A Model for Integrating Information Security into the Software Development Life Cycle

by

Lynn Ann Futcher

A Model for Integrating Information Security into the Software Development Life Cycle

by

Lynn Ann Futcher

Dissertation

submitted in fulfilment
of the requirements
for the degree

Magister Technologiae
in
Information Technology
in the
Faculty of Engineering, the Built Environment and Information Technology,
of the
Nelson Mandela Metropolitan University

Supervisor : **Prof R. Von Solms**

January 2007

Dedication

'Each of us is endowed with innate resources that enable us to achieve all we've ever dreamed of – and more!'

-Anthony Robbins-

I dedicate this to my father,

NORDAN WICHT

my mother,

VALERIE JOYCE WICHT

my husband,

SHANE HENRY FUTCHER

and my girls,

MISHALE TRACI FUTCHER

and

TAYLA COURTNEY FUTCHER

Acknowledgements

‘Alone we can do so little; together we can do so much.’

-Helen Keller-

My sincerest gratitude to the following people:

- *My supervisor, Prof Rossouw von Solms, for his invaluable support and guidance;*
- *My special friend and colleague, Cheryl Schröder, for believing in me and encouraging me every step of the way;*
- *My editorial team, Bron Kaplan and Debbie Box, for their dedication and commitment;*
- *My family, friends and colleagues, for their everlasting support and understanding.*

Abstract

It is within highly integrated technology environments that information security is becoming a focal point for designing, developing and deploying software applications. Ensuring a high level of trust in the security and quality of these applications is crucial to their ultimate success. Therefore, information security has become a core requirement for software applications, driven by the need to protect critical assets and the need to build and preserve widespread trust in computing.

However, a common weakness that is inherent in the traditional software development methodologies is the lack of attention given to the security aspects of software development. Most of these methodologies do not explicitly include a standardised method for incorporating information security into their life cycles. Meaningful security can be achieved when information security issues are considered as part of a routine development process, and security safeguards are integrated into the software application throughout its life cycle. This, in turn, will lead to users being more confident to use software applications, and to entrust today's computer systems with their personal information.

To build better or more secure software, an improved software development process is required. Security of a software application must be based on the risk associated with the application. In order to understand this risk, the relevant information assets need to be identified together with their threats and vulnerabilities. Therefore, security considerations provide input into every phase of the Software Development Life Cycle (SDLC), from requirements gathering to design, implementation, testing and deployment.

This research project presents a Secure Software Development Model (SecSDM) for incorporating information security into all phases of the SDLC, from requirements gathering to systems maintenance. The SecSDM is based on many of the recommendations provided by relevant international standards and best practices, for example, the ISO 7498-2 (1989) standard which addresses the underlying security services and mechanisms that form an integral part of the model.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ANSI:	American National Standards Institute
ASSDM:	Automated Secure Systems Development Methodology
BSI:	British Standards Institute
CCTA:	Central Computer and Telecommunications Agency
CERT:	Computer Emergency Response Team
COBIT:	Control Objectives for Information and Related Technology
CORBA:	Common Request Object Broker Architecture
CRAMM:	CCTA Risk Analysis and Management Method
CSRC:	Computer Security Resource Centre
DoD:	American Department of Defense
EU:	European Union
FDD:	Feature Driven Development
GLBA:	Gramm-Leach-Bliley Act
GMITS:	Guidelines for the Management of IT Security
HIPAA:	Health Insurance Portability and Accountability Act
IEC:	International Electrotechnical Commission
IEEE:	Institute for Electrical and Electronic Engineers
IS:	Information System/s
ISO:	International Standards Organisation
IT:	Information Technology
ITIL:	Information Technology Infrastructure Library
ITU:	International Telecommunications Union
MIS:	Management Information System/s
NIST:	National Institute for Standards and Technology
OCTAVE:	Operationally Critical Threat, Asset and Vulnerability Evaluation
OMG:	Object Management Group
OO:	Object-Oriented
OSI:	Open Systems Interconnection/s
PDA:	Personal Digital Assistant
RAD:	Rapid Application Development
RUP:	Rational Unified Process

SD^3:	Secure by Design, Secure by Default, Secure by Deployment
SDLC:	Software Development Life Cycle
SecSDM:	Secure Software Development Model
SEI:	Software Engineering Institute
SLC:	Software Life Cycle
SLCM:	Software Life Cycle Model
SPICE:	Software Process Improvement and Capability Determination
SQA:	Software Quality Assurance
SWI:	Secure Windows Initiative
TQM:	Total Quality Management
UML:	Unified Modelling Language
XP:	Extreme Programming

Chapter 1

Introduction

1.1 Introduction

Many people are reluctant to entrust current computer systems with their personal information. The main reason for this mistrust is that they are concerned about the security, reliability and ultimately the quality of the software applications associated with these systems (Mundie, de Vries, Haynes & Corwine, 2002).

Several models of software quality factors have been suggested over the years. Galin (2004) refers specifically to the factor model of McCall. This model classifies all software requirements into eleven software quality factors. These factors are grouped into three categories as follows:

- Product operation factors which include correctness, reliability, efficiency, integrity and usability;
- Product revision factors which include maintainability, flexibility and testability;
- Product transition factors which include portability, reusability and interoperability.

Aspects of software quality such as portability and flexibility, according to Wang and Wang (2003), are crucial to the study of overall software quality, but security threats and risks specifically target the software operational capabilities of correctness, reliability, efficiency, integrity and usability.

IBM is in agreement with this and summarises quality as the tangible and intangible sum of functionality, usability, reliability, performance, scalability, supportability, security and other factors. IBM states that quality results can only be attained if quality is strived for throughout all phases of the Software Development Life Cycle (SDLC). This will facilitate innovation and lower costs by increasing predictability, reducing risk and eliminating rework (Bessin, 2004).

A general past trend in software development in the past has been to invest heavily in adding functionality and delivering new capabilities that customers ask for. This is still a key focus for software developers and it is not surprising that major developers such as Microsoft are assigning a higher priority to security improvements.

Mundie et al (2002), in the White Paper on Trustworthy Computing by Microsoft, refer to four main goals that any Trustworthy Computing approach has to meet. These goals are summarised as follows:

- *Security*: the user can expect that systems are resilient to attack and that the confidentiality, integrity and availability of the system and its data are protected;
- *Privacy*: the user is able to control data about themselves, and other parties using such data adhere to fair information principles;
- *Reliability*: the user can depend on the product to fulfil its functions when required to do so;
- *Business Integrity*: the vendor of a product behaves in a responsive and responsible manner.

Experience gathered about the security of real-world software, has led to a set of high-level principles for building more secure software. Microsoft specifically refers to the following principles in their discussion of the Trustworthy Computing Development Life Cycle (Lipner and Howard, 2005):

- *Secure by Design*: The software should be architected, designed and implemented to protect itself and the information it processes and to resist attacks;
- *Secure by Default*: The default state of software should promote security. For example, software should run with the least necessary privilege. Services and features that are not widely needed should be disabled by default or accessible only to a small population of users;
- *Secure in Deployment*: Tools and guidelines should accompany software to help end users and/or administrators use it securely. Any update should be easy to deploy;

- *Communications*: Software developers should communicate openly and responsibly with end users and/or administrators on the discovery of product vulnerabilities to help them take protective action.

Wang and Wang (2003) further suggest that software developers need to select protection mechanisms via the application of appropriate security technologies and approaches to provide necessary safeguards. This can be achieved through considering the security risks and threats, and their impact on the quality of the target application. Therefore, the security of a software application must be based on its risk. The relevant information assets need to be identified together with their threats and vulnerabilities to understand this risk.

The large number of vulnerabilities identified in software applications today is due, in part, to the incredible complexity of modern systems. Each software application is unique with its own particular set of risks. An application is deemed secure when it is protected from both human and non-human threats, including accidental or intentional damage, destruction, theft, unintended or unauthorised modification (Whitman and Mattord, 2003).

1.2 Description of Specific Areas of Interest

In a climate where the protection of information is increasingly tied to the integrity of an organisation, security must be strongly coupled with the software development process to ensure that the desired level of security is achieved (Tipton and Krause, 2006). It is necessary to develop an improved software development process to build better or more secure software. Therefore, security considerations must provide input into every phase of the SDLC, from requirements engineering to design, implementation, testing and deployment.

It is important to acknowledge the recommendations provided by the relevant international standards and best practices to ensure that the proposed model conforms to best practices. For example, the international code of practice for information security management, ISO/IEC 17799 (2005) was examined and it was found that it specifically addresses issues relating to software development and maintenance. The

implementation of ISO/IEC 17799 (2005) involves establishing a cost-effective execution plan that includes appropriate security controls for mitigating identified risks, and that protect the confidentiality, integrity and availability of the information assets of the organisation.

It is important to consider that most software applications, developed currently, operate in a networked environment. The ISO 7498-2 (1989) standard describes the Basic Reference Model for Open Systems Interconnections (OSI). It provides a framework for coordinating the development of existing and future standards for the interconnection of systems. The security requirements for a particular software application can be described in terms of the five security services as defined in the ISO 7498-2 (1989) standard. However, the extent to which each of these services may be implemented, is determined by the security objectives of the envisaged application and that of the organisation.

The ISO 7498-2 (1989) standard provides the basis of information security in software systems through five basic security services, including:

- *Identification and authentication* which refers to the ability to identify the identity of all users of the system;
- *Authorisation/access control* which refers to the ability to admit or prohibit users from accessing the information assets of the organisation;
- *Confidentiality* which refers to the ability to ensure that information assets are only available to those who are authorised to access them;
- *Integrity* which refers to the ability to ensure that information has not been altered in any way by an unauthorised party;
- *Non-repudiation/non-denial* which refers to the ability to ensure that users do not deny their actions.

It is necessary to note the distinction between a security service and a security mechanism. Whereas a security service is a measure which can be incorporated to address a threat (for example, the provision of confidentiality), a security mechanism is a means to provide the security service (for example, encryption).

The five security services, according to the ISO 7498-2 (1989) standard, are supported by eight security mechanisms, namely:

- *Encipherment mechanisms*, are known as encryption or cipher algorithms. These mechanisms can help provide confidentiality of data and traffic flow information. They provide the basis for some authentication and key management techniques;
- *Digital signature mechanisms* can be used to provide non-repudiation, origin authentication and data integrity services;
- *Access control mechanisms* provide a means for using information associated with a client entity and a server entity to decide whether access to the resource of the server is granted to the client, for example, access control lists and security labels;
- *Data integrity mechanisms* are used to provide data integrity and origin authentication services;
- *Authentication exchange mechanisms*, known as authentication protocols, can be used to provide entity authentication;
- *Traffic padding* describes the addition of 'pretend' data to conceal the volumes of real data traffic. It can be used to help provide traffic flow confidentiality but is only effective if the added padding is enciphered;
- *Routing control mechanisms* can be used to prevent sensitive data from using insecure communications paths. For example, routes can be chosen to use only physically secure network components, depending on the properties of the data. Data carrying certain security labels may be forbidden entry to certain network components;
- *Notarisation mechanisms* can be used to guarantee the integrity, origin and/or the destination of transferred data. A third party notary, which must be trusted by the communication entities, will provide the guarantee typically by applying a cryptographic transformation to the transferred data.

