is required, and also determine if such access could compromise the software system if it was achieved, and if there are ways to circumvent it.

The notion of physical modification is not limited to resetting the BIOS password, and should an attacker be able to access the hardware necessary to reset the BIOS password, it is likely that the attacker can remove the hard drive itself and hook it up to another machine to begin accessing it. How the application should deal with this scenario merits exploration as well. The key question is whether the application should encrypt parts of its code and all sensitive information, so even if an attacker were able to directly access the hard drive they would be unable to read the information without the corresponding decryption key. This question is added to the list generated in the Requirements node of the Inception phase, and if the result is positive it should be tagged for attention in the Analysis and Design node of the Inception phase.

Although the key part of a technical vulnerability may be the exploit itself that allows the system to be compromised, the human component which acts on the exploit is represented by what is termed an *attacker* which has been used so far simply to define some-one attempting unauthorized access. Attackers can be divided into two distinct groups [26]. The first group is referred to by the term Gremlins. Although we can classify any compromise of the system as inherently malicious due to the invasion of privacy, an attacker classified as a Gremlin does not carry out any destructive behavior beyond compromising of the system and oftentimes the motivation for a Gremlin's actions are ascribed to the statement "...just for fun". The second classification is that of the Hard-intruder. A Hard-intruder has specific targets or goals and has garnered a sometimes more controversial name, that of the cyber-terrorist. DARPA, the United States Defense

Advanced Research Projects Agency, has published a threat model detailing the specifications for a cyber-terrorist[26, 33]. In addition, a more general model named the Adversary Model is described in [32]. This model describes the characteristics as well as common roles that the attacker may present. The four main criteria to take into consideration of the attacker are: the resources available, what access is available (both physical and cyberspace), what the objective of the attacker is, and lastly a measure of the risk tolerance, or essentially an analysis of the risk – reward ratio. The roles ascribed to the attackers are used to attempt to categorize their motivation, and as well to assist in defining the characteristics. Common themes are that of the industrial competitor, organized crime, malicious employee (or ex-employee as is often the case) or even government sanctioned espionage.

> Context: Although the Gremlins and Hard-intruder have different motivations and arguably non-malicious and malicious intent respectively, it would be appropriate for purposes of the Decision support tool to focus on the threat assessment in the risk analysis during the Requirements node of the Inception phase. Of concern should be what information is considered sensitive, where and how is this information present in the system, what level of effort needs to be made to protect it and what requirements will be needed for the protection. The DARPA threat model can be used to help assess the cost of the protection by using it to determine an escalating scale of the cost to the attacker to carry out the compromise versus the cost of protecting the information. As well the common themes should be taken into consideration during the planning stage, especially with regard to a malicious employee. Extra data backups may be required and potentially extra security features to ensure that the damage an employee could cause would be minimized.

The practice of carrying out an attack can be generalized into a three step process[32]. The first step of an 'attack execution' is to diagnose the attack itself; this means attempting to determine what is the motivation or purpose of the attack, the risks associated with it and what resources can be brought to bear. From there the second step is to gain access to the target system, or put another way, find a vulnerability to exploit

and then use it, which is followed by the last step of the attack, the actual execution. Most attacks can usually be described by three parts[37]: a single exploitable vulnerability, the capabilities that must be present in order to carry out the attack concept, and lastly the *attack signature,* which is the specific sequence of events that the attack is composed of.

A key problem, mentioned prior, in security in general with regard to software that we can clarify in the context of vulnerabilities is that a vulnerability can vary dynamically based on the changing environment in terms of hardware/software and also exposure, referring to both physicality as well as cyber-space presence[11, 36]. Relevant to this is the concept of *emergent behaviors* [16]. These are vulnerabilities that arise due to the evolving environment that were not present prior to some addition or change in software (although it is possible to ascribe this to poor context assumptions as well as under specification of behavior).

> Context: Attacks that are carried out often make the media announcements and as well are documented by security websites that track vulnerabilities and exposures. Once the phase has been completed, a survey of several appropriate websites searching for attacks specific to the domain of the software project should be preformed. The attacks should be broken down into the three step process and it should be documented how the software breaks at least one of the steps, so that the attack cannot take place. If the software doesn't break any of the steps, then that particular attack needs to be dealt with in the next design iteration, and once that is complete, the same process repeats. It should be noted that the tradeoff here will be the time needed to survey and evaluate the attacks, and the revision required to the project in order to deal with an attack. This step should be carried out in the Analysis and Design nodes of the Elaboration, Construction and Transition phases. The reason for leaving it out of the Analysis and Design node of the inception phase is derived from the aforementioned 'emergent behaviors'; during the Inception phase, much of the architecture and design is still under development. Any attacks that cannot be 'broken' by the current software should have the relevant code identified, and then

earmarked for modification to provide the break, in the Implementation node of whichever phase the project is currently in.

5.3.2 User-Driven Concerns

User-Driven Concerns is a term used to categorize both end-users of a system as well as software developers who may be making use of components of a developed by a third party. The main cross-over between this and the vulnerabilities theme is through the aspects of social engineering and usability that were previously introduced. It also crosses over into the theme of applying security for most of the other topics. The three main topics of User Driven can be described by: security as a component of "Quality of Service"[18], risk management[8], and engineering approaches for usable, or user friendly, technology for security[36].

Risk management represents perhaps one of the most complex areas to be dealt with. It is a concept that must be carefully weighed by a software developer in order to determine not only what level of security to implement in the product, but also what areas to forgo applying security to and therefore expose the company to a negative user-response should the software be breached. There are several possibilities when dealing with a risk management. Probably the most contentious issue is that of liability transfer[8]. In this scenario the developer must weigh the potential cost of not implementing a particular security feature. They must take into consideration the various compromises that could be achieved such as the improper disclosure of information for information that is destroyed or lost or even information which has had its integrity compromised, meaning that it has been modified in some way by a third party with malicious intent. All of these factors can be subsumed under the heading of information risk. Beyond this the developer must also consider human factors from the user's perspective, i.e., should the software provide for any form of backup for a data retention and if it does not, will the user consider the software product to be at fault, should this circumstance occur in which data is lost. This would be dependent on the type of application involved, and the purpose of the application, as well as whether the user would expect the application to maintain backup copies. Zurko cites the following as common questions end-users ask when dealing with a system[42]:

- What could go wrong?
- How likely is it, and what damage would it cause to me or to others if it did?
- How would I know if something went wrong?
- What reason do I have to believe that it won't?
- Who is responsible to ensure that it doesn't, and what recourse do I have if it does?

All of these questions posed by an end-user tie directly into the usability of an application with regard to security and user perceptions. A security breach, caused by a user's failure to utilize a security feature that is implemented, is not usually perceived by the end-user to be a consequence of their own actions. It is viewed rather as a failure on the part of the software to either A) inform or instruct the end-user in how to correctly utilize the security features or B) protect the user from the vulnerability regardless of whether they've implemented the feature or not.

> Context: The issue of liability transfer most suitably belongs in the Requirements node followed by its presence in the Business modeling node of the next phase. During the initial Business Modeling node in the Inception phase, security as an issue of software development needs to be raised with the customer. Two key points the developer must resolve are: exactly what the customer perceptions are with regard to security- to verify their level of awareness of the subject, and, what their expectations are with regard to security and the software product. From this a list of security issues needs to be generated in the Requirements node, and as well the cost of implementing security to resolve these issues needs to be ascertained so that the customer can indicate in the Business Modeling. As alternatives are often presented to customers and they choose whether or not to pay for the development of a certain feature, it is appropriate to ensure that the customer understands that they will take on the liability of failed security should they option to not have a particular feature. Also in the Requirements node a list of all implicit assumptions that are being made should be constructed, i.e., data retention, how often passwords should be changed, etc. and they should be made explicit meaning formalized in documentation. These assumptions should be verified/refuted by the

customer as well as the software developer, and then passed onto the next appropriate node through the use of tags.

Regarding the issue of end-user usability, the questions posed by Zurko are listed in the Test nodes of all phases in order to provide the developer with a means to ground their perspective into that of the end-user so as to gain insight into the usability.

The concept of risk also plays into the area of user-friendly software both from an end-user's perspective, as well as an intermediate developer who is using tools provided by another developer. There are several approaches that in some cases are orthogonal in this aspect. An argument is made in [18] that security should be application-centric, not specific to an operating system, so that it is better able to deal with constantly changing architectures. This is contrasted by counter arguments made, such as in [36], which indicate that to be more consistent, security infrastructure should be re-factored so that it is implemented in the operating system. This would in theory provide a more consistent, application of security to key software that sits on top of the operating system. Other paradigms are also promoted suggesting that security should be integrated in both applications and operating systems, so that it is transparent to the user, thereby providing the most user friendly approach. From the developer's perspective an argument is made that security should be constructed as a reusable components, the analogy used is Lego bricks, to facilitate the means of their integration with software. All of these extend into the paradigm of security as a component of the quality of service.

Context: In the Analysis and Design nodes of all phases when a determination is made as to specific requirements for the project, both the operating system, hardware and programming language should be examined and a risk analysis performed on where to implement security features, what security features are already implemented, and what vulnerabilities are known. This balances into the transparency, usability and presence criteria mentioned previously when discussing vulnerability concerns. Risk should be factored into this balance as part of the three criteria being weighed.

When dealing with quality of service, it is not considered a binary state so much as a trade between different levels of usability. Oftentimes security or more specifically

application of security techniques is resource-intensive[18]. This of course ties in to the availability of the machine to service the user's needs if usage of the machine is in a constant competition between the application and the security modules that are running in parallel. Oftentimes there are numerous policies and paradigms that must be waived from the user's perspective with regard to security. Some of the common ones are the level of protection, the level of detection, the level of response, what level of information security (such as sensitivity, content, quality, and release history[27]) is provided, as well as proactive deterrents such as the level of unhelpfulness which is designed to frustrate an attacker.

> Context: In the first iterations of the Implementation and Deployment nodes if possible (and where applicable) appropriate performance measurements should be done on security features that have been implemented, and these measurements used as feedback and rolled into the next planning iteration as part of the cost of implementation of the security features. In the Business Modeling node of the next phase the performance figures should indicate whether continuing to include the security feature under examination is valid or too expensive, and if it is too expensive, it should be re-evaluated as to whether it should continue as part of the software project, be removed from the project (thereby increasing the risk/exposure) or needs to be re-done to increase the performance. With regard to 'Unhelpfulness' as a security feature this was added to the Analysis and Design node of the Inception phase as a possible resolution to a security issue.

5.3.3 Formalized Security Procedures

The largest of the three themes with significant overlap with the other two is that of applying security to a software process model. The main goal of Formalized Security Procedures is that of security assurance, which is the level of confidence the developer has in security related properties[5]. This can be broken down further into implementation assurance, requirements assurance and design assurance. It is also important to indicate that applying security also encompasses the design aspects. This includes when and how to consider security in the design process and the previously

mentioned concept of risk which overlaps with both user-driven concerns and vulnerability concerns.

5.3.3.1 Security Assurance

Implementation assurance refers to specific tools or techniques that can be used to support the concept of security assurance. Examples of this are: the use of high level languages which have inherent security built in such as the Java programming language and it's notion of the "sandbox", configuration management tools such as version control (CVS) to organize and track alterations, code analysis tools that specialize and focus on known vulnerabilities, and both white box and black box testing techniques. All of these are methodologies designed to support and improve the application of security.

> Context: The decision of what language to use should fall into the Analysis and Design node of the Inception phase. The client will need to specify what environments that the software will be expected to operate in (i.e., stand-alone, networked, mobile device, etc.). Then, in the Analysis and Design node, a specific language or potentially languages that would meet the needs of the project would be chosen. Inherent to this should be an evaluation or listing of the language specific security issues. Configuration management tools such as CVS should be part of the developer's toolbox from the inception of the project through all the stages and again, this should be considered in the Design and Analysis node. It should apply not only to code but to all the documentation produced in the project. It should also detail and track all decisions made with regard to security.

Requirements assurance deals with formal verification of whether the specification is considered to be complete, consistent, and technically sound. It should be noted that this type of assurance relies heavily on an almost complete specification already existing which describes in detail all of the security ramifications, software security implementations and usability and risk analyses.

> Context: A document describing what the client feels is required and sufficient for security can be developed in Business modeling node of the Inception phase, reviewed in the Requirements node of the same phase and then checked in the Deployment node of the phase. It should be revised and

reviewed with the customer through each iteration as this will help move some of the liability of security to the customer as well as increase their awareness.

Design assurance is most often reflected in the architecture of the software such as a presence of a security kernel, or layering of the software and the presence of modularity in the architecture. There are several techniques for design assurance. The first is the notion of economy of mechanism or that security should be as simple as possible in terms of software and architecture. The second is that of complete mediation which, simply put, means that all access by the application being built, whether to information resources or to other applications, should be checked by some mechanism. The third is that of open design which means that any security mechanism should not depend upon secrecy. Cryptography is perhaps the most obvious example of the application of this paradigm. Software code for use in cryptography is not considered secure and reliable unless the code that has implemented the cryptographic protocols has been publicly reviewed and evaluated to ensure that it is secure. The fourth is the concept of least privilege in which rather than granting generic access rights to a user or process only the minimal privileges needed to accomplish the specific tasks are given. The fifth is the notion of a fail-safe defaults – that unless the user or process has been given explicit access, it should be denied. It should be noted that this is in contrast to the normal software or hardware development process in the realm of fault-tolerant computing. These two paradigms can have orthogonal goals. The purpose of a fault-tolerant system is to survive failure of either software or hardware and continue to provide some level of service. Implementation of this policy can lead to fatal security flaws. An example is a network router which contains both hardware and software. Applying the fault-tolerant perspective to this scenario may require that, should the router become unable to resolve addresses due to a fault, the default mechanism should fail gracefully by becoming a broadcast hub, forwarding all incoming network packets to all outgoing ports. This would ensure that a level of service is still provided by the router. In this scenario, however, by devolving into a broadcast hub a machine which previously had an invalid address, and was therefore ignored by the router and which was the property of a malicious attacker, would now be able to get access into the network due to the lack of

address authentication[26]. The sixth tool listed of design assurance is the use of auditing or logging to create an examinable trail of information for discovering incorrect behavior, either on behalf of the user, or the application itself or of some third party communicating with the application[41]. The seventh and final tool for design assurance is that of the least common mechanism. This prescribes that mechanisms used to access resources should not be shared, so that should be a mechanism be compromised it cannot be redirected to provide access to another resource, instead of the one for which it was originally intended. There are additional tools and techniques for design assurance and related concerns. An example of design assurance in architecture is that of the security check technique described in [31]. This technique proposes a framework for isolating parts of a system, to perform a security analysis by searching the isolated part for vulnerabilities. The reason for this isolation is that analysis of large systems can result in a near exponential explosion of possible states, making it impossible to perform an analysis of the whole system in a reasonable amount of time/cost. By focusing on targeted areas or components of the software, such analysis becomes feasible. In addition there is the use of the MOPS (Modeling checking Programs for Security properties) tool described in [34]. MOPS[17] is a static analysis tool that can check a program for violations of pre-defined security properties. It runs against programs written in the C programming language.

Context:

1. Economy of mechanism should be dealt with in the Analysis and Design node as well as the Implementation node of all phases. It may be relevant to have examples of common security techniques and protocols/code to create a body of knowledge to draw on for implementation of these techniques as well as somewhere to capture new information as it arises.
2. Complete mediation should be identified in the Analysis and Design node of the Inception phase. As anything to do with Network communication is commonly known as a high area of security risk, extra concern should be enforced during the design

[17] http://mopscode.sourceforge.net

to make sure security issues are identified, recognized and dealt with.

3. Open design should be utilized during the Implementation node of all phases, if it's not proprietary. Anything to do with Cryptography should rely on public libraries that have already been heavily scrutinized for possible flaws in the algorithms or implementations. This also adds a question for consideration into the Analysis and Design node of the Inception phase, whether the programming language being chosen for the project has built in support for cryptographic protocols and network security.
4. The Requirements node of the Inception phase should identify all sensitive information and exactly what access should be provided for it. Then, in the Analysis and Design node, the design should implement protocols to provide the specified access based on the paradigm of least privilege.
5. Fail-safe defaults should be identified in the Analysis and Design node of the Inception phase. Once the scenarios in which a decision between failing safe or failing secure must be made are discovered, they should be brought to the client in the next Business Modeling node and the client be asked to make a choice, thereby removing the liability from the developer.
6. The use of an audit trail or logging mechanism can be invaluable in tracing problems, security or otherwise. In the Analysis and Design node on the Inception phase the question of whether to integrate this mechanism into the software product should be asked. Some of the potential contexts in which this might be a valuable option are: monitor end-user activities to detect malicious behavior; monitor communication between the application and all other applications to detect security problems.
7. The least-common mechanism principle should be checked during the Analysis and Design node of the Inception phase to

ensure that no overlap exists and that privilege escalation cannot occur.

The different tools described, Security Check and MOPS, can be of use in the Test nodes of the different phases to ascertain if the code contains security issues. An issue that would need to be evaluated is whether the time investment in applying these tools would be cost effective in the scope of the software project.

5.3.3.2 Security in Design

When security should be considered in the design process is a critical consideration. How it should be implemented or even whether something should be implemented, if it falls short in the risk analysis of cost versus benefit/potential exposure, are questions fundamental to the overall topic.

With regard to the timing, there are generally three distinct intersection points within the design process. The first can be best categorized by the term 'pre-integration'. In this scenario security considerations are placed at the same level as other problem requirements: they are considered an inherent part of the design from the initial outset. The second is the 'incremental' or iterative approach, in which security is considered and added during the iterative development cycle of the production code. The third can be classified by the description 'post-implementation patch'. There are two perspectives to this: the first is simply a response to a security flaw that has been identified which requires a modification to the code in order to fix it. The second is an attempt to inject security into an existing system where it has been indicated that security is desirable and it becomes an add-on feature.

Context: Both pre-integration and incremental are part of the spiral model and therefore are already sufficiently covered by the framework of the tool as laid out in Chapter 4. With regard to patches, the vulnerabilities or problems are identified during the Deployment nodes of each phase, and then tagged for repair in the next phase. A post-mortem should be performed on why the patch was needed (i.e., where did the security hole come from and why wasn't it discovered earlier) and that knowledge

incorporated into future iterations as well as other projects which use the tool.

There are several considerations to the pre-integration approach. Though endeavoring to treat 'applying security' as a fundamental part of every software project present from inception is a laudable goal, like most typical software construction at the earlier stages in the development lifecycle, security itself will be subject to abstraction and therefore necessary if not critical details may be missed [31]. Other issues are: considering security aspects can significantly increase development time and lead to an explosion of the state space making the project untenable, and that also by fixing the security criteria at the outset of the project does not account for the fact that the information revolution is outpacing the security criteria, which in fact may be dated, inaccurate or simply dead wrong before the completion of the project[17]. Another aspect to consider is that such an approach is based on an idealized engineering development process that was inherited by software developers from other engineering disciplines labeled the waterfall development lifecycle as described in Chapter 3. This model assumes concrete development steps with no opportunity for revision or taking into consideration the flexible and dynamic nature of software. It has been proven that this approach when strictly followed does not work in the world of software development[20, 24].

Context: Incorporating security iteratively into the spiral model will likely increase the development time, however, security should be considered an indispensable part of the software development process and not an add-on feature. Involving the customer during the Business modeling nodes of each phase in the decisions surrounding security will also increase their awareness of security and help offset liability for the developer.

The second approach, that of the incremental or iterative application or injection of security into the design process, has met with the greatest amount of success. As previously mentioned this is sometimes referred to as the spiral model[17]. AEGIS[12], mentioned in Chapter 2 as related work, is an example of a software engineering methodology which attempts to integrate security throughout all phases of the software development process, including the cross-over to User driven concerns involving both risk assessment as well as usability concerns. Abrams in [2] formalizes the notion that an

evolutionary acquisition model needs to be built into the design process with regard to security, and that such a model makes a strong case for there to be a defined process of change in the software architecture.

Context: This supports the development of a decision support tool involving the spiral software development process which is the subject of this work.

The third approach, essentially a reaction to the need for security either due to a mandate or the discovery of vulnerability, is perhaps the most common but least proficient. In the case of a vulnerability that has been exposed as a flaw in the system the most common remedies are either to provide for a software patch to fix the specific hole, or if possible to re-design the vulnerable part from scratch to remove the vulnerability[31]. In both responses an implicit assumption is that the vulnerability is localized enough in the code for it to be fixed by a modification to that area only and that such a result will not cause a cascade of fixes or alterations to other parts of the software.

Context: Vulnerabilities and security concerns that require 'fixing' in the code are identified in the Deployment nodes of all phases. It would be appropriate to add a question in this node that when such concerns are discovered, a determination should be made whether to patch the existing software, or go further to determine if the code under consideration needs to be redesigned from scratch. Regardless this concern should then be tagged for consideration in the next Analysis and Design node.

All three of these approaches are subject to design validation techniques to support pre-integration, incremental or post-implementation approaches[5]. These techniques can consist of tracking informal correspondence to ensure consistency between external functional specifications, internal design specifications and the implementation code. Also requiring formal arguments that go beyond requirement's tracing to gain confidence in the design, as well as the requirements tracing technique itself to verify the security specification against the design specification, assist in the elevation of security considerations in the design process.

Context: The informal correspondence is an elaboration on the earlier inclusion of dealing with documentation for the code and design of security related modules that was added into the Analysis and Design node in all

phases. This point can be added now to add detail to the documentation that should be produced and tracked in this node.

It should be noted that in all these areas the concept of risk must be weighted and taken into account with respect to the actors involved with the software development process. The conventional or legacy approaches to security can be invalidated by a risk-driven process in cases where the risk analysis makes an argument for not implementing a security feature or protocol due to the cost benefit ratio[20]. As well the cost of requiring formal specifications across all of the system, with regard to security, may not be appropriate for all risk areas, and, in fact, even in the initial phase of design the cost of such consideration may outweigh any benefit.

Context: During the Business Modeling nodes of all phases, the overall vision of the software product under development is considered and reviewed and revised; as this is the point where all stakeholders are communicating and the overall plan is present, an analysis of the risk factors would be appropriate here that takes into account the overall context of the system.

The actors or stakeholders involved with the application of security can also dramatically affect the decisions made involving security. We can categorize them in four main groups: the end-users, the administrators, developers, and the government. The needs of each of the groups can both complement one another and contradict at the same time and in fact this can happen with the same group. The government[18, 32], in keeping with sound political principles, will create legislation to formalize the protection consumers should expect with regard to security in software, and to create and uphold legislation designed to protect privacy and data. Both these are designed to inspire public confidence. Contrary to this, a government also holds an obligation to deal with sound economic policy, which would encourage business by having as few restrictions as possible. There are also many intermediary agencies such as ANSI (American National Standards Institute) and OSI (Open Source Initiative) that publish recommendations or guidelines for security which are left for businesses to choose to follow or not.

Context: Any legislation or documents that are relevant to the software project should be identified in the Requirements node of the Inception

phase, and then the relevant documents tagged for consideration in the Analysis and Design node. There are two relevant questions spawned by the consideration of 'What legislation is relevant to the project', that of 'What standards, such as TCP/IP are relevant to this project' and a related sub-questions 'Are these standards secure', and 'What organizations publish information regarding security?' All these were added to the Requirements node for consideration.

Administrators straddle the line between end-users and developers. They may set the policies or impose security on the end-users and then can take on the role of the end-user when they utilize products created by the developers. The end-users weigh in on the decision with the issue of usability as well as their general knowledge level. In [9] Brostoff and Sasse state "...the fact of security is at odds with convenience" concisely encapsulating the users perspective on many security features. Oftentimes these security features require a larger investment on the part of the user then they are prepared to make, the typical result being that the user does not use the security provided in a product. In this case as in all cases, "Security is only as good as the weakest link in the chain, which is people"[9].

Context: Usability plays a large role in the success of security in a software project. In the Test nodes of all phases the usability of any security features should be explored.

Chapter 6 - Practical Tool Application

6.1 Introduction

The information injected into the Security Spiral in Chapter five contains a variety of notes targeted to different phases in the software development lifecycle. This information does not, however, contain an overview or process within each workplace activity to facilitate its application to a software project. In order to create a 'process within the process' the information is applied to an existing software project named TAPAS[18]. This will aid in integrating the application of the Security Spiral into a software project and increase the usability of the tool. It is also anticipated that by examining the TAPAS project, using the information encoded in the Security Spiral, areas of concern regarding software application security in TAPAS will be exposed thereby, in part, validating the construction of the tool. The motivation for choosing the TAPAS project is that several members of the EGADSS technical group, who are working on the project currently, reside at the University of Victoria where this thesis is being undertaken, and are therefore available for support and comment.

6.2 The TAPAS project

TAPAS (Technology Assisted Practical Application Framework) is a suite of software tools undertaken to support clinical information systems dealing with electronic medical records. The system is being developed as a collection of components that will allow physicians to share patient medical records in a variety of environments such as within an office on desktop systems, as well as portable connection devices such as Personal Digital Assistants. This would provide physician's electronic access to medical records while in the office and also while them are out of the office (on-call).

The core of the TAPAS project is a "virtual practice network server" that would house patient records and summaries. The server would be shared among a network of physicians and practices to allow the interchange of patient data as needed. The server component, named CASA, provides a front end to information stored in a PostgreSQL database. Physicians then use an application known as TAPEAR for remotely accessing

[18] http://www.opentapas.org/

the medical information and displaying it locally on their desktop system. There is also work on a system, which will include software, which provides access through PDA's and other mobile devices, which is named TAPETA.

As the TAPAS project deals with personal medical information for client's, it is an area where software security can be considered to be of great consequence. Physicians require a practical means to share and exchange the patient information. They are also required to protect the confidential nature of the information and ensure that it is not exposed or accessed inappropriately. Patients need to have a high level of confidence in their right to privacy regarding their own personal medical records and, as such, the software applications that manage this information must provide security to ensure beyond all reasonable expectations that this information is protected. Given this context, TAPAS is an appropriate project for application of security in the software development process.

With regard to the actual process of software development in the TAPAS project, there are a number of issues to note. The first is that there were approximately four to five developers on the project, making TAPAS fit within the criteria of usefulness of the tool- the target being small to medium size development groups. Also, if the range of application of security to software development is considered- at one end security is ignored, and the other having security applied by experts in the field to all aspects of development, TAPAS fits approximately in the middle. The developers of TAPAS were aware of security as an important issue in their project but were not experts in the field of security. Documentation was present showing some of the issues they had considered, and effort was made to consider and incorporate security issues during development. These facts combined together with the need mentioned prior for security in the TAPAS project, due to the sensitive nature of medical information, make it a good target for application testing of the Security Spiral.

6.2.1 Tool Application

The source code of TAPAS was downloaded from the current repository and a working copy of the application acquired for the purpose of evaluation. Then, the Decision Support Tool software was started and the evaluation process begun. Upon opening, the first screen displayed is the same one as depicted in Figure 4.6. All that is

present are two visual icons labeled "RUP Security Lifecycle" and the "Security Spiral". As this point, in terms of usability, it's clear that there is no guide or clear indicator on where to proceed. Although it may seem trivial to highlight the lack of a user's instruction set, one of the earlier considerations stated in this work was that this tool will leverage existing skill sets acquired by navigating the web, to minimize the learning needed to use this tool. Web sites usually provide some type of help document or link on the main page and so this tool should not be any different. In keeping with the philosophy for adhering to open source/publicly available/multiplatform where-ever possible (so as to be able to reach the maximum audience), a decision was made to restrict documents that are embedded into the application to text files that can be expressed in text only and Adobe pdf files (for when graphics are required or needed in a document or note). The Compendium tool also supports embedding of most Microsoft document formats but restricting documentation to pdf and text will make the use of the tool universal, as it can then be used on a platform that has no knowledge of Office products in any fashion. A help file was created, named "Getting_Started" and dropped into the main view. The help document indicates that to start using the tool proceed into the Security Spiral and once there begin at the Business Modeling node on the innermost spiral. Work outwards from this point through each spiral in a clock-wise manner, clicking your way into and out of each node to make the assessment. Before navigating into the Business Modeling node, the following visual clues should be noted: each workplace activity node is preceded by a capital letter which is the first initial of the iterative phase of the development process in which it is located, so "I-Business Modeling" represents the Business Modeling workspace/place of the inception node. This name will also appear in the title bar of the workspace once we move into the node. Secondly there are two numbers located at the bottom of each of the node icons. In the case of Business modeling, there is a value of 30 on the lower left and 2 on the lower right. The 30 indicates that within the Business Modeling node of the Inception phase there are 30 more nodes in the workspace. The value two in the lower right represents what the Compendium tool refers to as *transclusion*, which means this node has two representations in the current tool (the other is on the RUP Security Lifecycle map located off of the main page). By right-clicking on the node, choosing the properties

option, and going to the View display pane, all other maps on which this node is listed will be present. In addition, if the user holds the mouse over the two, a tool-tip box will pop up giving the same listing.