These mechanisms are normally implemented by tools and components inherent in the particular development environment (for example, the .Net framework), or written specifically by the developer. Different security tools and components are available in the various development environments. Many of these tools and components may be

commercially available or obtainable as shareware, depending on the development environment. These tools and components help developers implement the various security mechanisms. This means that developers are no longer required to necessarily code the security components from scratch.

1.3 Description of Problem Area

It is argued that building secure software begins with the effective education of the software developers. These professionals need to be educated to put security at the heart of software design and at the foundation of the development process. In the software industry, the key to meeting current demand for improved security, is to implement repeatable processes that reliably deliver measurably improved security. This requires a more stringent software development process that focuses on security. Such a process should minimise the number of security vulnerabilities present in the SDLC and detect and remove these vulnerabilities as early in the life cycle as possible (Lipner and Howard, 2005). Tryfonas and Kiountouzis, (2002) is in agreement and suggests that new ways of addressing and resolving security issues, early within the SDLC, must be introduced in the software development arena.

The SDLC is a methodology for the design and implementation of an information system within an organisation. A methodology is a formal approach to solving a problem, based on a structured sequence of procedures. There are many representations of the SDLC (for example, the incremental model, the prototyping model and the spiral model), all illustrating a logical flow of activity from the identification of a need through to the final software product. These methodologies encompass all the standards and procedures affecting the planning, requirements gathering, analysis, design, development and implementation of a software application. However, an apparent weakness in the methodologies studied, is the lack of attention given to the security aspects of software development. It is in the opinion of the author, that each phase should result in a "security deliverable" that aims at minimising risk by employing entry and exit criteria to determine how to proceed from phase to phase. This would help ensure that security be viewed as an integral part of the SDLC, and not just as an add-on or after-thought at the end of the development process.

1.4 Problem Statement

Software developers generally ignore the idea of security. This typically leads to their applications having many security weaknesses. Jurjens (2002) argues that software developers rely mostly on their intuition in developing secure software, and do not use much systematic help or guidance. Therefore, it is not surprising that security breaches continue to occur in software applications.

One of the main flaws of typical software development methodologies, is that security is not considered until the operational requirements have been defined and the system is into its implementation stage. This approach poses a problem, since it normally results in late and expensive attempts to incorporate security into the work in progress. It is seldom possible to provide a good level of security on a retrofit basis. The security aspects associated with the development of a software application need to become an integral part of the entire development process, to overcome this problem (Booysen and Eloff, 1995).

From the literature studied, it is evident that software developers tend to neglect security issues when developing software applications. The main reason for this neglect, is that most of the existing software development methodologies do not provide any guidance for integrating security into the SDLC.

1.5 Research Objectives

The primary research objective is to present a Secure Software Development Model (SecSDM) for incorporating security into all phases of the SDLC, from requirements gathering through to implementation and system maintenance. The SecSDM is based on the ISO 7498-2 (1989) standard, which addresses the underlying security services and mechanisms that form an integral part of the model.

The secondary research objectives are to:

- Provide an introduction to software quality and Trustworthy Computing and how these relate to information security and software development;

- Discuss the various international standards and best practices pertaining to information security and software development, with a specific focus on ISO 7498-2 (1989), ISO/IEC 17799 and ISO/IEC TR 13335;
- Provide an understanding of the SDLC, the existing software development methodologies, and discover any inherent weaknesses in terms of providing for security;
- Discuss risk analysis as the first and most critical stage of the SDLC. This will be based on ISO/IEC TR 13335-3 (1998);
- Provide evidence that the SecSDM is practical.

1.6 Research Philosophy

The understanding of the research philosophy of a research project is essential, since it provides the reader with a certain viewpoint of how the project should be interpreted. This ensures that the reader is aware of what to expect when examining the project, especially with regards to its limitations and scope of the project. For example, research philosophy in the computer sciences tends to be very positivistic-oriented but the study of information systems tends to lean more towards the social sciences where people are involved, and is therefore normally more phenomenological-oriented. The development of an information system is thus viewed as a social activity.

This research project mainly involves information systems, and therefore, may engage people in one or another way. Therefore, it forms part of a social phenomenon. This implies that the emphasis is more on the meaning of what is being researched, rather than on its actual measurement. Therefore, this research project is more qualitative-oriented, which results in a phenomenological, or interpretive-oriented research philosophy.

1.7 Research Methodology

The research methods used in this research project are literature studies, modelling, arguing and questionnaires. The results of this study support the argument that the SecSDM will provide a solution for integrating security into the SDLC.

This research project began with a literature study. Its purpose was to determine the extent of the problem, as stated in Section 1.4. This was followed by further literature study that focussed on the various standards and best practices that could support the development of a secure software development model. It was important to develop an understanding of the existing software development methodologies whilst developing the SecSDM.

The main sources of information included digital libraries for example, Communications of the Association for Computing Machinery and Emerald, proceedings from various international conferences, and other relevant publications. A number of international standards for example, ISO/IEC 17799, ISO/IEC TR 13335 and ISO 7498-2, and best practices, provided primary information in support of the SecSDM. Secondary sources of information included textbooks written by experts in the relevant fields of interest for example, Information Security, Risk Analysis, Software Quality, Systems Analysis and Design.

The various standards and best practices studied were used to argue towards an improved software development model for integrating security into the SDLC. It is always necessary, for any new model, to determine whether it does actually facilitate the achievement of the primary goal for which it was established. In this case, the development of more secure software applications. A questionnaire was created that addressed each phase of the development life cycle, to establish the effectiveness of the SecSDM. Results were gathered from both 2005 and 2006 third year IT project students. These results were analysed and reported in Chapter 7.

1.8 List of Chapters

Table 1.1 describes the list of chapters developed for the successful completion of this research topic, while Figure 1.1 provides a graphical view of how these chapters relate to one another.

Chapter	Title	Brief Description
Chapter 1	Introduction	Introduces the research topic, highlighting the problem area together with the research objectives, philosophy and methodology used.
Chapter 2	Secure Information Systems	Discusses the importance of information security and its relevance to systems development, software quality and Trustworthy Computing.
Chapter 3	Standards and Best Practices Relating to Information Security and Software Development	Addresses the various standards and best practices for example, ISO/IEC 17799, ISO/IEC TR 13335, ISO 7498-2 and NIST SP 800-14 (1996) that provide primary information in support of the SecSDM.
Chapter 4	The Software Development Life Cycle	Briefly discusses the most common software development methodologies. The focus is on understanding their similarities and differences and their strengths and weaknesses as far as security is concerned.
Chapter 5	Risk Analysis	Addresses risk analysis as the first and most critical stage of the SDLC. ISO/IEC TR 13335-3 (1998) forms the basis of this discussion.
Chapter 6	The Secure Software Development Model	Describes the Secure Software Development Model (SecSDM). It is a ten-step process for integrating security into the five phases of the SDLC. Each of these steps is described in detail.
Chapter 7	Secure Software Development in Practice	Reports on the results gathered from 2005 and 2006 third year Information Technology (IT) project students at a tertiary institution.
Chapter 8	Conclusion	Provides a summative conclusion and describes how the research objectives, as stated in Section 1.5, have been addressed. It suggests further research with respect to this particular research topic.

Table 1.1: List of Chapters

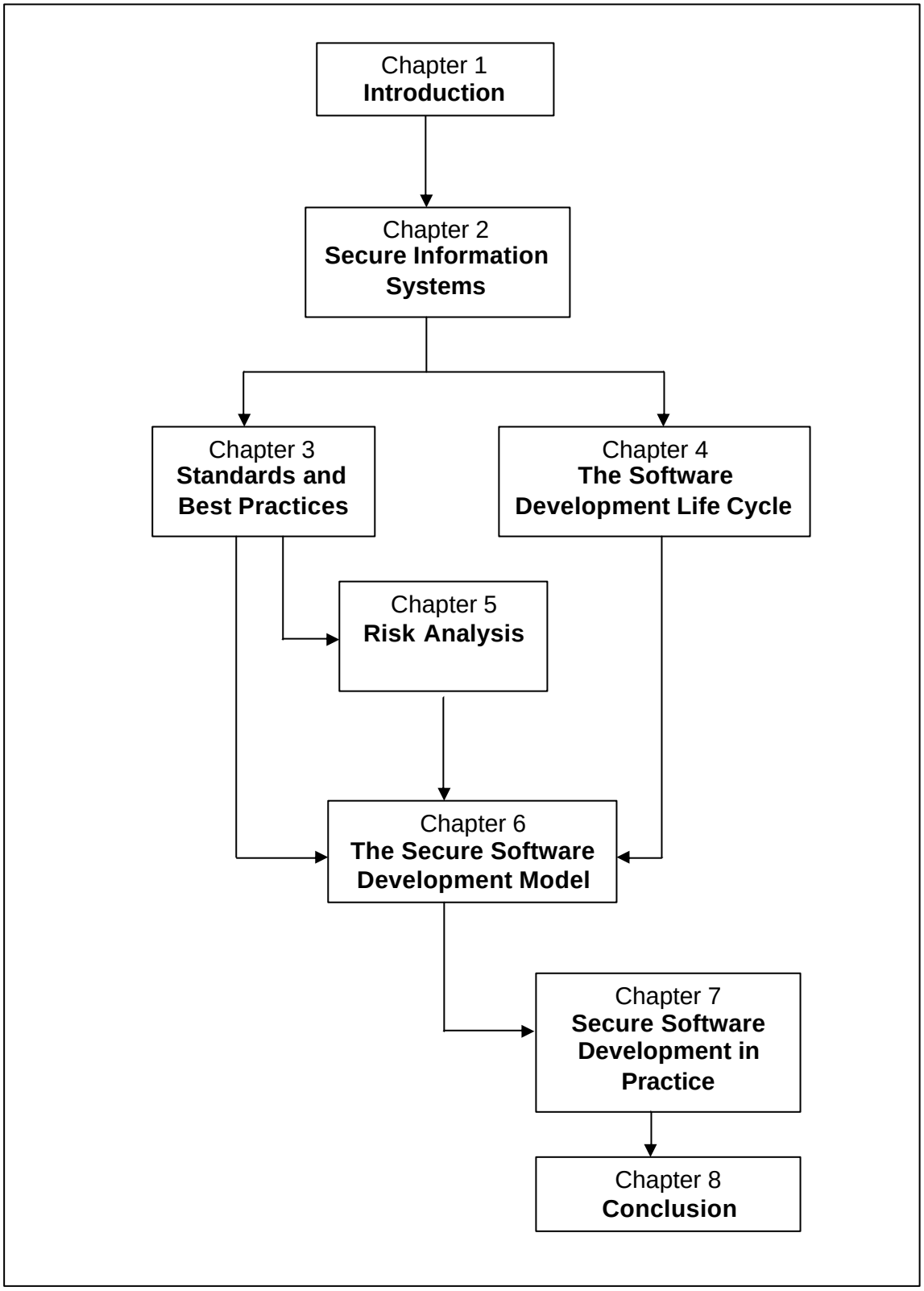

Figure 1.1: Structure of Research Project

1.9 Conclusion

Meaningful security is easier to achieve when security issues are considered as part of the routine development process and security safeguards are integrated into the application during its design. Security safeguards that are integral to a system are usually easier to use and less visible to the user. It is generally more expensive to retrofit security than to integrate it into a software application.