Moving into the business node the background of the workspace indicates that this is point in the software development process where the vision document is created, in consultation with the customer, and an attempt to define the scope of the project is done. There are a several nodes representing security issues to be explored at this point, but it is not clear which one should be done first or where to begin. In keeping with the notion of visual navigation to facilitate use, a clear starting point is demarked with the use of the decision icon, and then lines drawn to mark the order to follow in determining issues, as shown below in Figure 6.1

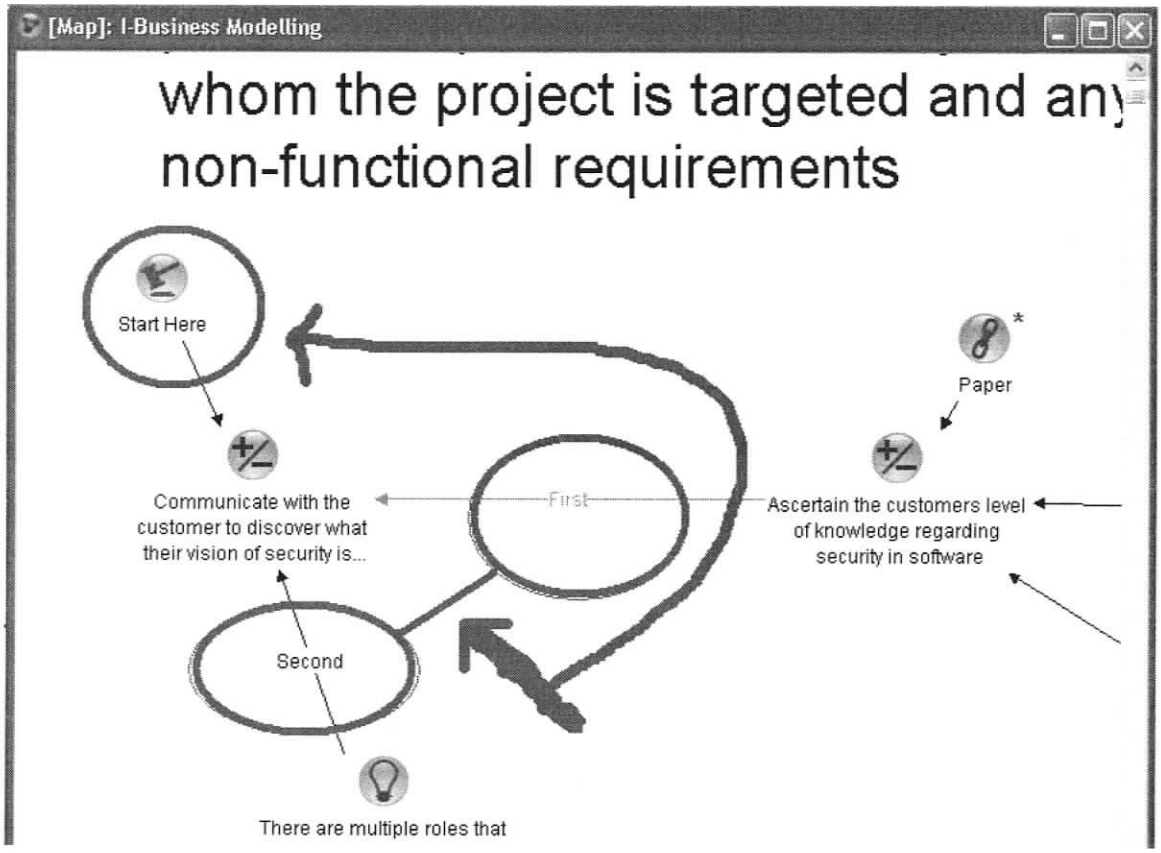


Figure 6. 1 - Adding order to issue exploration

The circles and arrows, drawn on the screen capture, show where the start icon was placed. The lines connecting issues are linked to a common root node, with labels on the line indicating the order of consideration (i.e. *first* or *second*). This same process will

need to occur for all 27 workplace nodes. A sequence needs to be laid out visually, to show how the security issues are addressed in each phase.

The next area where usability has an issue is in the presentation of the information injected from Chapter 5. Notes from this section were loaded with a question mark icon and a brief textual description describing the concern. Then option icons were placed beside the issue icon in which possible resolutions were listed. The option icon was then linked, through an arrow, to the issue. This depiction captures the piece of information revealed by the Ontology constructed, but it does not support a visually driven decision making process, or the propagation of issues or issue resolution through the rest of the tool. The visual representations of the nodes were re-designed as shown in Figure 6.2 below:

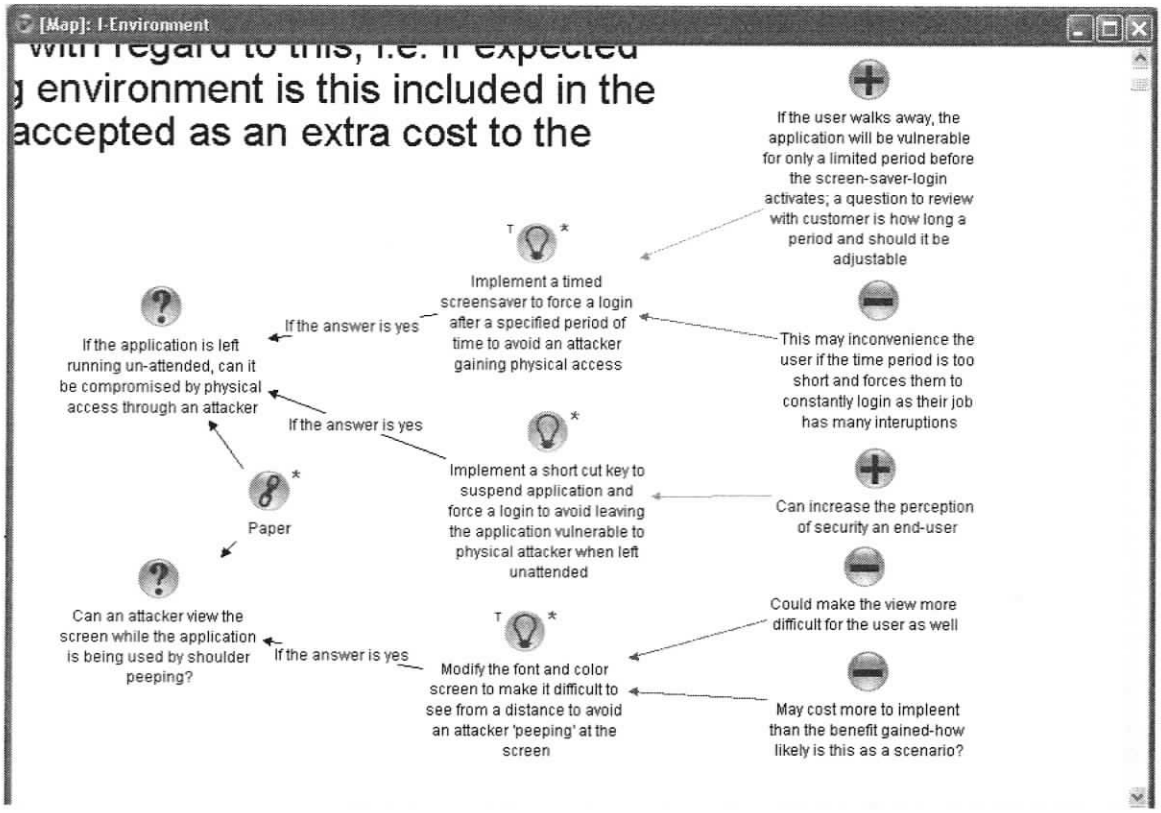


Figure 6. 2 - Issue/Decision re-organization

Originally this view was of five nodes: the two question nodes, one link node for the paper which was joined to each question and one option node (the light bulb) for each question detailing possible resolutions. The new configuration illustrated above is intended to visually walk a user through the security issue, possible resolutions and the pros and cons of the decision. It start's with the top question node, "If the application is

left running un-attended, can it be compromised by physical access through an attacker?" Immediately off this node there is a gray node with a paper click insignia and the text designation 'Paper'. This is used to indicate that there is a documented resource for reference on this problem for reading, if the user is interested or requires more information on the problem beyond what is represented. If the user holds the mouse over this link node, the details (in this case the bibliographic citation information for [12]) which includes some information on Social engineering and pertinent physical vulnerabilities, is shown. There are also two other ways to access this information in relation to the node. In the upper right corner of the icon there is an asterisk. The meaning of the asterisk, in this tool, is that it is a visual clue that there is more information in this node than what is currently on the screen. If the user moves the mouse over this asterisk, the information encoded (again, the citation information) will appear in a tool-tip box on the screen as shown in Figure 6.3.

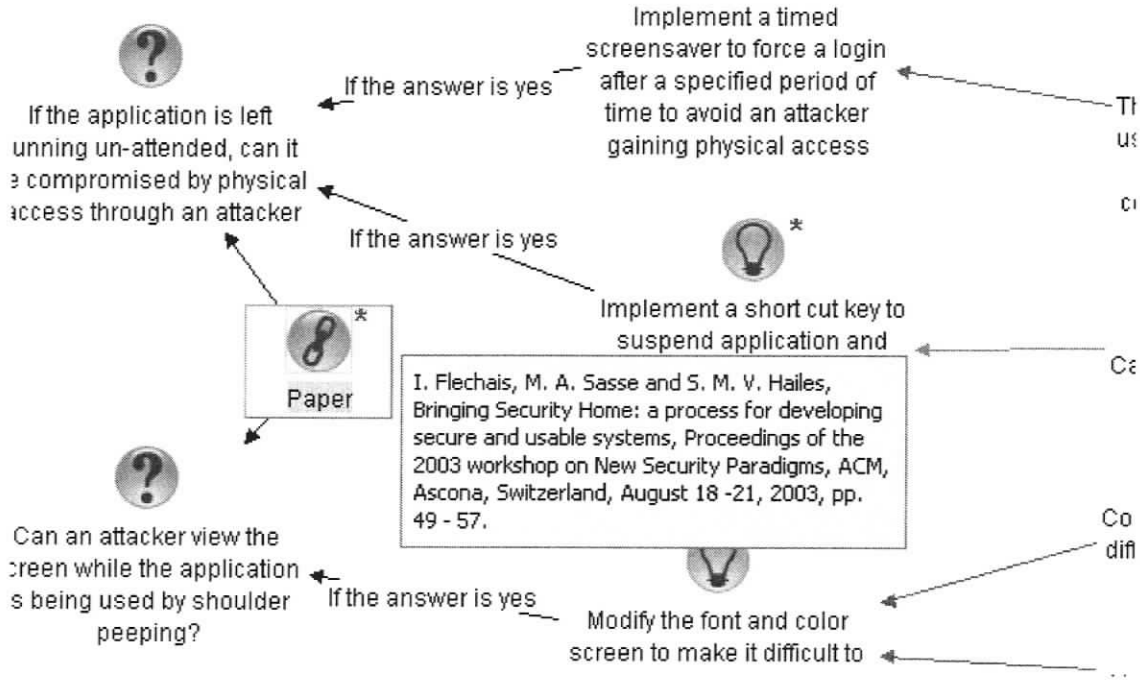


Figure 6. 3 - Tool tip for paper refernce node

As each node can have multiple pages of notes encoded, this information may not fit into a tool-tip box. The third option is to simply double click on the node, once again exercising the familiar visual navigation skills that are also common to moving around the internet, and a properties box will pop up to show the details as shown in Figure 6.4.

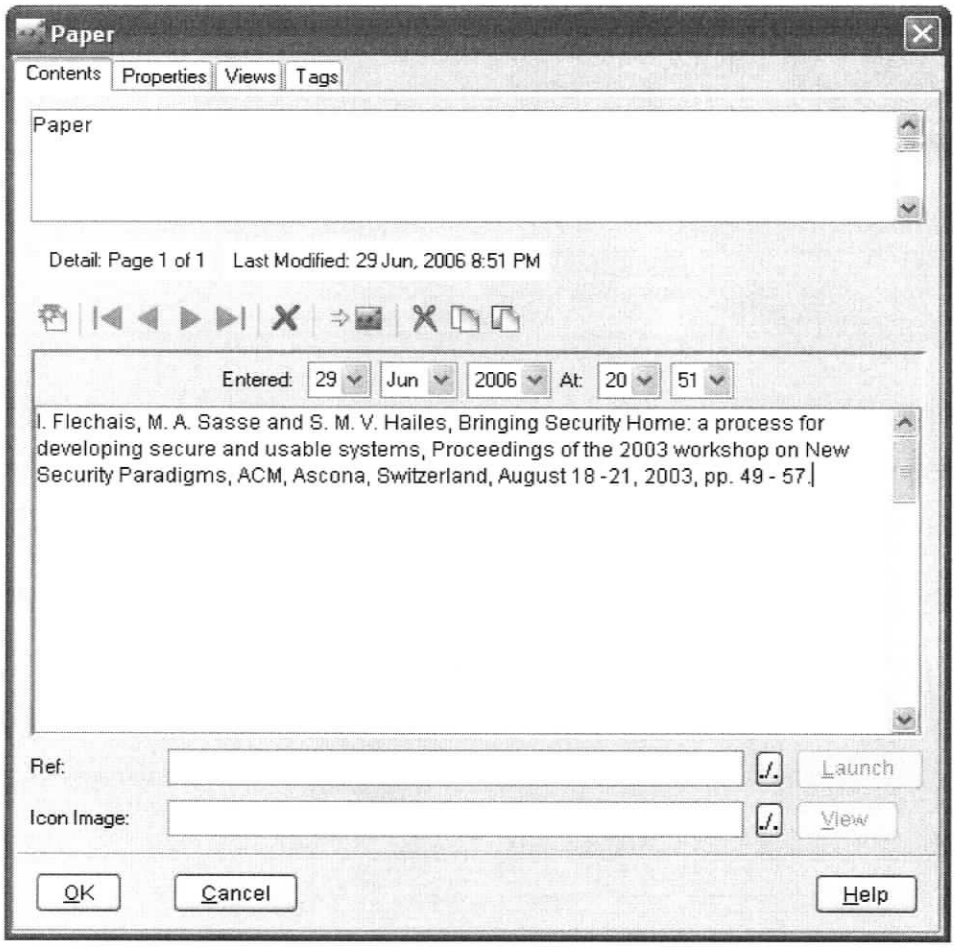


Figure 6. 4 - Properties box of a non-map node

As the figure shows, a text field exists to allow for the entry of multiple pages of information. Where more than one page of note information is present, there are arrow navigation buttons for moving forward and back through multiple pages of notes. These buttons are built into the underlying Compendium software where the Security Spiral is implemented, and provide the same behavior for the notes page as similar buttons on web browsers, MP3 players and even traditional tape cassette players. This is an un-intended plus of using the Compendium tool for the underlying implementation. There is one further option with regard to this node. In Figure 6.4 there are four panes in the window, the one being displayed is the *content* pane, and the last is the *tags* pane. In the *tags* pane there is a drop down list of 'tags' including some that describe state behavior and also actions. This allows the user to 'tag' this node with additional information such as *actionitem*, *resource* or *must_read.* The benefit of doing this is demonstrated later in the

development process but to summarize; the user can search for all nodes with a particular tag and generate a list for further action or response.

Referring back to figure 6.2 and the original question node that the paper node referenced, the relevant option issues have been further broken down- each potential response to a question has become its own node. Further alterations are that the line connecting the option to the question has been marked with "If the answer is yes". This is done to clearly mark the visual path or paths, if there are multiple responses, the user should follow if the issue identified applies to their project. In this case, there is no "If the answer is no" node. Should the user ask the question and no further response is needed in the context of their project, it is considered self-evident that they can stop traversing this path in the decision process. In other cases where there is ambiguity, or decisions and/or response(s) which vary depending on the answer, then nodes are added detailing both the affirmative and negative options. Only when it is self-evident that a yes answer or no answer eliminates the need for questioning on that particular issue is the corresponding node not included. This action, or more precisely, decision to take no further action, could still be detailed as a response in the tool. However, it would only contain one piece of information, "Do nothing more regarding this issue." Since populating the tool with dozens if not more of these "Do nothing more…" nodes would inundate the visual diagrams with a lot of 'clutter', to maintain simplicity they were not included.

The next revision involves the Option nodes from the question as shown in Figure 6.5 below.

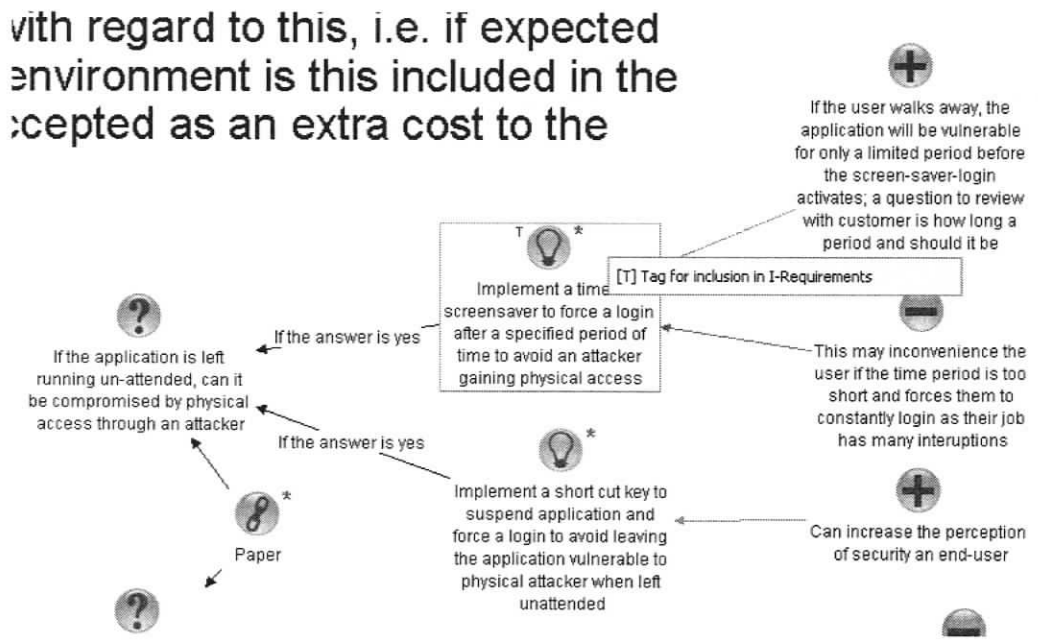


Figure 6. 5 - The option (response) node

As mentioned prior each response was broken out into its own option node. Additional detail was added so that the question, to which this option was a possible response, was included (the reasoning for this will be explained in the next paragraph), and the appropriate pros and cons of implementing this response detailed in the plus and minus nodes as illustrated above. In Chapter 4 during the tool construction phase, one of the requirements was the need to propagate information to other workplace activity nodes, which is covered next.

The depiction of the response nodes, using the option icon as shown, has an asterisk in the upper left corner of the icon and in one case ("Implement a timed…") a capital T in the upper right corner. As before, the asterisk indicates that they are additional notes that are not visible in the current view of the node, and by hovering over the asterisk the information will appear in a pop-up or alernatively; the user can double-click on a node to view the contents. In the case of the option/response node, hovering over the asterisk reveals a message in the pop-up, "(T) tag for inclusion in I-Requirements". This message indicates that, if this response is desirable in the product, it should have a tag added to it with the mentioned value. This feature will mean that when the user moves on to the Requirements activity of the Inception phase, they will have this information available to them. Assuming the user chooses to tag this node (the procedure is double-click to pull

up the properties box, select the *tag* pane and then use the drop-down listing to add the "I-Requirements" tag), the node then indicates that it has been tagged by placing a *T* in the upper right node of the icon. Holding the mouse over the *T* will produce a list of all tags that have been applied to this node. The reason for re-writing the text of the node to include the question, to which it is a response, is that when the node is tagged for inclusion into the I-requirements space, only the node tagged will be propagated. This is to forestall requiring that the question be tagged as well, and it is simpler to just include the question in the response so only one node needs to be tagged.

In order to see how to apply the result of this tagging process we now navigate to the I-Requirements node and perform the following steps. Firstly, create a list node on the I-requirements page naming it appropriately. Next, double-click on this list node which will yield a view of an empty list. We then perform a search on all nodes tagged with "I-Requirements" and choose to place them into the open list view as shown in Figure 6.6 below:

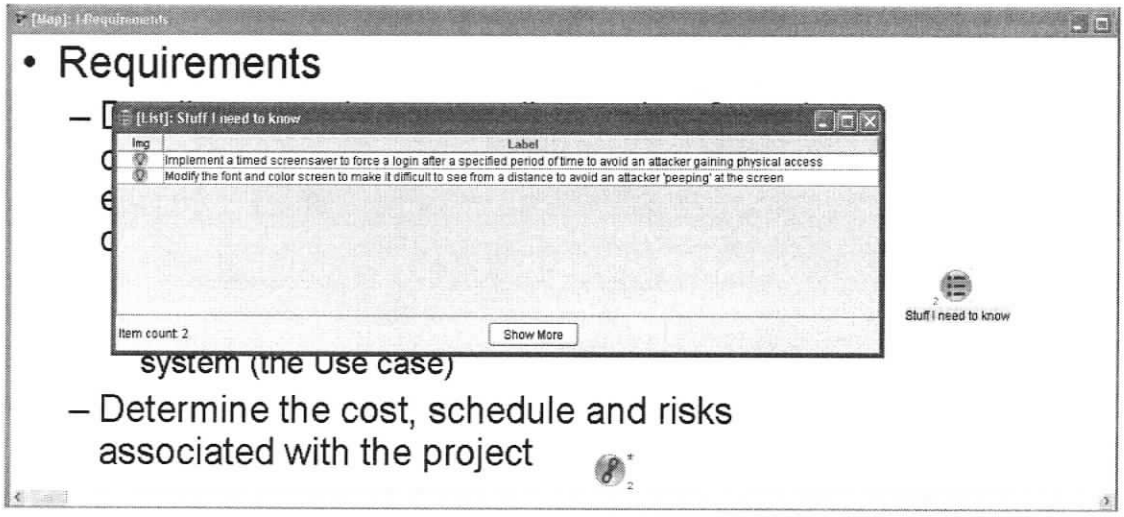


Figure 6. 6 - Generated Report Listing

The node for the list we have given the name "Stuff I need to know". Once the issue has been dealt with we can remove the tag from the node. This will not remove it from the generated list in "Stuff I need to know", but in any future list that is generated using the same process it will not appear. This allows the user to track issues that were identified but have since been resolved by comparing the contents of previous generated lists to current ones.

For each one of the 27 workplace nodes in the Security Spiral this following processes is repeated:

1. re-organizing the nodes to provide a serial workflow with a start point as well as a numbered scheme to address the issues in turn
2. refinement of question and response nodes; annotation of the decision flow (i.e. yes or no) and listing of the pros and cons of a decision
3. refinement of notes for each node to facilitate tagging as also include resource description and options

While this revision was on-going the TAPAS project was under examination. For example: when the following question node was encountered in the I-environment workplace- "If the application is left running un-attended, can it be compromised by physical access through an attacker", the available documentation was examined. From this examination there were no annotations indicating this issue had been addressed. To further confirm this, the TAPAS application was run and a patient record shown below in Figure 6.7 appeared.

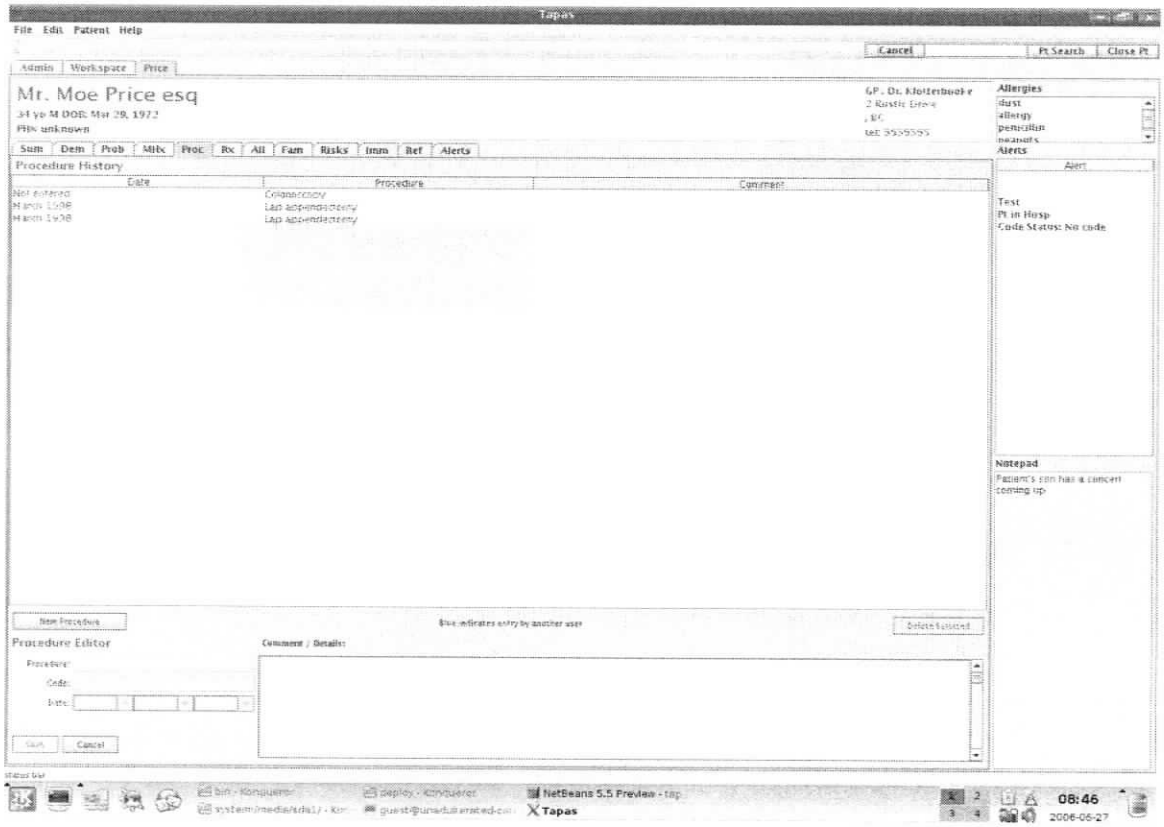


Figure 6. 7 - TAPAS patient record interface

Examining the options available there was no screensaver implemented, or shortcut screen to hide the information. This was then added to the list of questions regarding security issues for review with a TAPAS developer, to determine if this issue had been addressed, and if the developers were aware of this particular security concern. The information encoded in the rest of the nodes was applied to the TAPAS project as well during the usability revisions done while examining TAPAS, which led to the evaluation, covered in the next section.

6.2.2 Evaluation

During the application and evolution of the tool described in section 6.2.1, security concerns that were relevant to the TAPAS project were marked and identified. If the documentation and code provided for the project indicated that the issue had been addressed and satisfied then it was removed from the list. The remaining concerns were then formulated into a checklist. The purpose of this was to perform a systematic review of the identified issues with a stakeholder from the TAPAS project, who had been a contributor to its development. Such a review served two purposes. The first is to determine whether the issues identified by the tool are useful in the context of security application. The second is to provide feedback to the TAPAS project developers regarding security considerations in the context of their project. The following table summarizes both the questions asked and the responses. During the review process the questions were asked and a request made for one of three responses in reply A) Yes, the issue is applicable and Yes it was considered, B) Yes the issue is applicable and No, it had not been considered (thereby providing validation for the use of the tool) and C) No, the issue was not applicable. It was discovered during the review that these three categories were not quite sufficient, as there was additional pertinent information in the reply when it was a 'yes' on applicability but instead of a direct 'yes, the issue is applicable' the response was instead 'yes, but…'. These 'yes, but' responses were tracked and this fourth category of reply was added into the 'under consideration category'. The following table summarizes the review (*Note: The TAPAS project is currently in hiatus at the end of the Construction phase*):

QUESTION	*Applicable*	*Was it considered?*
Was an evaluation done to determine what level of expertise would be required by the end-user of the product?	Yes	Yes, initially considered but the customer had not determined what authentication would be required. It then 'fell through the cracks' meaning it was not marked to be followed-up on.
Assuming an attacker gains physical access to the machine, can the application be compromised?	Yes	Yes, but it was assumed that underlying choices for implementation languages and database would take care of this. Situations in which an attacker pops the hard drive were not considered.
Was an adversary model created to investigate compromise, risk and liability?	Yes	No
Will extra data backups be required to ensure integrity in the case of a malicious internal user?	Yes	Yes, but in the sense of a general failure of the primary database, not in the specific instant of malicious alternation of the information.
Was a liability document created to inform the customer what liability was theirs and what was the developer's responsibility?	Yes	No
Was applicable legislation explored and conformance to it prescribed in the development?	Yes	No, there was an awareness of relevant information re: medical records and access but not follow-up as of yet to ensure compliance
Were code sections involving security testing for performance vs. usability tradeoffs?	Yes	No
Was there a security document clarifying the customers understanding and expectations with regard to security?	Yes	No
Was a search performed for the known vulnerabilities of the selected platform/languages of the project?	Yes	No
Was a list of attack signatures for known vulnerabilities constructed?	Yes	No
Was the economy of mechanism issue applied to all code and design	Yes	Yes

dealing with security issues?		
If the application was left running un-attended could it be compromised by physical access of an attacker?	Yes	Yes, initially considered but then 'it fell through the cracks' – not marked for follow-up or response
Is the application vulnerable to 'shoulder-peeping' by an attacker?	Yes	No
Will material printed out by the application be vulnerable to the dumpster diving attack?	Yes	Yes, initially considered but then 'it fell through the cracks' – not marked for follow-up or response
Can printed material from the application be left in an internal area where it could be compromised?	Yes	Yes, initially considered but then 'it fell through the cracks' – not marked for follow-up or response
Can the end-user introduce an unsecured access point to the system?	Yes	Yes
Were the security implications of the chosen language reviewed?	Yes	Yes-it was assumed the Java was secure
Is there a mechanism in place to track changes and modifications to the project that affect security?	Yes	No, it was somewhat in place as the project was hosted on source forge and communication done through a mailing list, but some emails were not sent to the list concerning security and were lost and on other occasions emails containing sensitive information such as server passwords were posted to the list and publicly viewable
Was a risk assessment performed of the hardware architecture?	Yes	No, not the server or desktops. The security was considered for the mobile platforms however.
For each identified vulnerability, was a risk assessment performed?	Yes	No
Was their an assessment of whether the application is vulnerable to the 'message replay' attack?	Yes	No
Was their an assessment of whether the application is vulnerable to a Denial of Service attack?	Yes	Yes, but what response to take was deferred and has since 'fallen through the cracks'
Was 'Un-helpfullness' as a tool to resolve security issues considered?	Yes	No
Is all data access completely mediated?	Yes	Yes
Were all 'fail-safe' code sections	Yes	No

reviewed for 'fail-secure' also?		
Were static analysis tools such as Security Check considered and researched for potential use in the project?	Yes	No
Is there an audit trail or logging mechanism in place?	Yes	Yes
Will communication between this application and all others be monitored to detect security issues?	Yes	Yes-through the use of the logging mechanism just discussed
Were visual clues to build confidence in security presence considered for the user interface?	Yes	No
Were all security related code sections analyzed for vulnerability to the protocol timing attack?	Yes	No
Were all algorithms dealing with Cryptography publicly reviewed to expose flaws?	Yes	Yes-used public libraries
Were the common questions asked by end-users considered and mechanism in place to respond?	Yes	No

Table 7. 1-Systematic review of potential identified security issues in TAPAS

The table below lists the summary of the responses. There were 9 questions asked that do not appear in Table 7.1 as the response was that the identified consideration was not applicable, so the presence of the question can neither validate or invalidate the purpose of the systematic review. The 'Yes- but' category was added as there was a significant number of questions that fell into that category of the 32 questions asked, and the feedback gained during the response bears relevance.