It is evident that information security is becoming a focal point for designing, developing and deploying software applications. However, security features alone do not necessarily increase the quality of the software, because they still need to be properly implemented. The education of software developers is therefore crucial to creating secure software products.

The primary focus of this research project is the development of the SecSDM that aims to ensure the secure development of software applications, whilst instilling security awareness and discipline at each stage of the SDLC.

Chapter 2

Secure Information Systems

2.1 Introduction

An Information System (IS) is broadly defined as the entire set of software, hardware, data, people and procedures necessary to use information as a resource within an organisation. Each of these components have their own security requirements. Software, however, is in all likelihood the most difficult component to secure. There are daily reports warning weaknesses or other fundamental flaws in software. Hardware, on the other hand, is a physical asset that is typically protected from harm or theft by physical security, such as locks and keys. Since access to data is usually the main objective of intentional attacks, it is important that data which is stored, processed and transmitted is protected. An information system is only as strong or trustworthy as its weakest link which is all too frequently human. It is not surprising that people are often considered a major threat to information security, although not always intentionally. Education and training, awareness and technology need to be properly employed to reduce such risks, and to prevent accidental or intentional damage or the loss of information. Procedures are information in their own right, therefore, proper education on their protection is necessary.

The main focus of this research is on the software component of an IS, rather than the system in its entirety. The aim of this chapter is to provide a broad overview of the context within which the research has been conducted. It provides background information relating to the various aspects of securing information systems, specifically from a software development point of view.

2.2 The Importance of Software

Software no longer simply supports back offices and home entertainment. It is the lifeblood of most businesses today and has become intertwined within the fabric of everyday life. Businesses and organisations, regardless of their size, increasingly need information systems to respond to the problems and opportunities of the current global

business environment. There is increasing reliance on computerised activities, which if they fail or are poorly used, can have harmful consequences.

Examples of the harmful consequences can be found in both mission-critical and safety-critical systems of organisations. Mission-critical systems are essential to the viability of an organisation. The organisation cannot function should the computer system fail. A safety-critical system is one on which human lives depend, such as an air-traffic control system. Should these systems fail, it will put human lives at stake. For these reasons, the software applications underlying these systems need to be secure.

Another important issue to consider is the Internet. The Internet clearly continues to fundamentally and radically change the role that software plays in the business world. It has emerged as an unparalleled public medium for communication and commerce – and it is changing the world. The growing role of electronic commerce is even changing the means of shopping. People are increasingly using the Internet to view and order goods and services online. In addition, growing numbers of people use the Internet to telecommute to work. Telecommuting means that employees work at home and stay in touch by means of computer-based communications. Internet users are increasingly opening bank accounts and trading stocks online. Online banking means that customers can use a Web browser to access their accounts, balance cheque books, transfer funds and pay accounts online. Online stock trading sites enable investors to buy and sell stocks online, without the aid of a broker. All of these online activities increase the number of potential threats to their users.

The explosive rise of wireless computing has, in addition to the Internet, created a need to protect information systems and corporate networks with refined security solutions. Mobile commerce may not be very different from the existing Internet. Users simply use cellular phones, Personal Digital Assistants (PDAs), notebooks or touch pads to connect wirelessly to the Internet. A benefit of wireless connections is that they create new opportunities for online users but simultaneously increase the potential risk. If people are connected everywhere, a vast amount of information becomes available at any time.

Research has shown that many people are reluctant to entrust the current information systems with their personal details, because they are increasingly concerned about their security and reliability. The ability to design systems that can be used safely and effectively, may be increased by carefully considering the security aspects of information systems. Amid the social and ethical issues raised by the rapid spread of ubiquitous, highly-networked computers, threats to privacy and anonymity are amongst the most contentious. The Trusted Computing approach, for business users, reduces one of the largest barriers to electronic business, namely the fear that everything from identities to intellectual property to confidential corporate data could be stolen.

The purpose of software engineering is to find ways of building quality software which ensures a high level of trust. The following section addresses the important issues relating to software quality and Trustworthy Computing.

2.3 Software Quality and Trustworthy Computing

Software quality is often viewed as a luxury – something that can be sacrificed, if necessary, for added functionality, faster development or lower costs. Successful software development organisations have found that, in practice, an organisational commitment to quality increases development, reduces costs, and allows new features to be added with greater ease. An organisation that develops poor quality software is essentially always looking backward, spending time and money fixing defects in "completed" software. On the other hand, organisations that build quality from the beginning are forward-looking, able to innovate, and pursue new opportunities (Bessin, 2004).

Quality software is fit for use. According to Bessin (2004), quality is a well-defined process for creating a useful product that adds value for both the user and the software developer. A high-quality process means the business does not lose time reworking, refactoring and rewriting. Quality must be defined in terms of the target audience, i.e. the software users, to create a useful product. Software quality must focus on more than simply eliminating software bugs. This implies that a quality application is not one that provides the correct results without failing. Software quality is complex because it includes the tangible and intangible aggregate of functionality, usability, reliability,

performance, scalability, supportability, security and any other factors important to the customer and business. Software quality improvement, like software development, is an iterative process which adds to the complexity.

Bessin (2004) argues that continuously ensuring software quality will consistently cost less than ignoring quality considerations. In effect, he further suggests that raising product quality costs next to nothing when done correctly.

It is important to understand the attributes and characteristics that contribute to software quality, to improve its quality. Wang and Wang (2003) refer specifically to the factor model of McCall, which classifies all the software requirements into 11 software quality factors. These factors are grouped into three categories as follows:

- Product operation factors which include correctness, reliability, efficiency, integrity and usability;
- Product revision factors which include maintainability, flexibility and testability;
- Product transition factors which include portability, reusability and interoperability.

According to Wang and Wang (2003), aspects of software quality in product revision and transition, such as portability and flexibility are crucial to the study of overall software quality, however security threats and risks specifically target the product operation capabilities, i.e. correctness, reliability, efficiency, integrity and usability. These factors can be defined as follows:

- *Correctness* is the extent to which a program satisfies its specification and fulfils the functional objectives of the customer so that the system is behaving correctly given the prescribed situation;
- *Reliability* is the extent to which a program can be expected to perform its intended function with required precision and is available at the expected time periods;
- *Efficiency* is the amount of computing resources and interactions required by a program to perform its function;

- *Integrity* is the extent to which access to software or data by unauthorised persons is controlled and that the software or data is verifiable throughout its lifetime;
- *Usability* is the time and resources required to learn, operate, prepare input, and interpret output of a program.

The user of a system is never able to obtain perfect knowledge of the system in use, nor of the external or internal threats, and is therefore unable to exactly determine its security. By gathering as much knowledge as possible about the system, the user will form an idea or belief about the security, i.e. the user will gain a certain trust in the system. Security can be understood as an idealistic goal for the system designers and developers, whereas trust represents the actual, imperfect knowledge of the user, about how successful the designers have been in reaching their idealistic goal. The question posed is, what are the methods that establish trust in information systems? These methods should be dynamic and take into consideration new evidence such as security incidents and new threats (Josang, van Laenen, Knapskog & Vandewalle, 1997).

There was a response to this question from Microsoft in January 2002, when Bill Gates sent his Trustworthy Computing memo to all Microsoft employees. The memo outlined the need to deliver more secure and robust applications to users because of the increased threat to computer systems (Howard and LeBlanc, 2003).

Mundie et al (2002) refer to the following goals that any Trustworthy Computing approach has to meet:

- *Security* – the user can expect that systems are resilient to attack, and that the confidentiality, integrity and availability of the system and its data are protected;
- *Privacy* – the user is able to control data about themselves, and those using such data adhere to fair information principles;
- *Reliability* – the user can depend on the product to fulfil its functions when required to do so;
- *Business integrity* – the developer of a product behaves in a responsive and responsible manner.

The Microsoft development teams have been helped to meet their short and long-term security goals through their Secure Windows Initiative (SWI) team adopting a simple set of strategies called SD^3 – secure by design, secure by default and secure in deployment. This means that steps have been taken to protect the confidentiality, integrity and availability of data and systems at every phase of the software development process – from design to delivery and maintenance.

Similarly, IBM and IBM Rational have developed a comprehensive software development and deployment solution that helps teams innovate and deliver higher quality results by simplifying, automating and integrating the software development and deployment process. This process, referred to as Rational Unified Process (RUP), is discussed in more detail in Chapter 4. Information security is a subset of software quality as will be discussed in the following section.

2.4 Information Security

The protection of information is a topic that has existed since the earliest of times. It is not surprising that, some of the strongest developments in information security have come from the military. The traditional purpose of information security is to prevent breaches of confidentiality, integrity and availability by implementing threat countermeasures expressed as technical aspects of the information system. The purpose of the countermeasures is to generate trust which is a human and social phenomenon. This trust allows users to use a system in ways which they otherwise would avoid, so that in practice it becomes more valuable and a more powerful tool (Josang et al, 1997).

The task of information security professionals is to protect the confidentiality, integrity and availability of information and information systems, whether in the state of transmission, storage or processing. According to Whitman and Mattord (2003), information security can be defined as the protection of information and the systems and hardware that use, store and transmit that information. Certain tools are necessary such as policy, awareness, technology, education and training to protect the information and its related systems.

The value of information arises from the characteristics it possesses. Should any one of these characteristics be compromised, the value of the information is equally compromised. For example, the timeliness of information is a critical factor to users since information often loses its value if delivered late. The critical characteristics, as defined by Whitman and Mattord (2003), are as follows:

- *Availability*: the availability of information enables authorised users who need access to information to do so without interference or obstruction, and to receive it in the correct format;
- *Accuracy*: information is accurate when it is free from mistakes or errors and it has the value that the end user expects;
- *Authenticity*: information is authentic when it is the information that was originally created, placed, stored or transferred;
- *Confidentiality*: confidentiality of information is ensuring that only those with the rights and privileges to access a particular set of information are able to do so. This is especially important when involving personal information about employees, customers or patients;
- *Integrity*: the integrity of information is the quality or state of being whole, complete and uncorrupted;
- *Utility*: information has utility when it serves a particular purpose;
- *Possession*: information is said to be in possession if one obtains it, independent of format or other characteristic.

The transaction and accounting data that is used by the information systems of an organisation, is used to illustrate these characteristics. An example of confidentiality, is that hackers should be prevented access to the credit card data of the customer. Integrity, on the other hand, is related to employees having the ability to modify their payroll records to change their pay rates. The availability of information is equally important, as this relates to one being denied access to the information assets needed to make business decisions, for example, the latest inventory levels required to decide on reorder purchases.

The privacy and confidentiality of information is an increasingly important and controversial issue in this technology-driven society. Several serious privacy violations

have been reported by government agencies. For example, some law enforcement officers used the national computer systems to run background checks for private agencies. Wireless applications offer even more potential invasions of privacy. Many proposed mobile commerce methods use the location of the consumer to offer information and advertisements (Post and Anderson, 2003).

The Internet and electronic commerce add further complications to protecting the information assets of a company. An online company may need to give their customers access to important company information to provide the best electronic commerce site. For example, customers may like to know if an item is in stock before placing an order. This will entail connecting the customer to the inventory system of the company. Anytime a connection from the Internet to company data is opened, the interaction needs to be carefully controlled. This trade-off between user access and information security is an ongoing problem which often arises because the security tools used are not sophisticated enough (Post and Anderson, 2003).