Response	***Number of Responses***
Yes it applies and Yes it was considered	**6**
Yes it applies and No it was not considered	**18**
Yes it applies and Yes it was considered but…	**8**

Table 7. 2 - Results of the Systematic Review

The first category of response is the easiest to evaluate in terms of the tool. Six issues had been identified for consideration and in all six cases they had been considered and

resolved, and hence the value of tool in these could act only as a review of what had already been accomplished.

The second category portrays the most significant results. Of the 32 applicable issues that were discussed more than half were considered relevant to the TAPAS project and had not been identified or addressed. This would lend value to justification for the tool in that it has successfully identified security issues in the TAPAS development process that are areas of opportunity. It is important to note that this result is in no way a comment on the work done for TAPAS, and in fact security issues in general were considered from the outset of the project and included in the original documents of the Business Model and followed through in implementation. This work has highlighted additional areas were concerns may exist and, assuming they are then resolved, can only serve to increase the security of the TAPAS software project.

The third category of "Yes it applies and yes it was considered but..." generated perhaps the most interesting commentary of the review process. First and foremost there was the issue of "it was considered but has since fallen through the cracks". A comment was made that a tool that easily facilitated tracking of these issues throughout the development lifecycle would have prevented any possible 'slippage' in terms of issues that were lost during the development process. The implementation of the Security Spiral with its tagging and report features would provide such a venue. The second insight was that in some cases situations were identified were security had been dealt with by accident, for example in the case of the "data backup to in preserve integrity to prevent attack by malicious internal user" the data backup was being performed as part of a general assumption that backing up was a good policy to avoid hardware failure, loss of data, and so on. The context of a malicious internal user had not been considered in this case and, in fact, when this was included as part of the question it spawned further consideration of exactly what a malicious internal user might due to compromise integrity, and how and when this could be detected and corrected. As well it forced the consideration of the following question: "what could the result be of a successful compromise in this scenario?" So though the response to the question was answered "Yes it was applicable and yes it was considered/dealt with" it had value in that it generated insight into related security issues that may not have been considered.

Overall this systematic review of security concerns and issues identified during the application of the tool to TAPAS has lent credence to the value of the tool in raising awareness of security issues at least in the case of where it was applied to TAPAS.

Chapter 7 - Conclusions

7.1 Summary

Applying security in the software development process has quickly become a highly desirable characteristic as evidenced by the every-growing media presence of articles decrying the cost and exposure generated by security vulnerabilities. Most small to moderate size development teams lack an expert in the domain of security that can bring years of practical experience to a project to ensure that appropriate security is incorporated into a project. Furthermore these teams cannot halt their development projects in order to gain the education and experience required. A methodology or tool for injecting security into the software process is needed that can identify security concerns during development and assist with the decision support needed in order to determine the appropriate response. In addition this tool will need to provide insight into security issues, integrate into whatever software process model is being used for project development, and endeavor to minimize the learning needed in order to apply this tool.

This objective of this thesis was to develop an approach to facilitate injection of security issues into software development. This approach should map to existing development processes and provide a framework for identifying security concerns as well as provide resources for resolving these issues. The presentation and use of the tool should be enacted to minimize the learning required for its use and should presume only the presence of skills common to software developers and not specific to security issues. Additionally the approach should be open source, extensible and multiplatform to allow for its application in the widest possible venue.

This is realized by the creation of the Security Spiral as a framework for anchoring security issues in the process. Through combination of existing process models, the Security Spiral provides a visually driven paradigm for ease of mapping into other process models. Through leveraging the existing familiarity that developers have with navigating the internet in the presentation and use of the tool, developers are able to focus their time on the issues identified by the tool rather than learning how to navigate the tool. Using an open source Java based software application for tool construction achieves

the goals of extensibility, supporting a multiplatform operating environment, and allowing essentially any interested party to utilize the tool.

There is at least one limitation to the current framework that requires consideration. It revolves around the potential scalability of the tool. Given the breadth of material encompassed by the field of Security, if all the possible facts were loaded into the tool, it is possible that the large volume of material would exhaust the resources of the underlying platform on which the tool is running. In other words, the tool would not be able to execute if it was fully loaded due to resource limitations. Another issue involving scalability comes into consideration when considering the issue of usability from a user's perspective. As the Security Spiral relies on visual mapping of the knowledge, there is a direct correlation between an increase in the amount of information in the tool, and the visual 'real estate' that must be navigated in order to use the tool. This increase may result in a decrease in the usability of the tool, specifically in navigation and comprehension. In both these cases of scalability being an issue, if the focus of the tool, and therefore the information encoded, was narrowed to a more specific problem domain, the problems of scalability may be overcome, with the tradeoff being that some information that may have been pertinent to the case is not present in the tool during the analysis.

7.2 Contributions

There are four contributions made in this work:

1. Construction of the Security Spiral as a means of creating a visual map that can overlay with other software process models, and be used to anchor security issues in the software development lifecycle.
2. Implementing the workflow of the tool to imitate internet navigation and using icons to visually drive the decision support process which leverages the existing skill set of the developers and minimizes the time required to learn to use the tool.
3. Organizing and gathering security information and injecting it into a software process.

4. The creation of a workflow through the Security Spiral that parallels software development to facilitate a systematic process in using the tool and increase usability.

In comparison to the AEGIS project or Microsoft's SDL platform, the Security Spiral tool offers the following advantages:

- It does not rely on the presence of a security domain expert
- Implemented in open source software it is not vulnerable to proprietary considerations, and can be used by anyone whom chooses too.

The other related works all deal with reactive assessment of existing software projects, whereas the Security Spiral is intended to deal with issues during development to forestall their presence after the project has been deployed. This is a more cost effective solution to dealing with security issues and concerns in software applications.

7.3 Future Work

The following are possible areas for future enhancements or research that can be explored using the Security Spiral

- Currently security issues are stored internally in the built in Derby database. Compendium allows for the use of an external MySQL database to both store the associated project map and resources to provide for concurrent use. This storage uses an xml format for the data. As Compendium is open source and written in Java ,a way to create a document report interface or view similar to the one used by RUP can be explored as an expansion to the current tool.
- The Security Spiral can be improved through successive case studies involving both short term and long term software projects to increase both its usability and the information encoded in the tool

-

Bibliography

[1] *Improving security across the software development lifecycle, Task force Report*, National Cyber Security Partnership, April 1, 2004.

[2] M. D. Abrams, *Security engineering in an evolutionary acquisition environment, Proceedings of the 1998 workshop on New security paradigms*, ACM, Carlottesville, Virginia, USA, September 22- 26, 1998, pp. 11 - 20.

[3] S. Andersson, A. Clark, G. Mohay, B. Schatz and J. Zimmermann, *A Framework for Detecting Network-based Code Injection Attacks Targeting Windows and UNIX, Computer Security Applications Conference, 21st Annual*, IEEE, Tucson, AZ, USA, December 5-9, 2005.

[4] D. E. Bell and L. Lapadula, *Secure Computer System: Unified Exposition and Multics Interpretation, ESD-TR-75-306, ESD/AFSC*, Hanscom AFB, Bedford, MA, USA, 1975.

[5] K. Beznosov and P. Kruchten, *Towards agile security assurance, Proceedings of the 2004 workshop on new security paradigms*, ACM, Nova Scotia, Canada, September 20 - 23, 2004, pp. 47 - 54.

[6] M. Bishop, *Computer Security History Project Home Page*, Electronic Article, http://seclab.cs.ucdavis.edu/projects/history/, *Accessed* June 16, 2006

[7] M. Bishop, *Computer Security: Art and Science*, Addison-Wesley, Boston, MA, USA, 2003.

[8] B. Blakley, E. McDermott and D. Geer, *Information security is information risk management, Proceedings of the 2001 workshop on New security paradigms*, ACM, Cloudcroft, New Mexico, September 10 - 13, 2001, pp. 97 - 104.

[9] S. Brostoff and M. A. Sasse, *Safe and sound: a safety-critical approach to security, Proceedings of the 2001 workshop on New security paradigms*, ACM, Cloudcroft, New Mexico, September 10 - 13, 2001, pp. 41 - 50.

[10] G. Cray, *UNIVAC I: The First Mass-Produced Computer*, Electronic Article, 5, http://www-static.cc.gatech.edu/gvu/people/randy.carpenter/folklore/v5n1.html, *Accessed* June 14, 2006

[11] B. Dragovic and J. Crowcoft, *Information exposure control through data manipulation for ubiquitous computing, Proceedings of the 2004 workshop on New security paradigms*, Nova Scotia, Canada, September 20 - 23, 2004, pp. 57 - 64.

[12] I. Flechais, M. A. Sasse and S. M. V. Hailes, *Bringing Security Home: a process for developing secure and usable systems, Proceedings of the 2003 workshop on New Security Paradigms*, ACM, Ascona, Switzerland, August 18 -21, 2003, pp. 49 - 57.

[13] J. T. Giffin, M. Christodorescu and L. Kruger, *Strengthening Software Self-Checksumming via Self-Modifying Code, Computer Security Applications Conference, 21st Annual*, IEEE, Tucson, AZ, USA, December 5-9, 2005.

[14] D. Green and A. DiCaterino, *A Survey of System Developement Process Models, CTG.MFA-003*, Center for Technology and Government, University of Alabany, USA, February 1998.

[15] B. H. Hansen and K. Kautz, *Grounded Theory Applied - Studying Information Systems Developement Methodologies in Practice*, *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, IEEE, Hawaii, USA, Jan. 3 - 6, 2005, pp. 264b-264b.

[16] H. M. Hinton, *Under-specification, composition and emergent properties*, *Proceedings of the 1997 workshop on New security paradigms*, ACM, Langdale, Cumbria, United Kingdom, September 23 - 26, 1997, pp. 83 - 93.

[17] D. M. Howe, *Information system security engineering: a spiral approach to evolution*, *Proceedings of the 1992 - 1993 workshop on New security paradigms*, IEEE, Little Compton, Rhode Islane, USA 1993, pp. 53 - 56.

[18] C. Irvine and T. Levin, *Quality of security service*, *Proceedings of the 2000 workshop on New security paradigms*, ACM, Ballycotton, County Cork, Ireland, September 18 - 21, 2000, pp. 91 - 99.

[19] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[20] D. M. Kienzle and W. A. Wulf, *A practical approach to security assessment*, *Proceedings of the 1997 workshop on New security paradigms*, ACM, Langdale, Cumbria, United Kingdom, September 23 - 26, 1997, pp. 1 - 4.

[21] P. Kroll and P. Kruchten, *The Rational Unified Process made easy - A practitioner's guide to the RUP*, Addison-Wesley, Boston, MA, USA, 2003.

[22] S. Lipner, *The trustworthy computing security development lifecycle*, *Computer Security Applications Conference, 20th Annual*, IEEE, Tucson, AZ, USA, December 6-10, 2004.

[23] L. A. Maciaszek and B. L. Liong, *Practical Software Engineering: A Case Study Approach*, Addision-Wesley, Edinburg Gate, Harlow, Essex, England, 2005.

[24] T. Markham, D. Colby and M. Denz, *Security Modelling in the COTS environment*, *Proceedings of the 1999 workshop on New security paradigms*, ACM, Caledon Hills, Ontario, Canada, September 24 - 29, 1999, pp. 96 - 102.

[25] Y. Mashevsky, *Malware Evolution: 2005*, Electronic Article, Kapersky Labs, http://www.viruslist.com/en/analysis?pubid=178949694, *Accessed* July 5, 2006

[26] J. McDermott, A. Kim and J. Froscher, *Merging paradigms of survivability and security: stochastic faults and designed faults*, *Proceedings of the 2003 workshop on New security paradigms*, Ascona, Switzerland, August 18 - 21, 2003, pp. 19 - 25.

[27] R. Nelson, *Unhelpfulness as a security policy: or it's about time*, *Proceedings of the 1995 workshop on New security paradigms*, IEEE, La Jolia, California, USA, August 22 - 25, 1995, pp. 29 - 32.

[28] D. A. Norman, *The Design of Everyday Things*, Basic Books, Sept 17., 2002 (reprint edition).

[29] D. J. Power, *Decision support systems : concepts and resources for managers*, Quorum Books Westport, Conn., USA, 2002.

[30] R. S. Pressman, *Software Engineering - A Practitioner's Approach*, McGraw-Hill, New York, NY, USA, 2005.

[31] A. Ray, *Security check: a formal yet practical framework for secure software architecture*, *Proceedings of the 2003 workshop on New security paradigms*, ACM, Ascona, Switzerland, August 18 - 21, 2003, pp. 59 - 65.

[32] C. Salter, O. S. Saydjari, B. Schneier and J. Wallner, *Toward a secure system engineering methodology*, *Proceedings of the 1998 workshop on New security paradigms*, ACM, Carlottesville, Virginia, USA, September 22 - 26, 1998, pp. 2 - 10.

[33] G. Schudel and B. Wood, *Adversary work factor as a metric for information assurance*, *Proceedings of the 2000 workshop on New security paradigms*, ACM, Ballycotton, County Cork, Ireland, September 18 - 21, 2000, pp. 23 - 30.

[34] B. Schwarz, H. Chen, D. Wagner, J. Lin, W. Tu, G. Morrison and J. West, *Model Checking An Entire Linux Distribution for Security Violations*, *Computer Security Applications Conference, 21st Annual*, IEEE, Tucson, AZ, USA, December 5-9, 2005.

[35] R. Shirey, *Internet Security Glossary, RFC 2828*, Electronic Article, http://www.ietf.org/rfc/rfc2828.txt, *Accessed* June 16, 2006

[36] D. K. Smetters and R. E. Grinter, *Moving from the design of usuable security technologies to the design of useful secure applications*, *Proceedings of the 2002 workshop on New security paradigms*, ACM, Virginia Beach, Virginia, USA, September 23 - 26, 2002, pp. 82 - 89.

[37] S. J. Templeton and K. Levitt, *A requires/provides model for computer attacks*, *Proceedings of the 2000 workshop on New security paradigms*, ACM, Ballycotton, County Cork, Ireland, September 18 - 21, 2000, pp. 31 - 38.

[38] B. Verduyn, *2005 FBI Computer Crime Survey*, 2005.

[39] H. V. Vliet, *Software Engineering: Principles and Practice*, John Wiley and Sons, Ltd., New York, NY, USA, 2000.

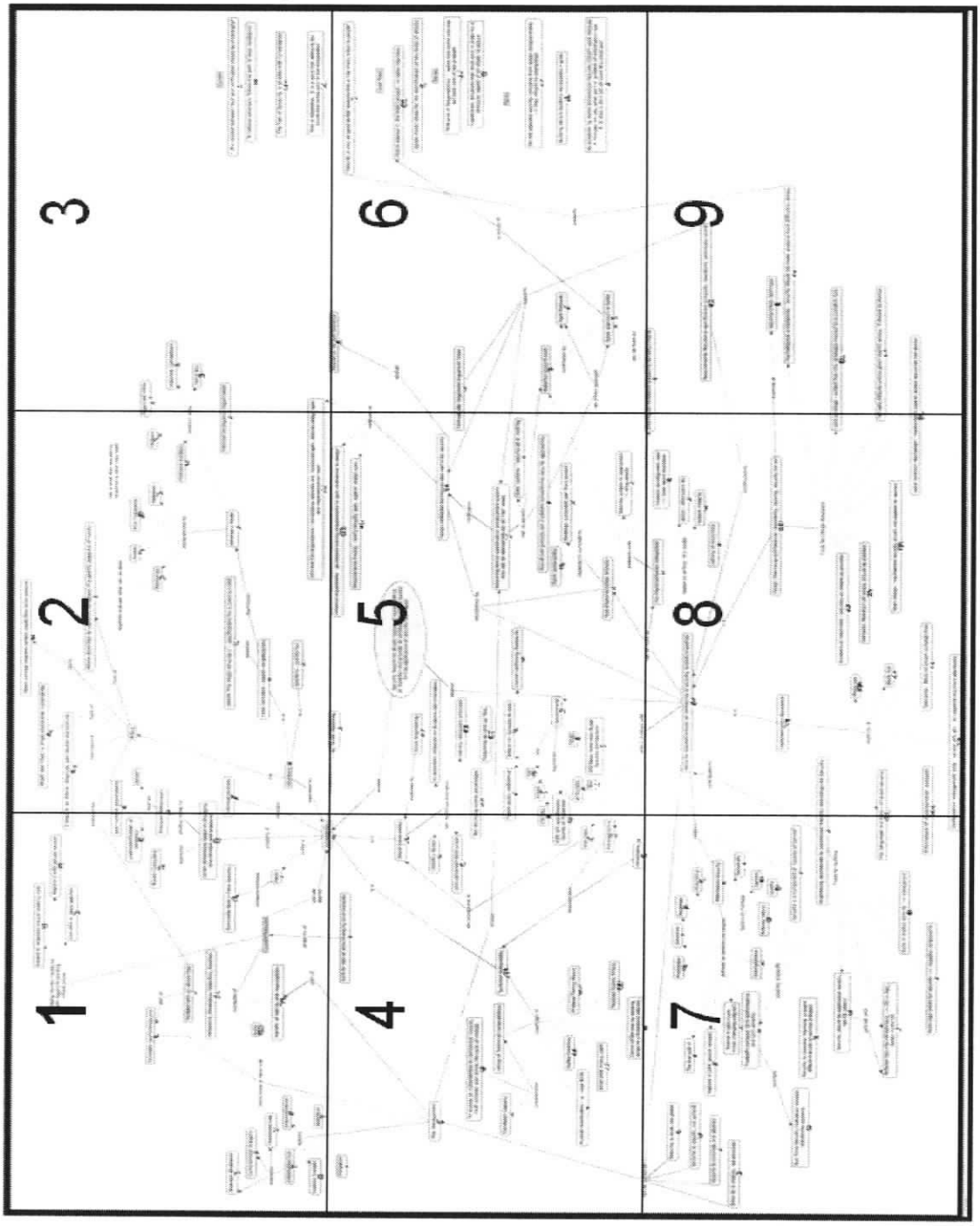
[40] W. Ware, *Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security Rand Report R609-1*, February, 1970.

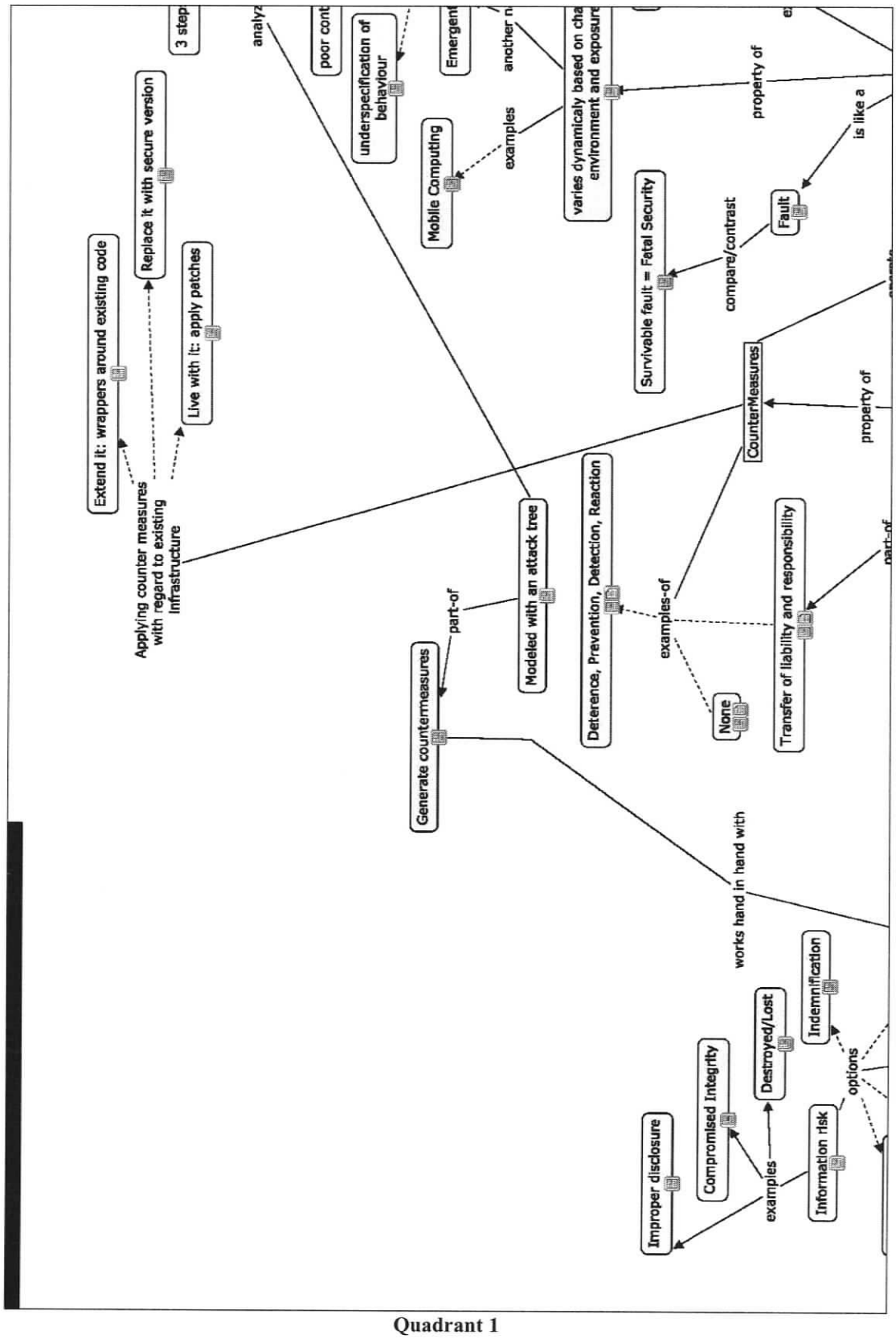
[41] Y. Wu and R. H. C. Yap, *A User-level Framework for Auditing and Monitoring*, *IComputer Security Applications Conference, 21st Annual*, IEEE, Tucson, AZ, USA, December 5-9, 2005.

[42] M. E. Zurko, *User-Centered Security: Stepping up to the Grand Challenge*, *Computer Security Applications Conference, 21st Annual*, IEEE, Tucson, AZ, USA, December 5-9, 2005.

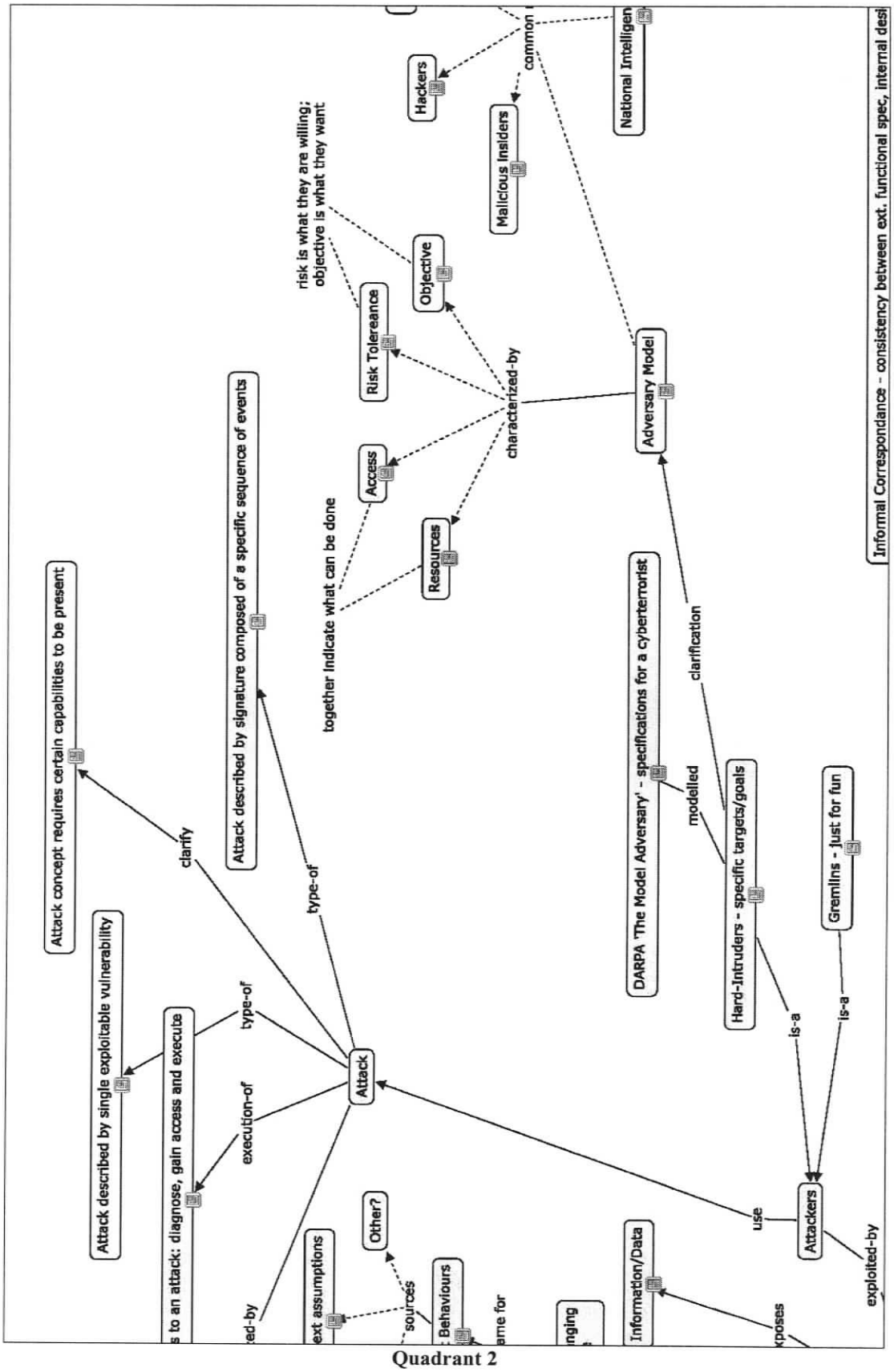
Appendix A - Ontology map for Theoretical Analysis

The Ontology map depicted below shows the 9 quadrants illustrated on the following pages and their associated position and number:

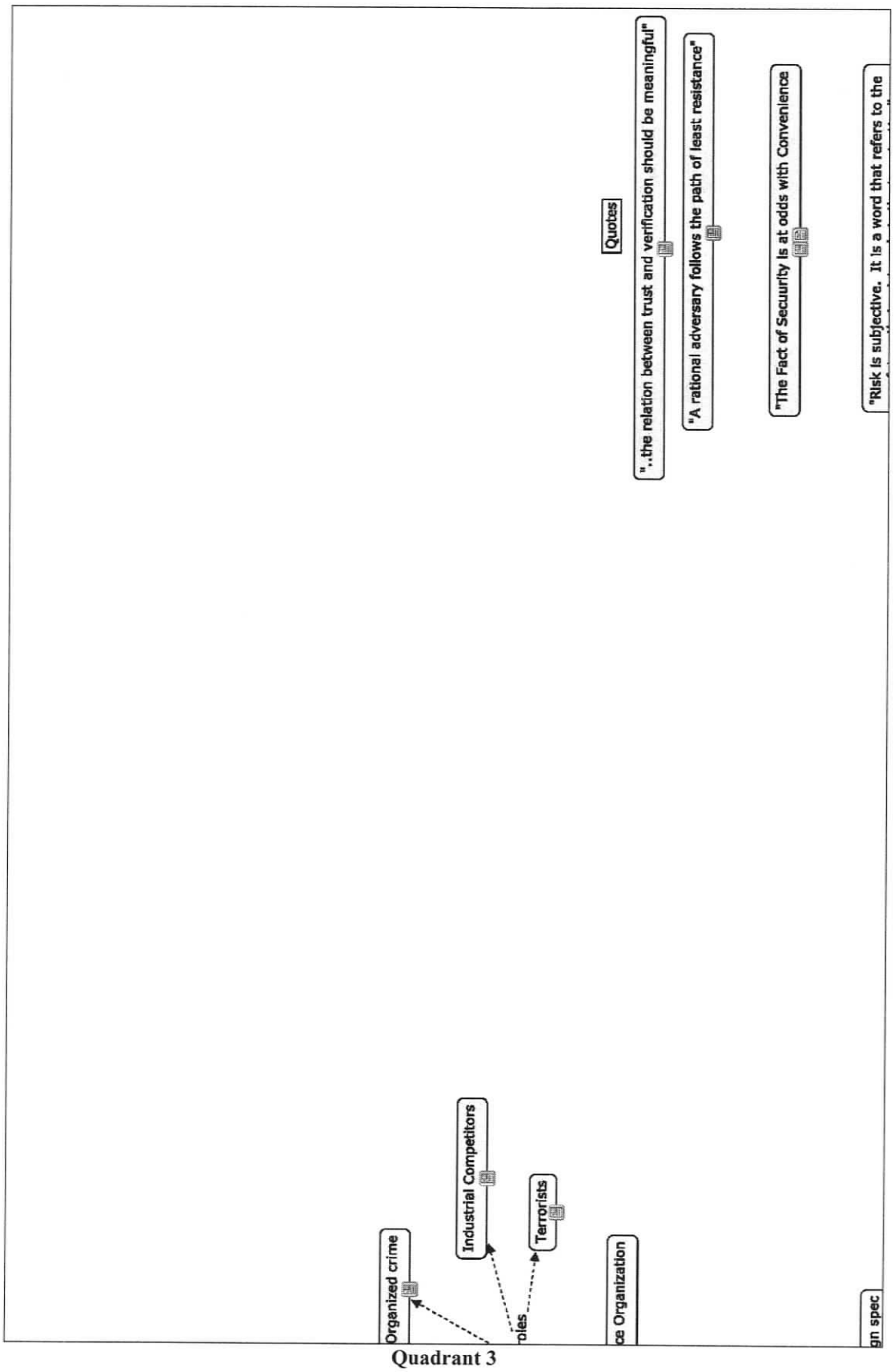



Extend it: wrappers around existing code
Replace it with secure version
Live with it: apply patches
Applying counter measures with regard to existing Infrastructure
3 step
analyz
poor cont
underspecification of behaviour
Emergent
another n
Mobile Computing
examples
varies dynamicaly based on cha environment and exposure
property of
is like a
Survivable fault = Fatal Security
compare/contrast
Fault
CounterMeasures
property of
Generate countermeasures
part-of
Modeled with an attack tree
Deterence, Prevention, Detection, Reaction
examples-of
None
Transfer of liability and responsibility
part-of
works hand in hand with
Improper disclosure
Compromised Integrity
Destroyed/Lost
Indemnification
examples
Information risk
options