A further difficulty with providing information security lies in identifying the user. In a manual security system, a guard can be used to physically identify each person by asking to see identification. The most common method of identifying users to information systems is with a password. Passwords, however, are not the perfect solution to identifying users. No matter how well they are chosen, or how often they are changed, there is always a chance that someone could guess the password. In fact, passwords are deemed so risky that top-secret information of the American government is stored on computers that cannot be connected to phone lines or the Internet. There is a smaller chance that the information could be compromised, by physically preventing outsiders from accessing the computers. This approach to information security, however, is not feasible for most business applications, because it would render these systems unusable.

Denial of service attacks have gained importance in the last few years, especially since the essence of an electronic commerce site is the ability to reach customers 24 hours a day. The flooding of the site with meaningless traffic means no one can use the service and the company may go out of business.

It is important to realise that attackers are highly skilled and very clever. They have the time and energy to probe and analyse software applications for security vulnerabilities. The consequences of compromised systems are many and varied, including loss of production, loss of customer faith and loss of money. Information security should be considered a balance between protection and availability. The level of security must allow reasonable access but protect against potential threats, to achieve such balance (Whitman and Mattord, 2003).

The immediate need for organisations to protect critical information assets continues to increase. These organisations will rely on the next generation of professionals to possess the correct mix of skills and experiences to develop more secure computing environments. There is a need to prepare software developers to recognise the potential threats and vulnerabilities in existing systems and to learn to design and develop secure systems needed presently and into the future.

Microsoft has discovered that convincing designers, developers and testers of the importance of security is reasonably easy, since most people care about the quality of their product. These professionals, although specialists in their own right, require a broad range of security knowledge. For example, in addition to understanding cryptography, they need to understand vulnerabilities, prevention, accountability, authentication, authorisation and real world security requirements that affect users. Many people have the ability to recognise and complain about bad security and subsequently offer remedies that secure the system in a manner which renders it unusable. There is a fine line between secure systems and usable secure systems that are useful for the intended audience (Howard and LeBlanc, 2003).

Information security, as expected by the author, is becoming a focal point for designing, developing and deploying software applications. However, it is important to determine the security needs and requirements of the specific application domain. Security features alone do not necessarily increase the quality of the software, because they need to be properly implemented. Education is crucial in creating secure software products. The questions posed are, what is secure software development and how could security awareness and discipline be instilled at each stage of the SDLC?

2.5 Secure Software Development

Currently, the Internet has brought millions of unsecured computer networks into communication with each other. No longer are computers islands of functionality with little, if any, interconnectivity. In the past, most people did not care about information security, as long as an application performed its task successfully. However, times have changed. In the current Internet era, virtually all computers are interconnected. These computers include servers, desktop personal computers, cellular phones and other mobile devices. The problem is that these interconnected computers can be attacked at any point. The ability to secure the stored information of each computer is now influenced by the security on every other computer to which it is connected.

The smooth operation of communication and computing systems becomes vital as global networks expand the world. However, recurring events such as virus and worm attacks and the success of criminal attackers clearly illustrate the weaknesses in current information technologies and the need for heightened security of these systems.

It is important to consider some of the reasons why people choose not to build secure systems, to understand the need for secure software development. Some of the reasons, according to Howard and LeBlanc (2003), include the following:

- Security is boring;
- Security is often seen as a functionality disabler;
- Security is difficult to measure;
- Security is usually not the primary skill or interest of the designers and developers creating the product;
- Security means not doing something exciting and new.

These reasons illustrate why security is considered as something that “gets in the way” and costs money, while offering little or no financial return. However, there are many arguments that show that secure applications are good for business. First and foremost, secure products are quality products. It can be argued that a product designed and developed by security-aware professionals is likely to exhibit fewer security defects than one developed by more undisciplined professionals. However, the need for security and its strength are context-driven. This means that different situations call for different

solutions. The key when developing secure software products, is to design and build them so that they are sufficiently secure for the environment in which they will operate (Howard and LeBlanc, 2003).

However, there are many reasons for building secure software whether viewed from an organisational, a software developer or an end-user perspective. The failure to design and build secure software, from the perspective of the software developer, leads to more work in the long run and a bad reputation. This, in turn, can lead to the loss of sales for an organisation as customers switch to a competing product perceived to have better security support. Users, on the other hand, demand applications to work as expected. They do not want their systems to be infected by viruses, their credit card information posted on the Internet or their medical data stolen. Software applications are expected to securely manipulate, transmit and store confidential user and corporate information. Users demand secure applications and they see such systems as a right and not a privilege. Although most users require secure environments, security should be hidden so that it does not "get in the way".

An important part of delivering secure systems is raising awareness through security education. This implies educating the end users, the software designers, developers and all the stakeholders involved in the software development process. According to Howard and LeBlanc (2003), many software developers understand how to build security features into software, but many have never been taught how to build secure systems. A reason for this may be that it is far easier to teach people about security features than to train them to think with a security mindset. A simple understanding of how features work will not help build a secure system. It is necessary to know how to alleviate security threats when building secure systems. This is achieved by identifying such threats and their associated risks before being able to alleviate them.

Education is critical to delivering secure systems. Software developers cannot be expected to understand how to design, build, test, document and deploy secure systems. They may know how various security features work, however, that is not enough to ensure a secure system. The teaching of these professionals needs examination. They clearly need more education about secure requirements analysis, secure design and secure coding to build secure software. Additionally, Howard and LeBlanc (2003)

suggest that the ultimate, and most important goal of security education is to teach people not to introduce security flaws into the product in the first place. Software developers need to be taught to design security into every aspect of their applications. This means that all product functional specifications should include a section outlining the security implications of each feature.

It is important that software developers understand the processes involved in designing and building secure systems that can withstand attack. It is recognised that software will always have vulnerabilities, regardless of how much time was spent designing and building it. However, software developers should strive at reducing the overall number of vulnerabilities and making it substantially more difficult to find and exploit vulnerabilities.

It is important to adopt a disciplined process that incorporates all aspects of software development to secure software applications. There exists a need to update and improve the software development process itself. Process improvements should be added at every step of the SDLC, regardless of the particular methodology chosen, to better focus on security issues. Innovations that will add more accountability and structure in terms of securing the software development process are required.

2.6 Conclusion

It is within highly integrated information technology environments that information security is becoming a focal point for designing, developing and deploying software applications. The ensuring of a high level of trust in the security and quality of software systems is crucial to their success. Although software programs are the vessels that carry the lifeblood of information through an organisation, they are often created under the demanding constraints of project management. This means that security is usually applied as an afterthought, rather than being integrated from the very beginning. Organisations can no longer afford to consider security issues after the application has been constructed.

Information security can be regarded as a trust enabler. Securing information is about identifying, measuring and managing the various risks that threaten the confidentiality,

integrity and availability of information assets. Integration of security into software development is necessary to build secure systems. Confidentiality ensures that there is no deliberate or accidental disclosure of sensitive information. Integrity protects against the deliberate or accidental corruption of information. Availability protects against deliberate or accidental actions that cause information resources to be unavailable to users when needed.

Information systems literature fails to present a comprehensive methodology for integrating security into systems development. Security of information is of particular importance and must be considered throughout the software development lifecycle. The entire project team must commit to improving quality throughout the software development process, including post-deployment, and that commitment must be driven from the top by business leadership.

The business benefits of including quality-oriented activities in all phases of the software development cycle are both broad and deep. These measures facilitate innovation and lower costs by increasing predictability, reducing risk and eliminating rework and they will differentiate a business from its competitors.

Secure systems contribute to quality systems. Code designed and built with security as a prime feature is more robust than code written with security as an afterthought. It is not possible to have quality without security (Howard and LeBlanc, 2003).

The following chapter addresses some standards and best practices relating to information security and software development. These standards and best practices form the basis of the Secure Software Development Model (SecSDM), as described in Chapter 6.

Chapter 3

Standards and Best Practices Relating to Information Security and Software Development

---

3.1 Introduction

The previous chapter addressed the importance of secure information systems with a particular focus on information security, secure software development, software quality and Trustworthy Computing. This chapter discusses a number of internationally recognised standards and best practices that need to be considered when examining secure software development.

Standards consist of a specific set of rules, procedures or conventions that are agreed upon between parties to perform business operations more uniformly and effectively (Killmeyer, 2006). Standards can be viewed as a set of predetermined guidelines in which the issues, considerations and effects of doing something have already been analysed by someone authorised, experienced and qualified in the specified area. This implies that standards reflect industry best practices. This implies that security and software development standards reflect industry best practices. When standards are not taken into consideration, they have a definite impact on both the cost and the amount of risk an organisation is exposed to.

A number of best-practice frameworks exist to help organisations assess their security risks, implement appropriate security controls and, comply with governance requirements, and privacy and information security regulations. It must be noted, however, that best practices are a moving target. For example, something that worked well two years ago may be completely worthless against current threats. Information security has to keep abreast of these new threats, together with the methods, techniques, policies, guidelines, educational and training approaches and technologies used to combat them. Similarly, the software development community should consider standards and best practices as the basis for software planning, design and deployment.

Standards are normally prescriptive, contain requirements for conformance and generally employ the verb "shall". Best practices, on the other hand, present recommended approaches and usually employ the verb "should". Guidelines often suggest several alternative approaches to good engineering practice and traditionally employ the verb "may" (Schultz, 2000).

The following sections discuss various information security and software development standards and best practices, and introduce many of the organisations and institutions responsible for the setting of such standards. This chapter concludes by proposing a set of criteria for secure software development, based on the specific standards and best practices studied.

3.2 Key Role-Players

Standards committees have a vital role to play in protecting telecommunications and information technology systems, firstly, by maintaining an awareness of security issues; secondly, by ensuring that security considerations are a fundamental part of system specifications; and thirdly, by providing guidance to assist implementers and users in the task of making information systems and services sufficiently robust (Bertine, Chadwick, Euchner & Harrop, 2004).

Many organisations are the sources of software development and information security standards. The Institute for Electrical and Electronic Engineers (IEEE), the National Institute for Standards and Technology (NIST), the International Standards Organisation (ISO), the American National Standards Institute (ANSI), the American Department of Defense (DoD), the British Standards Institute (BSI), the Common Request Object Broker Architecture (CORBA) and the Object Management Group (OMG) are all sources of these standards.

The IEEE regularly publishes software development standards. ISO standards cover design and description in the ISO 6593; documentation in the ISO 9127; and software quality management in the ISO 9000 series. ISO/IEC JTC1 SC7 has released international standards for software quality and software engineering. ANSI works closely with the IEEE in developing industrial software development standards. The

DoD publishes military standards for software, however, the BSI serves as a rich source of standards concerning every aspect of software development.

The main international standards bodies relevant to information security are the ISO, the International Electrotechnical Commission (IEC) and the International Telecommunications Union (ITU). Their main function is the production of base standards. In addition, governments around the world are reacting with legislation and regulations that seek to establish requirements for information security. For example, the Gramm-Leach-Bliley Act (GLBA), the Health Insurance Portability and Accountability Act (HIPAA), the Sarbanes-Oxley Act, and the EU Data Protection Directive outline specific security and privacy requirements that share common themes. The GLBA was signed into law in 1999. Its primary purpose is to provide for the privacy of customer information by financial services organisations. It requires financial institutions to (Peltier, 2005):

- Ensure the security and confidentiality of customer records and information;
- Protect against any anticipated threats or hazards to the security or integrity of such records;
- Protect against unauthorised access.