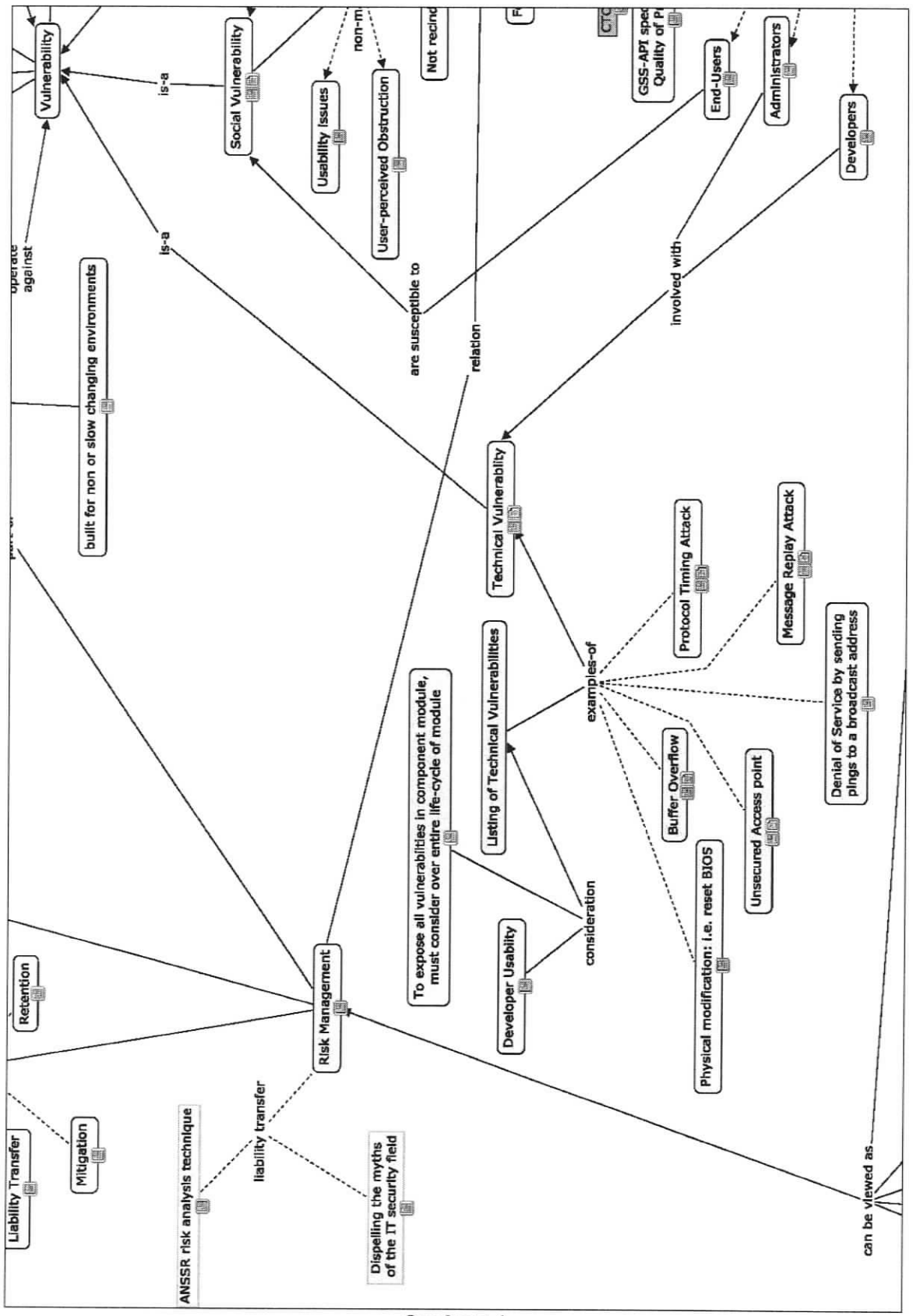
Quadrant 1


Informal Correspondance - consistency between ext. functional spec, internal desi
Attack concept requires certain capabilities to be present
Attack described by single exploitable vulnerability
Attack described by signature composed of a specific sequence of events
Attack
clarify
type-of
type-of
execution-of
s to an attack: diagnose, gain access and execute
ed-by
ext assumptions
Other?
sources
Behaviours
ame for
nging
Information/Data
use
Attackers
exploited-by
kposes
is-a
is-a
Hard-Intruders - specific targets/goals
Gremlins - just for fun
modelled
DARPA 'The Model Adversary' - specifications for a cyberterrorist
clarification
Adversary Model
characterized-by
Risk Tolereance
Objective
Access
Resources
together Indicate what can be done
risk is what they are willing;
objective is what they want
Hackers
Malicious Insiders
National Intelligen
common

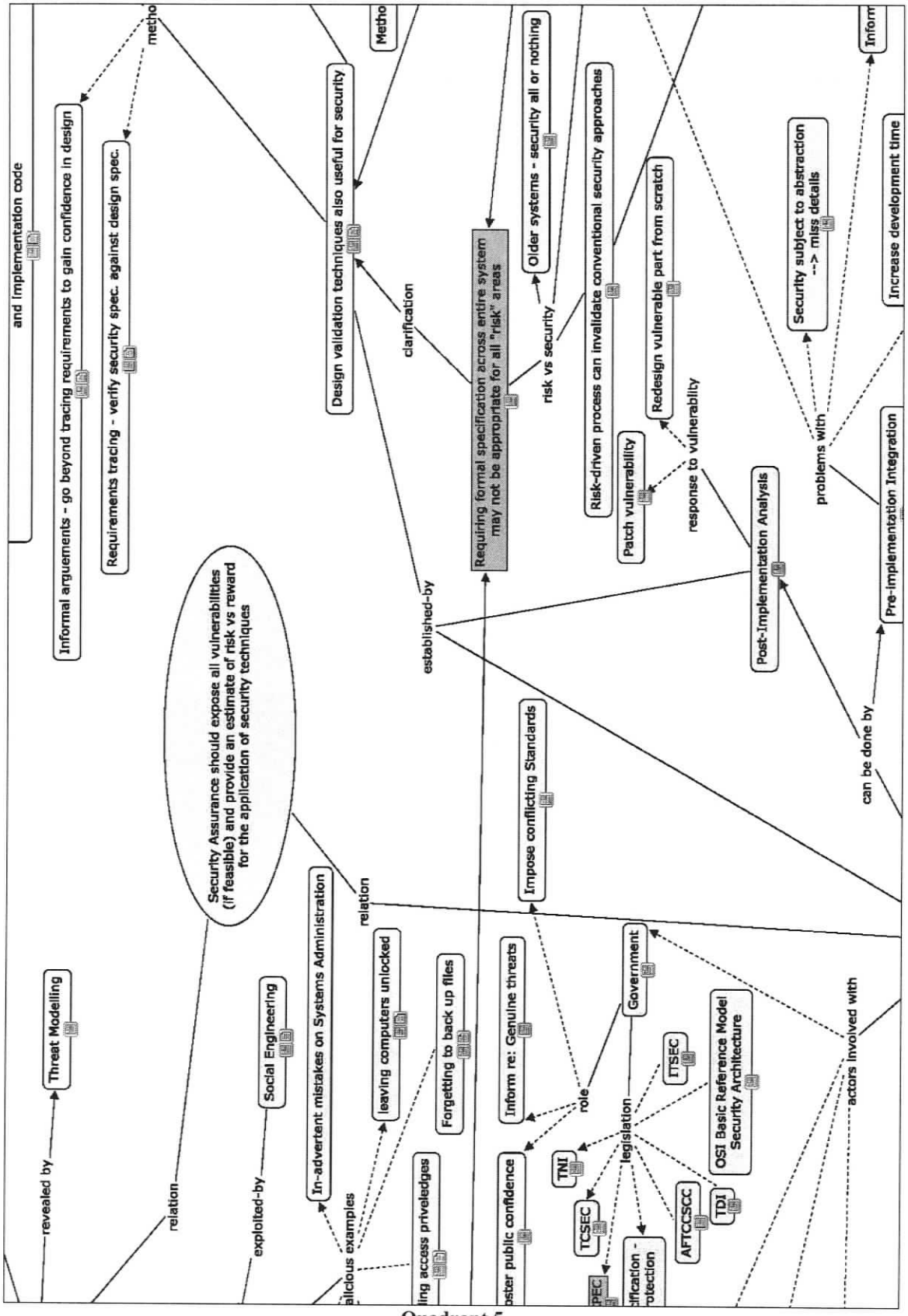
Quadrant 2


Quotes
"..the relation between trust and verification should be meaningful"
"A rational adversary follows the path of least resistance"
"The Fact of Secuurity Is at odds with Convenience
"Risk is subjective. It is a word that refers to the
Organized crime
Industrial Competitors
Terrorists
ce Organization
gn spec

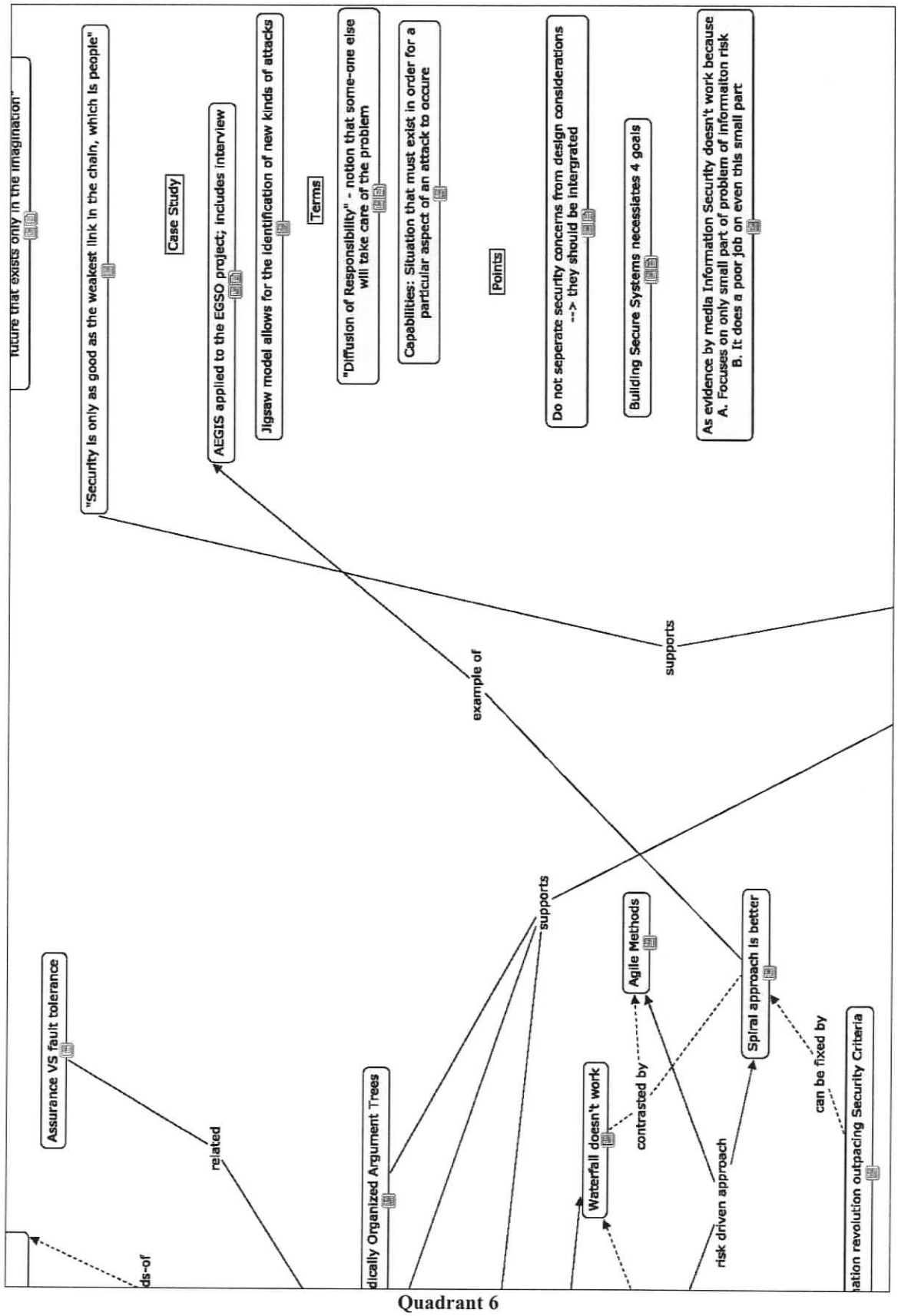
Quadrant 3


Vulnerability
is-a
Social Vulnerability
Usability Issues
User-perceived Obstruction
Not recind
GSS-APT spec
Quality of P
End-Users
Administrators
Developers
involved with
are susceptible to
relation
operate against
built for non or slow changing environments
Technical Vulnerability
examples-of
Protocol Timing Attack
Message Replay Attack
Denial of Service by sending pings to a broadcast address
Buffer Overflow
Unsecured Access point
Physical modification: i.e. reset BIOS
Listing of Technical Vulnerabilities
To expose all vulnerabilities in component module, must consider over entire life-cycle of module
consideration
Developer Usability
Risk Management
Retention
Liability Transfer
Mitigation
ANSSR risk analysis technique
liability transfer
Dispelling the myths of the IT security field
can be viewed as

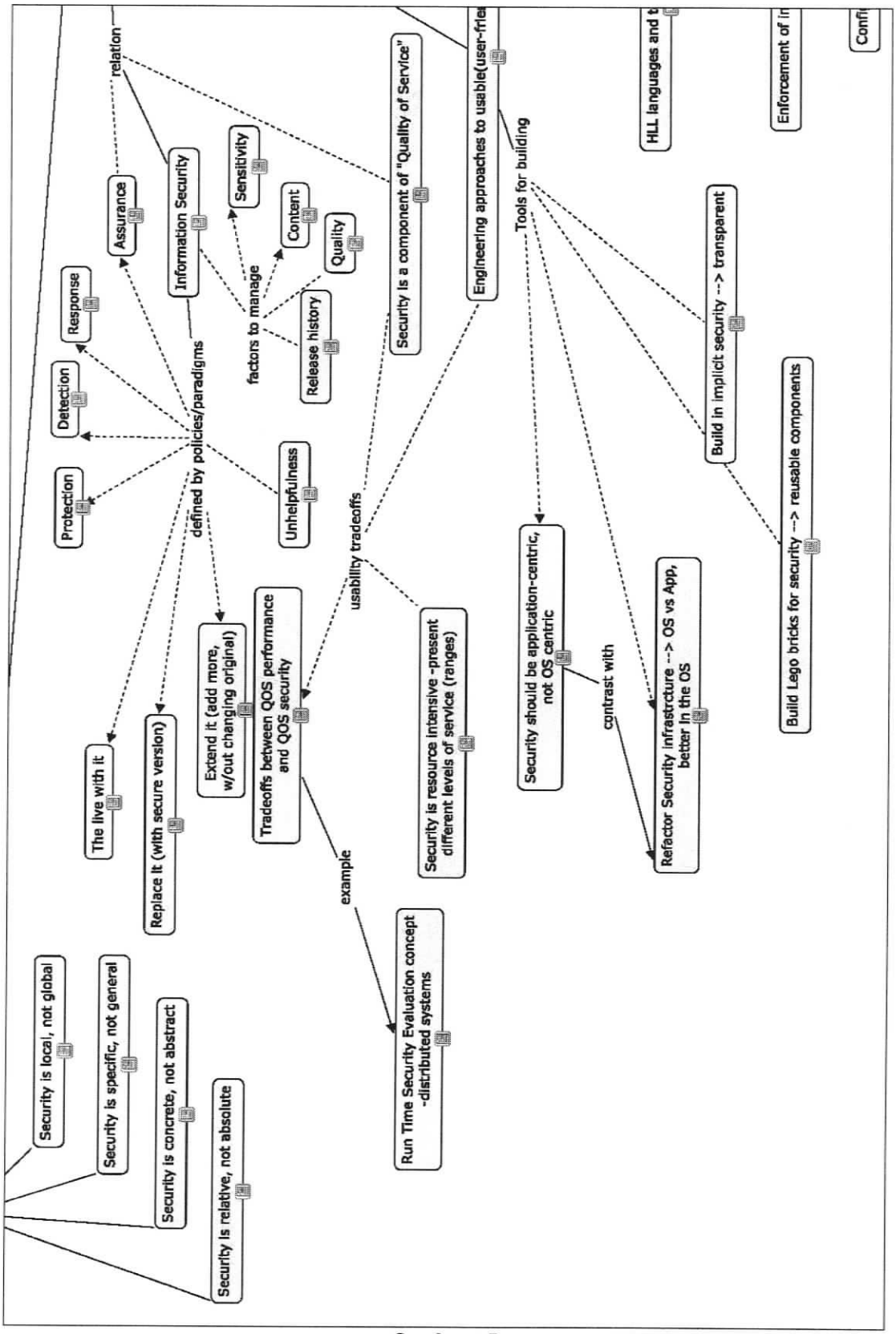
Quadrant 4


Security Assurance should expose all vulnerabilities (if feasible) and provide an estimate of risk vs reward for the application of security techniques
and Implementation code
Informal arguements - go beyond tracing requirements to gain confidence in design
Requirements tracing - verify security spec. against design spec.
Design validation techniques also useful for security
clarification
established-by
Requiring formal specification across entire system may not be appropriate for all "risk" areas
Older systems - security all or nothing
risk vs security
Risk-driven process can invalidate conventional security approaches
Patch vulnerability
Redesign vulnerable part from scratch
response to vulnerability
Post-Implementation Analysis
problems with
Security subject to abstraction --> miss details
Increase development time
Pre-implementation Integration
can be done by
revealed by
Threat Modelling
relation
exploited-by
Social Engineering
In-advertent mistakes on Systems Administration
leaving computers unlocked
Forgetting to back up files
relation
Impose conflicting Standards
Inform re: Genuine threats
role
Government
legislation
ITSEC
OSI Basic Reference Model Security Architecture
TDI
AFTCCSCC
TCSEC
TNI
actors involved with

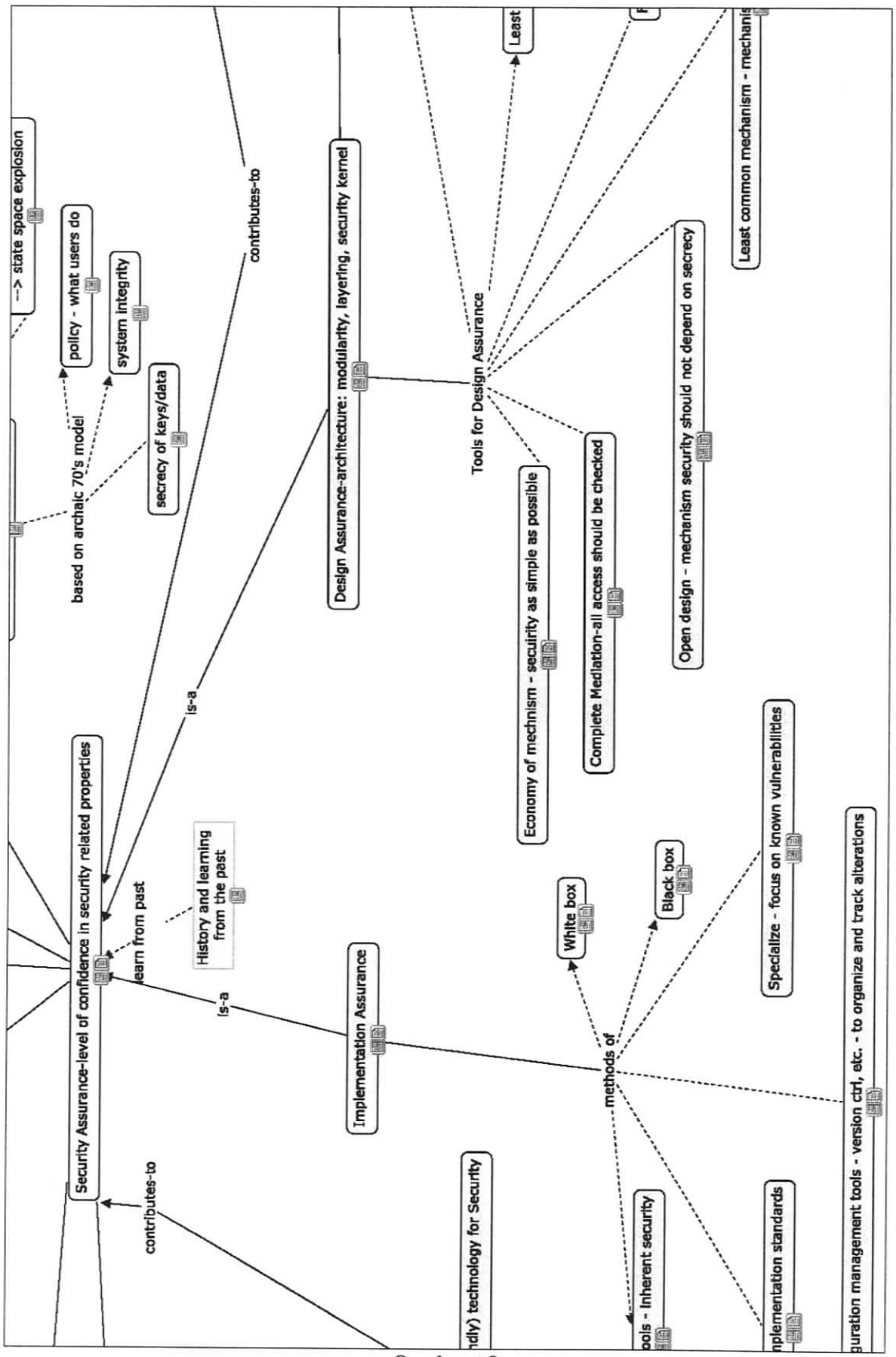
Quadrant 5


ture that exists only in the imagination"
Assurance VS fault tolerance
"Security is only as good as the weakest link in the chain, which is people"
Case Study
AEGIS applied to the EGSO project; includes interview
Jigsaw model allows for the identification of new kinds of attacks
Terms
"Diffusion of Responsibility" - notion that some-one else will take care of the problem
Capabilities: Situation that must exist in order for a particular aspect of an attack to occure
Points
Do not seperate security concerns from design considerations --> they should be intergrated
Building Secure Systems necessiates 4 goals
As evidence by media Information Security doesn't work because A. Focuses on only small part of problem of informaiton risk B. It does a poor job on even this small part
supports
example of
supports
related
ds-of
dically Organized Argument Trees
Waterfall doesn't work
contrasted by
Agile Methods
risk driven approach
Spiral approach is better
can be fixed by
nation revolution outpacing Security Criteria

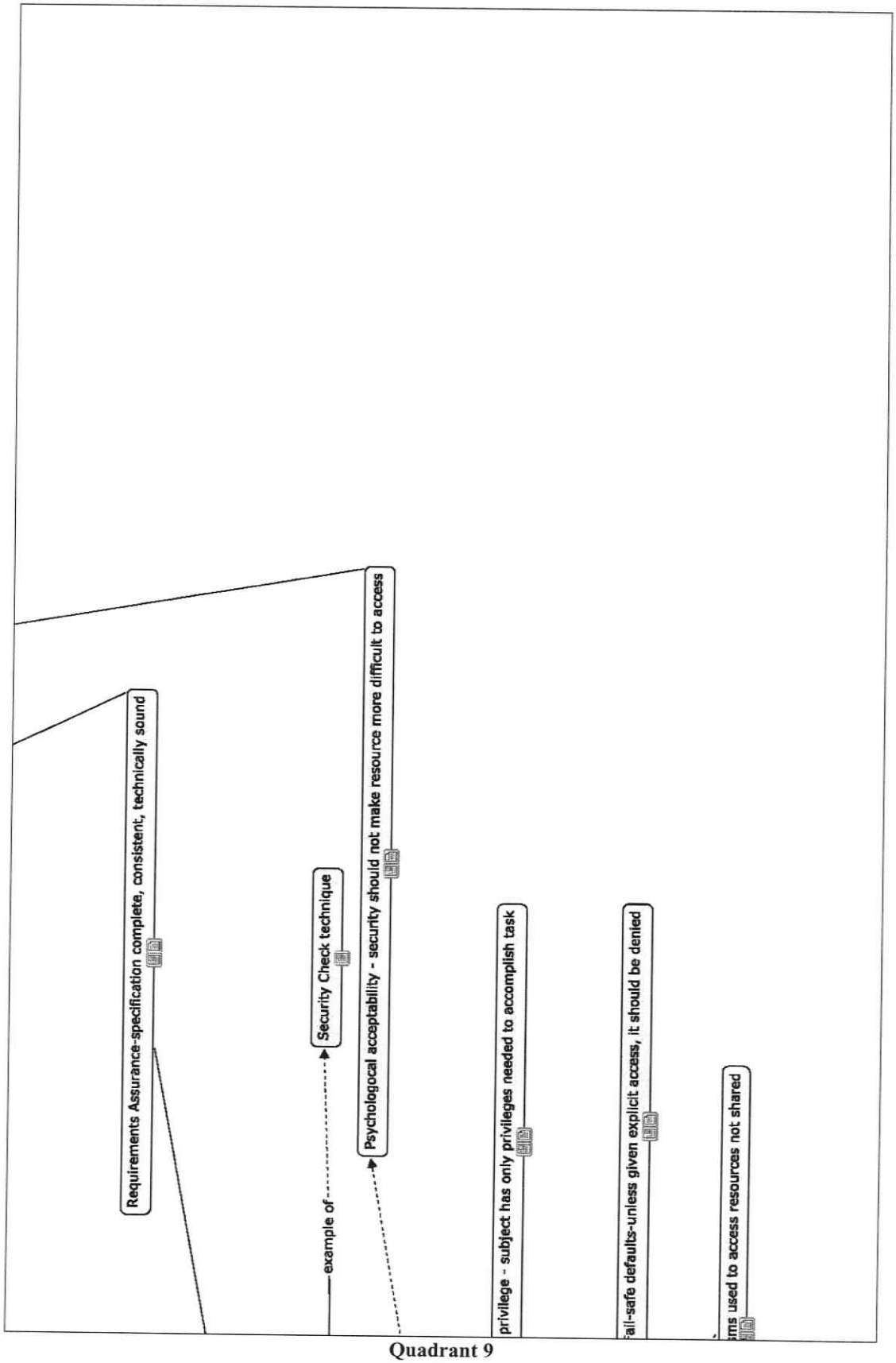
Quadrant 6


Security is local, not global
Security is specific, not general
Security is concrete, not abstract
Security is relative, not absolute
Run Time Security Evaluation concept -distributed systems
example
The live with it
Replace it (with secure version)
Extend it (add more, w/out changing original)
Tradeoffs between QOS performance and QOS security
Security is resource intensive -present different levels of service (ranges)
usability tradeoffs
Protection
Detection
Response
Assurance
relation
defined by policies/paradigms
Unhelpfulness
Information Security
factors to manage
Sensitivity
Content
Quality
Release history
Security is a component of "Quality of Service"
Engineering approaches to usable(user-frie
Tools for building
Security should be application-centric, not OS centric
contrast with
Refactor Security infrastrcture --> OS vs App, better in the OS
Build in implicit security --> transparent
Build Lego bricks for security --> reusable components
HLL languages and t
Enforcement of in
Confi

Quadrant 7


--> state space explosion
policy - what users do
system integrity
based on archaic 70's model
secrecy of keys/data
contributes-to
Design Assurance-architecture: modularity, layering, security kernel
Tools for Design Assurance
Least
Economy of mechanism - security as simple as possible
Complete Mediation-all access should be checked
Open design - mechanism security should not depend on secrecy
Least common mechanism - mechanis
Security Assurance-level of confidence in security related properties
is-a
History and learning from the past
learn from past
is-a
Implementation Assurance
White box
Black box
methods of
Specialize - focus on known vulnerabilities
contributes-to
ndly) technology for Security
ools - Inherent security
nplementation standards
guration management tools - version ctrl, etc. - to organize and track alterations

Quadrant 8


Requirements Assurance-specification complete, consistent, technically sound
example of
Security Check technique
Psychologocal acceptability - security should not make resource more difficult to access
privilege - subject has only privileges needed to accomplish task
ail-safe defaults-unless given explicit access, it should be denied
ms used to access resources not shared

Quadrant 9