Similarly, the HIPAA, passed in 1996 by President Clinton, includes strict rules for the privacy and security of health information, giving individuals more control over how their information is used. It impacts virtually every aspect of health care in America. The privacy and security rules within HIPAA govern the use, disclosure and handling of any identifiable patient information (Peltier, 2005).

The Sarbanes-Oxley Act of 2002, on the other hand, deals with effective internal controls within public corporations. It is not prescriptive with respect to security, but does outline that both general controls and application-level security controls are required.

Companies and organisations in America have devoted significant time and resources to achieve compliance with many facets of legislation, such as HIPAA, Sarbanes-Oxley

and GLBA. These are not legislation in other countries but many organisations have taken note of these acts in establishing their own standards and best practices.

Jones and Rastogi (2004) argue, however, that the information security-related actions of government, private and public organisations do not directly address the critical problem that is at the root of most security and privacy breaches. This is the failure of software developers to take a security view of their products from inception through deployment and beyond.

The following section introduces the main standards, best practices and guidelines that were used to establish a set of criteria for secure software development.

3.3 Standards and Best Practices

Some best practices that facilitate the implementation of security controls include Control Objectives for Information and Related Technology (COBIT), ISO/IEC 17799, ISO/IEC TR 13335, Information Technology Infrastructure Library (ITIL) and Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE).

It is the opinion of the author that the focus on the ISO/IEC 17799 (2005) standard and the ISO/IEC 13335 technical reports is warranted because they provide the most comprehensive approaches and guidelines to information security management. The other best practices focus more on IT governance, in general, or on the technical aspects of information security.

3.3.1 ISO/IEC 17799 standard

One of the most widely referenced and discussed security models is the *"Information Technology – Code of Practice for Information Security Management"*, which was originally published as the British Standard BS 7799. In 2000, this Code of Practice was adopted as an international standard framework for information security by the ISO and IEC committees as ISO/IEC 17799 (Whitman and Mattord, 2003). It provides best practice recommendations on information security management for use by those who are responsible for initiating, implementing or maintaining information security management systems. It was last updated in 2005.

The 2005 version addresses the security of information in its broadest sense, providing best business practice, guidelines and general principles for implementing, maintaining and managing information security in any organisation, producing and using information in any form. The ISO/IEC 17799 (2005) standard recognises that the level of security that can be achieved purely through technical means is very limited. The required level of security – established through assessing the levels of risk and associated costs from breaches of security, measured against the costs of implementing security – should always be driven by appropriate management controls and procedures. Information security management requires, at a minimum, participation by all employees within an organisation. It can also require participation by other stakeholders, for example shareholders, suppliers, third parties and customers.

It was found, in considering the ISO/IEC 17799 (2005) standard, that it specifically addresses issues relating to systems development and maintenance. This occurs in Section 12 as follows:

- Section 12.1 (Security Requirements of Information Systems) aims to ensure that security is built into information systems and states that *"Security requirements should be identified and agreed prior to the development and/or implementation of information systems";*
- Section 12.2 (Correct Processing in Applications) aims to prevent errors, loss, unauthorised modification or misuse of information in applications and states that *"Appropriate controls should be designed into applications. Additional controls may be required for systems that process, or have an impact on sensitive, valuable or critical information. Such controls should be determined on the basis of security requirements and risk assessment";*
- Section 12.3 (Cryptographic Controls) aims to protect the confidentiality, authenticity and integrity of information by cryptographic means and states that *"A policy should be developed on the use of cryptographic controls";*
- Section 12.4 (Security of System Files) aims to ensure the security of system files and states that *"Access to system files and program source code should be controlled, and IT projects and support activities conducted in a secure manner. Care should be taken to avoid exposure of sensitive data in test environments";*

- Section 12.5 (Security in Development and Support Processes) aims to maintain the security of application system software and information and states that *"Managers responsible for application systems should also be responsible for the security of the project or support environment. They should ensure that all proposed system changes are reviewed to check that they do not compromise the security of either the system or the operating environment";*
- Section 12.6 (Technical Vulnerability Management) aims to reduce risks resulting from exploitation of published technical vulnerabilities and states that *"Technical vulnerability management should be implemented in an effective, systematic, and repeatable way with measurements taken to confirm its effectiveness. These considerations should include operating systems and other applications in use".*

The implementation of the ISO/IEC 17799 (2005) standard involves putting in place a cost-effective execution plan that includes appropriate security controls for mitigating identified risks and protecting the confidentiality, integrity and availability of the information assets of an organisation.

The ISO/IEC 27001 standard was published in November 2005 and complements the ISO/IEC 17799 (2005) code of practice for information security management. It defines an Information Security Management System. The objective of the standard is stated as *"to ensure the selection of adequate and proportionate security controls that protect information assets and give confidence to interested parties including an organisation's customers"*. The standard itself has been harmonised to align with other management systems standards, such as ISO 9000. It is this standard, rather than ISO/IEC 17799 (2005), against which certification is achieved.

3.3.2 ISO/IEC TR 13335 guideline

A further guideline to be considered is ISO/IEC TR 13335. This technical report, entitled "*Information Technology – Guidelines for the Management of IT Security*" (GMITS), provides further guidelines for the management of IT security. It consists of five parts, namely:

- *Part 1:* Concepts and Models - introduces a series of concepts and models for IT security;

- *Part 2:* Managing and Planning IT security – presents the issues that an organisation must tackle before establishing or altering its IT security program;
- *Part 3:* Techniques for the Management of IT security – pays particular attention to the complex topic of IT security risk assessment;
- *Part 4:* Selection of Safeguards – provides pointers to readily available safeguard catalogues;
- *Part 5:* Safeguards for External Connections – looks at the problem of crossing the "trust boundary".

An important part of the IT security management process is the assessment of risks and how they can be reduced to an acceptable level. It is necessary to take into account the specific needs and risks of each information system and to align these with the business objectives, organisational and environmental aspects. It is important to perform a risk analysis, to determine the security requirements of a particular system. For the purposes of this proposed risk analysis approach, it is assumed that the application being developed is at high risk. It is necessary to follow the guidelines of the detailed risk analysis approach, as determined by ISO/IEC TR 13335-3 (1998), are followed. Figure 3.1 clearly highlights the various elements that need to be considered when performing a detailed risk analysis. It requires the careful consideration of assets, threats and vulnerabilities which will ultimately facilitate the selection of effective safeguards appropriate to the assessed risks.

A detailed risk analysis involves the identification and valuation of assets, the assessment of threats to those assets, and an assessment of the associated vulnerabilities. The results from these activities are used to assess the risks and consequently identify justified security safeguards. By implementing this approach it is more likely that appropriate safeguards will be identified. Additionally, the results of the detailed analysis can be used in the management of system changes. The disadvantage, however, is that it normally requires a considerable amount of time, effort and expertise to obtain such detailed results. It is important that a simplified approach is used, but not to the detriment of the results achieved, to educate software developers in risk analysis. A good approach may be to find a balance between minimising the time and effort spent in identifying the safeguards, while still ensuring that the high risk assets are appropriately

protected. The incorporation of an initial quick and simple approach is likely to gain greater acceptance and therefore be more widely used.

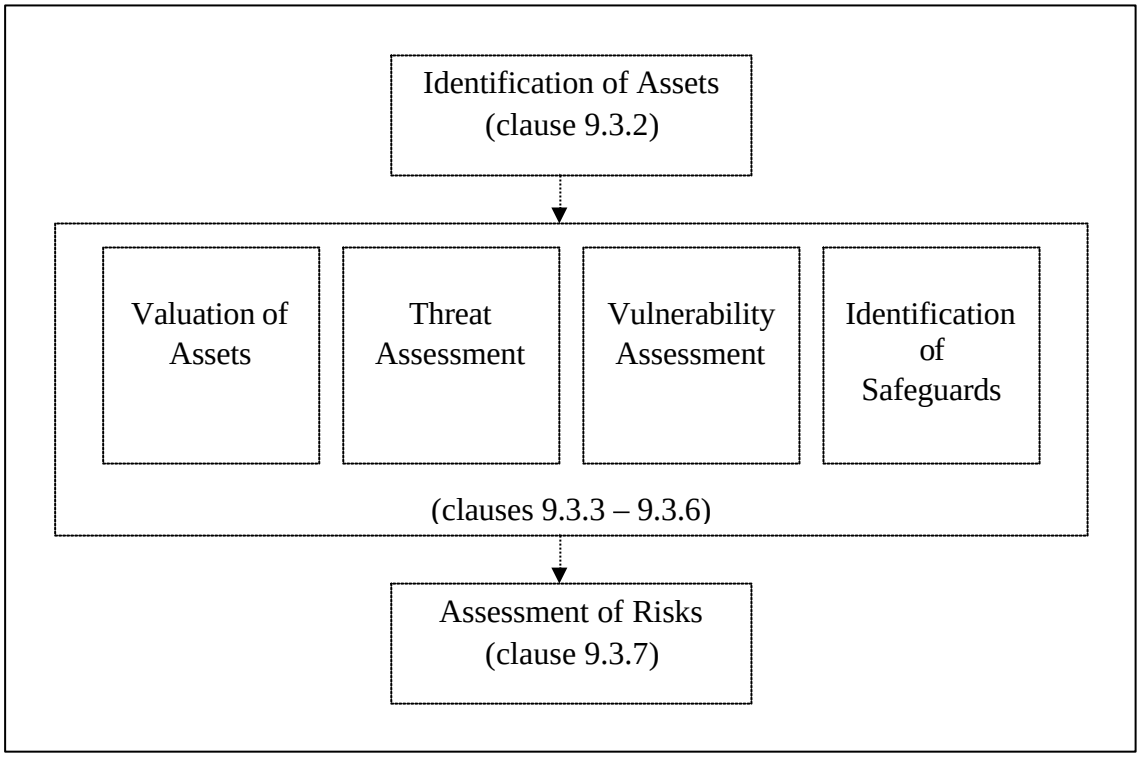

Figure 3.1: Detailed Risk Analysis (ISO/IEC TR 13335-3,1998)

Many of the guidelines provided by GMITS are used in the proposed risk analysis approach and may also assist in educating software developers. This approach is discussed in greater detail in Chapter 5.

3.3.3 NIST security models

A number of security models are described in the many documents available from the Computer Security Resource Centre (CSRC) of the NIST institute. The NIST special publication SP 800-14 (1996), entitled *Generally Accepted Principles and Practices for Securing Information Technology Systems* is of specific interest. This publication provides principles that a security team should integrate into the entire information security process, in addition to providing best practices and security principles.

Principles for Securing Information Technology Systems NIST SP 800-14 Generally Accepted Principles and Practices for Securing Information Technology Systems	
Principle 1	Establish a sound security policy as the foundation for design.
Principle 2	Treat security as in integral part of the overall system design.
Principle 3	Clearly delineate the physical and logical security boundaries governed by associated security policies.
Principle 4	Reduce risk to an acceptable level.
Principle 5	Assume that external systems are insecure.
Principle 6	Identify potential trade-offs between reducing risk and increased costs and decrease in other aspects of operational effectiveness.
Principle 7	Implement layered security (ensure no single point of vulnerability).
Principle 8	Implement tailored system security measures to meet organisational security goals.
Principle 9	Strive for simplicity.
Principle 10	Design and operate an IT system to limit vulnerability and to be resilient in response.
Principle 11	Minimise the system elements to be trusted.
Principle 12	Implement security through a combination of measures distributed both physically and logically.
Principle 13	Provide assurance that the system is, and continues to be, resilient in the face of expected threats.
Principle 14	Limit or contain vulnerabilities.
Principle 15	Formulate security measures to address multiple overlapping information domains.
Principle 16	Isolate public access systems from mission critical resources (e.g. data, processes, etc.).
Principle 17	Use boundary mechanisms to separate computing systems and network infrastructures.
Principle 18	Where possible, base security on open standards for portability and interoperability.
Principle 19	Use common language in developing security requirements.
Principle 20	Design and implement audit mechanisms to detect unauthorised use and to support incident investigations.
Principle 21	Design security to allow for regular adoption of new technology, including a secure and logical technology upgrade process.
Principle 22	Authenticate users and processes to ensure appropriate access control decisions both within and across domains.
Principle 23	Use unique identities to ensure accountability.
Principle 24	Implement least privilege.
Principle 25	Do not implement unnecessary security mechanisms.
Principle 26	Protect information while being processed, in transit and in storage.
Principle 27	Strive for operational ease of use.
Principle 28	Develop and exercise contingency or disaster recovery procedures to ensure appropriate availability.
Principle 29	Consider custom products to achieve adequate security.
Principle 30	Ensure proper security in the shutdown or disposal of a system.
Principle 31	Protect against all likely classes of “attacks”.
Principle 32	Identify and prevent common errors and vulnerabilities.
Principle 33	Ensure that developers are trained in how to develop secure software.

Table 3.1: Principles of NIST SP 800-14 (NIST, 1996)

This publication lists 33 principles as indicated in Table 3.1, however, the following principles are of particular importance in developing a model for secure software development:

- *Principle 2:* Treat security as an integral part of the overall system design;
- *Principle 4:* Reduce risk to an acceptable level;
- *Principle 7:* Implement layered security (i.e. ensure no single point of vulnerability);
- *Principle 14:* Limit or contain vulnerabilities;
- *Principle 19:* Use common language in developing security requirements;
- *Principle 25:* Do not implement unnecessary security mechanisms;
- *Principle 32:* Identify and prevent common errors and vulnerabilities;
- *Principle 33:* Ensure that developers are trained in how to develop secure software.

The following sub-section discusses the interconnection standards that are particularly important in the current networked environment and must therefore be considered when examining secure software development.

3.3.4 Open systems security (interconnection standards)

Security has become a concern to almost everyone with the rapid and widespread growth in the use of data communications, particularly the Internet. Most software applications developed today operate in a highly networked environment. The operation of the Internet and these networked environments rely primarily on interconnection standards.

Open Systems Security Architecture (X.800) was developed during the eighties to address elements of security architecture. This was the first stage in the development of a suite of standards to support security services and mechanisms (Bertine et al, 2004). X.800 provides a general description of security services and the related mechanisms that can be used to provide these services. The X.800 standard was developed specifically as the Open Systems Interconnection (OSI) security architecture but the underlying concepts of X.800 have been shown to have much broader applicability and

acceptance. This standard represents the first internationally agreed consensus on the definitions of the five basic security services, namely:

- Authentication;
- Access Control;
- Data Confidentiality;
- Data Integrity;
- Non-Repudiation.

The X.800 standard defines, in terms of the seven-layer OSI Basic Reference Model, the most appropriate location for implementing the security services.

The ISO 7498-2 (1989) standard describes the Basic Reference Model for OSI. It presents a framework for coordinating the development of existing and future standards for the interconnection of systems. Additionally, the ISO 7498-2 (1989) standard provides the basis of information security in software systems through the five basic security services as determined by X.800, namely:

- *Identification and authentication*: this is the ability to identify and uniquely authenticate all users of a system;
- *Authorisation/access control*: this is the ability to allow or prohibit users from accessing the information assets of an organisation;
- *Confidentiality*: this is the ability to ensure that information assets are only available to those who are authorised to access them;
- *Integrity*: this is the ability to ensure that information is complete and uncorrupted and that it has not been altered in any way;
- *Non-repudiation/non-denial*: this is the ability to ensure that users do not deny their actions.

It is important to note the distinction between a security service and a security mechanism. A security service is a measure which can be put in place to address a threat (e.g. provision of confidentiality) and a security mechanism is the means to provide a service (e.g. encryption).

The five security services referred to by the X.800 and ISO 7498-2 (1989) standards are supported by eight security mechanisms, namely:

- *Encipherment mechanisms*, commonly known as encryption or cipher algorithms. These mechanisms can help provide confidentiality of data and traffic flow information. They provide the basis for some authentication and key management techniques;
- *Digital signature mechanisms* can be used to provide non-repudiation, origin authentication and data integrity services;
- *Access control mechanisms* provide a means for using information associated with a client entity and a server entity to decide whether access to the resources of the server is granted to a client, for example, access control lists and security labels;
- *Data integrity mechanisms* are used to provide data integrity and origin authentication services;
- *Authentication exchange mechanisms*, also known as authentication protocols, can be used to provide entity authentication;
- *Traffic padding* describes the addition of 'pretend' data to conceal the volumes of real data traffic. It can be used to help provide traffic flow confidentiality but is only effective if the added padding is enciphered;
- *Routing control mechanisms* can be used to prevent sensitive data from using insecure communications paths. For example, depending on the properties of the data, routes can be chosen to use only physically secure network components. Data carrying certain security labels may be forbidden to enter certain network components.
- *Notarisation mechanisms* can be used to guarantee the integrity, origin and/or the destination of transferred data. A third party notary, who must be trusted by the communication entities, will provide the guarantee typically by applying a cryptographic transformation to the transferred data.

The ITU-T Recommendations Report X.805 in support of the X.800 and ISO 7498-2 (1989) standards, as cited by Bertine et al (2004), refers to eight security dimensions that have been defined to address end-to-end security in networked applications. In addition to the five basic security services already mentioned, X.805 identifies a further three dimensions, namely:

- Privacy;

- Availability;
- Communication Security.

The functionalities of the five basic security services of X.800 (authentication, access control, data confidentiality, data integrity, non-repudiation) match the functionalities of the corresponding security dimensions of X.805, as shown in Figure 3.2. The privacy, availability and communication security dimensions of X.805 offer new types of network protection. These eight security dimensions are summarised as follows:

- *Authentication* serves to confirm the identities of the communicating entities;
- *Access Control* protects against unauthorised use of network resources;
- *Data Confidentiality* protects data from unauthorised disclosure. Data confidentiality ensures that the data content cannot be understood by unauthorised entities;
- *Data Integrity* ensures the correctness or accuracy of data. The data is protected against unauthorised modification, deletion, creation, and replication and provides an indication of these unauthorised activities;
- *Non-repudiation* provides the means for preventing an individual or entity from denying having performed a particular action related to data by making available proof of various network-related actions (such as proof of obligation, intent or commitment; proof of data origin, proof of ownership, proof of resource use);
- *Privacy* provides for the protection of information that might be derived from the observation of network activities;
- *Availability* ensures that there is no denial of authorised access to network elements, stored information, information flows, services and applications due to events impacting the network;
- *Communication security* ensures that information flows only between the authorised end points.

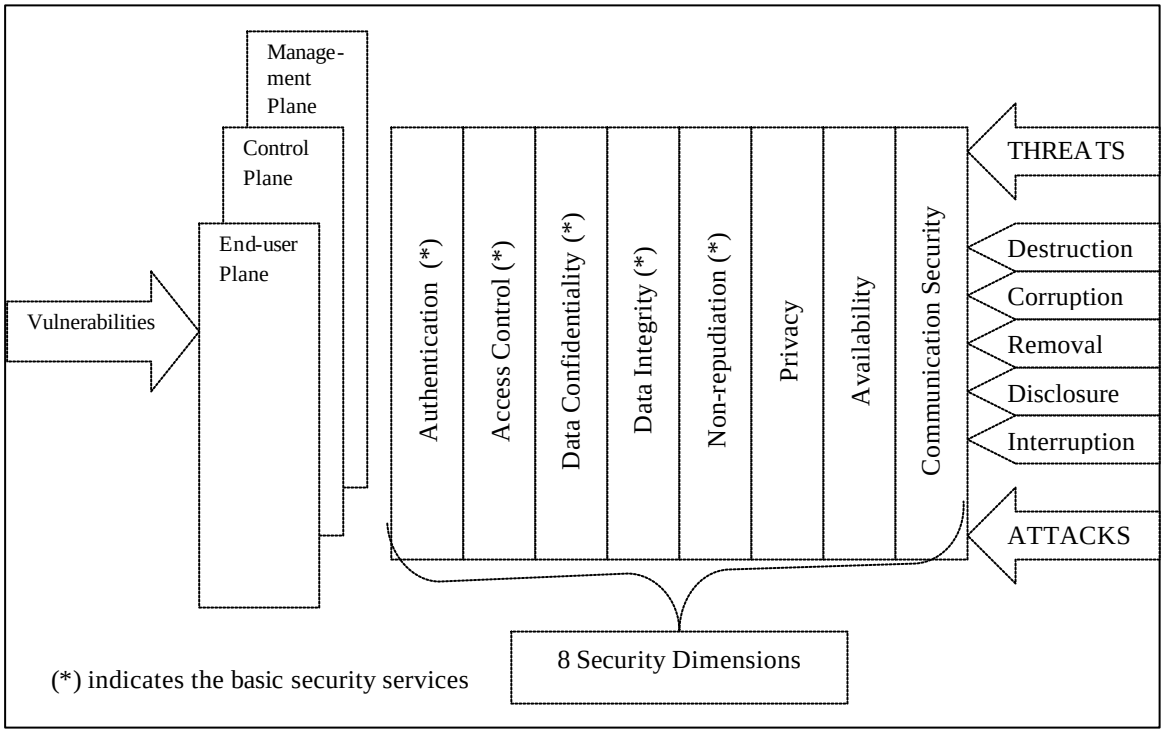

Figure 3.2: Security Architectural Elements in ITU-T Recommendation X.805 (Bertine et al, 2004)

3.3.5 Quality standards

A principal factor in the performance of an organisation is the quality of its products or services. Information security is definitely a contributing factor to quality and should therefore be covered by the ISO 9000 umbrella. ISO 9001 is part of the ISO 9000 family of quality standards and as such applies to both manufacturing and software development (Tipton and Krause, 2006).

The ISO committee has developed a collection of standards for software processes called Software Process Improvement and Capability Determination (SPICE). These standards support ISO 9000 quality standards. ISO 9000 was not developed initially with software development in mind. Software and information systems developers are advised to use ISO 9000-3, entitled *"Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software"*, due to the specificity of software as a product. ISO 9000-3 is a set of guidelines that relate directly to software preparation but is still not ideal in terms of information security. However, ISO 9000 does force the organisation to identify the control measures that ensure the quality of products is maintained. This could be related to a type of risk analysis, where all risks

affecting the information systems of the organisation are identified. Additionally, ISO 9000 forces the organisation and its top management to state their policy regarding quality and to adhere to it. Information security will gain a lot more attention if the information security policy were included in this quality document (Von Solms and Meyer, 1995).

The ISO 9001 standard is not specifically aimed at software products, but it is possible to apply this standard to software quality control. Since software development projects have specific characteristics (for example, frequent changes in requirements during the development process and the invisible nature of the product during its development), there is a need for quality assurance procedures which are tailored towards software development. The purpose of Software Quality Assurance (SQA) is to ensure that work gets done the way it is supposed to be done. More specifically, Van Vliet (2000) summarises the goals of SQA as:

- To improve software quality by appropriately monitoring the software and its development process;
- To ensure full compliance with the established standards and procedures for the software and the development process;
- To ensure that any inadequacies in the product, the process, or the standards are brought to the attention of management to ensure inadequacies can be fixed.

3.3.6 Software development standards and best practices

The ISO/IEC 12207 (1995) standard defines a framework for software life cycle processes. It contains a hierarchy of processes, activities and tasks to be applied in an environment where software is being developed, supplied, acquired, operated and maintained. The standard was first published in 1995 as ISO/IEC 12207:1995, "*Information Technology – Software Life Cycle Processes*".

Three principle processes are identified in the IEEE 1074 (1995) standard for software development, namely (Peters and Pedrycz, 2000):

- *Requirements*, i.e. decide what a system must do, its activities, risks and testing plan;

- *Design*, i.e. determine how a system computes, its specific functions and structure;
- *Implementation*, i.e. produce source code, documentation and tests; validate and verify.

The Rational Unified Process (RUP) describes how to effectively deploy commercially proven approaches to software development for software development teams. It captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organisations. They are typically referred to as "best practices" because they are used in industry by many successful software development organisations. The following six best practices are of significant importance, namely:

- *Develop software iteratively.* This is of particular importance because it is not possible to sequentially define the entire problem, design the entire solution, build the software and test the product at the end. An iterative approach is required that allows for successive refinements and promotes that high risk items be addressed at every stage of the lifecycle;
- *Manage requirements.* A software application is more likely to fulfil user needs if the functional requirements and constraints have been elicited, organised and documented in such a way that they drive the design, implementation and testing of the application;
- *Use component-based architectures.* Components are non-trivial modules or sub-systems that fulfil a clear function. The use of component-based architectures promotes more effective software reuse;
- *Visually model software.* Visual abstractions help to communicate different aspects of the software; understand how the elements of the system fit together; ensure that the building blocks are consistent with the code; maintain consistency between a design and its implementation and promote unambiguous communication within the software development team;
- *Verify software quality.* Quality is not treated as an afterthought but rather built into the process, in all activities and involving all participants;

- *Control changes to software.* This involves the ability to manage change, through making certain that each change is acceptable and being able to track changes.

Microsoft (2005) presents an overview of information security at Microsoft in a TechNet report. Some of the lessons learned and best practices relating to applications include:

- Apply basic security principles at the earliest design stages to create reliable, security-enhanced applications;
- Create awareness and a high priority for security by establishing education programmes for internal application development and for sharing best practices.

Microsoft authors, Howard and LeBlanc (2003), in support of developing secure software, stress that software developers should avoid adding security as an afterthought for the following reasons:

- Adding any feature (including security) as an afterthought, is expensive;
- Adding security later may change the way features have been implemented. This, too, is expensive;
- Adding security later involves wrapping security around existing features, rather than designing features with security in mind;
- Adding security as an afterthought may change the application interface, which may in turn break the code that has come to rely on the current interface.

One of the security best practices, as identified by Howard and LeBlanc (2003), is to perform threat modelling first. Their motivation is that the threat modelling process allows developers to determine which parts of the envisaged product are most at risk and should, therefore, be evaluated more deeply.

3.4 Criteria for Secure Software Development

A high-level analysis of the key information security and software development standards and best practices shows a considerable amount of overlap. Such an overlap reinforces the fact that those activities found in multiple approaches can surely be regarded as essential or core best practices.

The following set of criteria, based on the standards and best practices studied, has been established to ensure secure software development:

- Ensure that developers are trained in how to develop secure software (NIST SP 800-14, Microsoft TechNet Report);
- Information security must be integrated into the software development life cycle. It is vital that security be a well-thought-out process at all stages, from system inception and design through implementation and deployment (Jones and Rastogi, ISO/IEC 17799, BS 7799, NIST SP 800-14, RUP, Microsoft TechNet Report);
- Some form of risk analysis, risk assessment and threat modelling must be carried out during the initial phase of the software development lifecycle (Howard and LeBlanc, 2003; BS 7799, ISO/IEC 17799, NIST SP 800-14, ISO/IEC TR 13335-3);
- Security requirements must be identified early in the development lifecycle (ISO/IEC 17799, BS 7799, NIST SP 800-14, RUP);
- Relevant security services must be determined (ISO 7498-2, X.800, X.805);
- Design appropriate security controls and mechanisms into application systems to meet the security requirements (ISO 7498-2, ISO/IEC 17799, BS 7799, NIST SP 800-14). These information security controls and mechanisms should be selected as a result of some risk-based approach;
- Ensure that any system changes do not compromise the security of the application (ISO/IEC 17799, BS 7799, RUP).

3.5 Conclusion

Computing and networking are now such an important part of life and the need for effective security measures to protect the information systems of governments, industry, commerce and consumers is imperative. An increasing number of countries have data protection legislation that requires compliance with demonstrated standards and best practices to ensure the confidentiality, integrity and availability of their information systems. Many organisations recognise their ability to create a more secure, trustworthy environment by implementing and enforcing an internal set of best practices, together with achieving legislative compliance.

The most comprehensive approach, of the best practice frameworks available, is based on the implementation of the international information security management standard ISO/IEC 17799 (2005). The NIST institute provides a number of best practices and security principles that security teams can integrate into the entire information security process.

Most software applications developed currently operate in a highly networked environment, therefore, some of the key interconnection standards were considered together with quality and software development standards and best practices.

A set of criteria, based on the various standards and best practices studied, has been established to ensure secure software development. These criteria, as listed in Section 3.4, are addressed in the following chapters. Chapter 4 specifically addresses the software development life cycle and the extent to which the various models currently incorporate information security issues.

Chapter 4

The Software Development Life Cycle

4.1 Introduction

Chapter 3 discussed the standards and best practices to be considered when studying secure software development. This chapter expands on this issue by addressing a number of alternative software development models and the typical phases of the SDLC.

Software plays a key role in the modern world. The value of software is derived from its ability to increase productivity and efficiencies, its resiliency to attack, and its ability to perform at the required levels during times of crisis and normal operations.

The development of successful software applications require an engineering approach. This approach is characterised by the application of scientific principles, methods, models, standards and theories that make it possible to manage, plan, analyse, model, design, implement, maintain, measure and evolve a software system (Peters and Pedrycz, 2000).

Software development typically follows a life cycle. This can be defined as an orderly set of activities conducted and managed for each software development project (Maciaszek, 2001). It identifies the phases along which the software product moves, from initial inception to maintenance and eventual replacement. The development of software has always been regarded as a difficult task. For this reason, many different methodologies have been proposed by various researchers to guide the software development process as a whole.

In the early years of business computing, information software development was a disorganised, ad-hoc process that frequently produced unsatisfactory results. Software was typically delivered late, went over budget and importantly, did not provide the services and requirements that users expected. The SDLC approach was developed to improve the quality of information systems. The SDLC has proved to produce much

better quality information systems, by encouraging an organised approach to problem-solving (Pfaffenberger, 2002).

The SDLC is a methodology for the design and implementation of an information system within an organisation. A methodology is a formal approach to solving a problem, based on a structured sequence of procedures. The use of a methodology ensures a rigorous process and avoids missing any steps that could lead to compromising the end goal. It can, therefore, be argued that using a methodology increases the probability of success. Once a methodology has been adopted, the key milestones corresponding to each of the phases are established, and a team of individuals is selected and made accountable to accomplish the project goals (Whitman and Mattord, 2003).

The methodology can be referred to as the SDLC or Software Life Cycle (SLC). Peters and Pedrycz (2000), define the SLC as "the period of time beginning with a concept for a software product and ending whenever the software is no longer available for use". They state that a Software Life Cycle Model (SLCM) represents the activities, their inputs and outputs (for example, documents, tables and measurements) and any interactions during its life cycle. Examples of software life cycle models include the waterfall, incremental, spiral, prototyping, evolutionary, object-oriented and agile models. These models are discussed in Section 4.3.

The number of phases may vary from one model to another, or even the names of the phases may differ, but the general objectives remain the same. The term "software development life cycle" is used in terms of this research, because the focus is more on software development than on information systems development at large.

The objective of this chapter is to establish the extent to which quality, and specifically security, is currently integrated into the SDLC. This is achieved by examining the most common software development models, and the typical phases of the SDLC. The following section addresses traditional software development.

4.2 Traditional Software Development

The waterfall model is the oldest, known SDLC model. It was first identified in 1970 as a formal alternative to the code-and-fix software development method prevalent at the time. It was the first model to formalise a framework for software development, placing emphasis on up-front requirements and design activities, and on producing documentation during the early stages of development. It describes a sequence of activities that begins with concept exploration and concludes with maintenance and eventual replacement. Peters and Pedrycz (2000) refer to it specifically as the forward engineering of software products. An understanding of its strengths and weaknesses improves the ability to assess other, possibly more effective life cycle models that are based on this traditional approach.

The waterfall model was created to control large, complex development projects. Therefore, it does not work well for small projects that require rapid development or heavy user involvement with many changes. This model has been highly criticised but, it does have a number of advantages. Post and Anderson (2003) refer specifically to the increased control and the improved ability to monitor large projects that it provides. It is able to tolerate changes in staffing, and the resulting system is normally easier to maintain owing to the strong emphasis on documentation. It identifies a fixed set of documents that are produced as a result of each phase in the life cycle. However, generating such detailed specifications and signing-off documents is costly and time-consuming, and may delay the installation of a system for extended periods of time.

A further criticism of the waterfall model is that the notions of rapid prototyping and incremental development are absent from this model. Many critics refer to it as being very rigid and inflexible. It encourages the freezing of specifications early in the development process. Changes can be made later on in the process, but they become very costly. Additionally, it does not prescribe how to reverse engineer an existing system.

Maciaszek (2001) suggests that such a structured approach to analysis and design is characterised by a number of features, some of which, however, are not well aligned with modern software engineering. These features include:

- The tendency to be sequential and transformational, rather than iterative and incremental. This means that it does not facilitate a seamless development process through iterative refinement and incremental software delivery;
- The tendency to deliver inflexible solutions that satisfy a set of identified business functions, but which can be difficult to scale up and extend in the future;
- The assumption that development takes place from scratch, and that it does not support reuse of pre-existing components.

The application of the waterfall model, because of its inherent weaknesses, should be limited to those situations in which their requirements and their implementation are well understood. Large mainframe or complex client-server systems, and systems with highly complex technical requirements, may continue using this traditional approach. However, since it not practical for most of the current applications which are running on highly networked PCs and workstations, a number of alternative software development models have emerged over recent years. Section 4.3 addresses some of these models.

4.3 Alternative Software Development Models

Various software development models have evolved from attempts to optimise the waterfall model. Modern software development processes are invariably iterative and incremental. This means that details are typically added in successive iterations allowing for changes and improvements to be introduced as needed. Incremental development allows for a number of releases of software modules, thereby maintaining user satisfaction and providing important feedback to modules still under development (Maciaszek, 2001).

A software development process determines the activities and procedures to enhance collaboration in the development team to ensure that a quality product is delivered to the customer. According to Maciaszek (2001), a process model should:

- State an order for carrying out the activities;
- Specify what development artefacts are to be delivered and when;
- Assign activities and artefacts to developers;
- Offer criteria for monitoring the progress of the project, for measuring its outcomes, and for planning future projects.

Maciaszek (2001) argues that software development processes cannot be standardised to be automatically embraced by an organisation. Each organisation has to develop its own process model or customise it from a generic process template, such as the template provided by the Rational Software Corporation, known as the Rational Unified Process (RUP).

There are many representations of the SDLC to choose from, all illustrating a logical flow of activity from the identification of a need through to the final software product. These methodologies use all the standards and procedures which will affect the planning, requirements gathering, analysis, design, development and implementation of a software system. Each SDLC model has its own strengths and weaknesses and may, therefore, be better suited to certain types of projects within an organisation. The expected size and complexity of the system, development schedule, and lifespan of a system will affect the choice of which SDLC model to use.

4.3.1 The incremental/evolutionary model

A common problem with software development is that the software is needed quickly, but takes time to fully develop. One solution is to form a compromise between timescales and functionality, providing "interim" deliveries of software. Each delivery has reduced functionality, but serves as a stepping stone towards the fully functional software. It is possible to use such an approach as a means of reducing risk.

The product is said to evolve within the incremental/evolutionary life cycle model because it consists of the planned development of multiple releases. Generally, increments become smaller and implement fewer requirements each time. It typically entails the continual overlapping of development activities and produces a succession of software releases. However, it can be costly if it is assumed that a current release is superseded by an improved version of the software later.

The incremental/evolutionary approach to software development reduces costs, controls the impact of changing requirements and produces an operational system more quickly by developing the system in a building-block fashion (Futrell, Shafer & Shafer, 2002). The main criticisms of this model are that it is does not allow for iterations within each

increment, and that the definition of a complete, fully functional system must be done early in the life cycle to allow for the definition of the increments.

4.3.2 The prototyping model

Prototyping is the process of building a working replica of a system. This approach focuses on producing software products quickly. Prototyping makes it possible for clients and developers to see how a particular module works in the early stages of a software development process. Prototyping is thus viewed as a means of reducing risk since potential problems may be discovered before committing to a fully-fledged system.

Prototyping solves the waiting problem of the waterfall model. It is said to aid the understanding of a system. It is effective for smaller applications, and when user requirements are unclear. However, it cannot easily be applied to large mainframe-based systems with complex processing instructions and calculations; in these cases the traditional approach is more appropriate (Laudon and Laudon, 1998). A major criticism is that it usually ignores quality, reliability, maintainability and safety requirements.

4.3.3 The spiral model

The spiral life cycle model, introduced by Boehm in 1986, combines many good features of other software development models. These include the idea of baseline management (i.e. the documents associated with cycle phases), apparent in the waterfall model, the overlapping phases which are found in the incremental model, and early versions of a software application from the prototyping model. These software development models can be coupled with the spiral model in a natural way (Van Vliet, 2000).

A number of problems have to be solved during the development of a software application. Problem-solving normally means the most difficult parts are tackled first, or the parts that have the highest risks with respect to its successful completion. Each convolution of the spiral gives rise to the following activities (Van Vliet, 2000):

- Identify the sub-problem which has the highest associated risk;
- Find a solution for that problem.

The basic assumption of the spiral model is that the form of software development cannot be completely determined in advance (Peters and Pedrycz, 2000). Each phase of the spiral model has four principle activities, including:

- Elaborating software entity objectives, constraints and alternatives;
- Evaluating alternatives relative to objectives and constraints and identifying any major sources of risk;
- Elaborating the definition of software entities for a project;
- Planning the next cycle, terminating the project if too risky and securing management commitment.

The original spiral methodology developed by Boehm serves as the basis for Booysen's Automated Secure Systems Development Methodology (ASSDM) (Booysen and Eloff, 1995). This methodology refers specifically to a security spiral, divided into four quadrants, namely:

- A "*Determine objectives, alternatives, constraints*" quadrant that helps to define the security requirements needed by the system;
- An "*Evaluate alternatives, identify, resolve risks*" quadrant to help in selecting the best security development strategy, while considering the security risks associated with the requirements;
- A "*Develop, verify next-level product*" quadrant that describes the phases involved in system development;
- A "*Plan next phases*" quadrant that combines the deliverables from the other quadrants to assist the project team and developer in planning the next spiral.

The driving force behind the spiral model is a strategy for minimising risk. It is evident that it encompasses the strengths of the waterfall model, while including risk analysis, risk management and support and management processes. The risk analysis activity is used to show the beginning of a new spiral. The main advantage of the spiral-type model is that it is possible to return to previous spirals during any stage of the development process. This approach, however, has the unwanted side-effect of increasing development costs (Peters and Pedrycz, 2000). It does not specifically address the issue of how developers specify, design and test the conceptual construct of software being developed.

4.3.4 The object-oriented model

The Object-Oriented (OO) approach to software development was popularised in the nineties. The Object Management Group approved a standard for it called the Unified Modelling Language (UML). The OO approach follows the iterative and incremental process. It is more data-centric when compared to the structured approach since it evolves around class models. The development of classes supports multiple instances of objects and encapsulation, which leads to reuse. This means that it simplifies software development by hiding much of the complexity. The OO model prescribes software development in terms of a synergy between abstraction, modularity, encapsulation, hierarchy, typing, concurrency and persistence.

Developers currently tend to use the OO approach because of the many associated technical advantages, namely abstraction, encapsulation, reuse, inheritance and polymorphism. Maciaszek (2001) further suggests that these technical properties may lead to many benefits, including greater reusability of code and data, shorter development times, increased programmer productivity, improved software quality and greater understandability.

4.3.5 Rapid application development

Companies, in their never-ending quest to design software applications with greater speed, started to entertain the idea of Rapid Application Development (RAD). It allows developers to code on the fly in contrast to the waterfall model. Its supporters subscribe to the theory that delivering a functional product as quickly as possible is of the utmost importance. They can upgrade the software later if the demand arises. They believe that the sooner the end-users see a result, the more confident they are that the application will meet their expectations (Laudon and Laudon, 1998).

A RAD approach is characterised by the quick turnaround time from requirements definition to completed system. It follows a sequence of evolutionary system integrations or prototypes that are reviewed with the customer, discovering requirements along the way (Futrell et al, 2002).

Microsoft, as a commercial software vendor developing highly complex systems used by millions of people, has developed its own methodologies for quickly creating code.

The have been a proponent of RAD to reduce the time it takes to complete large projects. This methodology is designed to segment the code so that hundreds of programmers can work on it simultaneously. It relies on creating code that can be modified later for improvements and patches (Post and Anderson, 2003).

4.3.6 The rational unified process

The Rational Unified Process (RUP) of IBM, embodies many of the best practices of software development, including an iterative development approach, managing change, modelling visually and ensuring quality.

The iterative phases of RUP (i.e. analysis, design, development, testing and deployment) emphasize one or more core process disciplines. In each phase and each discipline, an attention to quality is emphasized to identify problems earlier in the life cycle, when they are typically easier to solve. Bessin (2004) refers to the various quality issues to be addressed at each phase of the SDLC as follows:

- *Analysis* – Meta Group reports that up to 80% of the issues leading to customer dissatisfaction can be traced to poor understanding of requirements. Quality starts with analysis of the business to ensure that system requirements accurately reflect, with clarity, the business or customer needs. Managing requirements is the cornerstone of success. These requirements must be written in a manner that is understandable to software developers;
- *Design* – during design, the primary focus is on architecture. Poor architecture causes a wide array of quality problems including fragility, lack of scalability and resistance to modification;
- *Development* – it is estimated that developers make 100 to 150 errors for every thousand lines of code. There is now a stronger emphasis on developer-lead testing and analysis, where tests are built as a precursor to code. Unit testing and runtime analysis have become more mainstream;
- *Test* – system level functional and performance tests are an integral part of continuously ensuring quality;
- *Deploy and monitor* – eventually business applications are implemented. Inevitably, functional and reliability errors will occur. Constant monitoring and

assessment ensure the viability of a deployed system and ensures the ability to rapidly detect and respond to inadequate performance.

4.3.7 Agile development methods

Traditional, heavyweight, software development models are rigid and heavily documentation and process-oriented. Many of these models are difficult to follow in the current software development environment. A new generation of lightweight methodologies has evolved in response to this. These lightweight models insist on far less documentation and specify only a few simple rules that need to be adopted.

Agile software development methods are characterised by nimbleness to rapid changes, multiple incremental iterations and a fast development pace. Existing agile methods include Feature Driven Development (FDD), Internet-speed or short-cycle time systems development and Extreme Programming (XP). Siponen, Baskerville and Kuivalainen (2005) specify four requirements for security methods that are targeted to be integrated into agile software methods, namely:

- Must be adaptive to agile software development methods;
- Must be simple and not hinder the development project;
- Should offer concrete guidance and tools at all phases of development, i.e., from requirements gathering to implementation and testing;
- Should be able to adapt rapidly to ever-changing requirements owing to a fast-paced business environment, including support for handling several incremental iterations.

Siponen et al (2005) describe how software developers can use agile software development methods to build secure information systems. Developers, for competitive reasons, often use these methods for web and network applications where security risks are prominent. Agile development methods, however, have few explicit security features. One reason for this may stem from the misconception that security hinders development. Siponen et al (2005) further suggest that to seamlessly integrate security into agile software development methods, security techniques need to be adaptable and agile to operate in changing conditions.

Traditional development approaches create much documentation. Such records serve as useful artefacts for communication and the traceability of design. Agile methodologies, on the other hand, encourage lean thinking and cutting down on overheads, particularly documentation. They are ideal for projects that exhibit high variability in tasks because of changing requirements, in the capabilities of people, and in the technology being used (Nerur, Mahapatra & Mangalara, 2005).

4.3.8 Extreme programming

Extreme programming is one of the earliest and most important agile methodologies. It is a relatively new concept, but is in many ways an extension of the earlier work in prototyping and RAD. The main premise of XP is that the SDLC and its many alternatives are too large and cumbersome. Many of them provide good control but they typically end up adding complexity, taking more time, and slowing down programmers. XP simplifies the development process by focusing on small releases, similar to prototyping, that add value to the customer.

One new aspect to XP is paired programming, whereby two programmers work together constantly. This aspect has been seen by many as an inefficient use of resources (Post and Anderson, 2003). However, although it has been empirically found that pair programming increases productivity by up to 15% (Kuppuswami, Vivekanandan, Ramaswamy & Rodrigues, 2005). The core XP practices can be summarised as pair programming, simple design, test-driven development and design improvement. The inventors of XP claim that one of the most important benefits of XP is the reduction of the software development effort.

It is clear from the discussion in Section 4.3 that none of the software development models studied explicitly integrate security into their life cycle. This could be attributed to the fact that incorporating security is often viewed as something that decreases productivity and increases the cost of software development projects. Section 4.4 addresses the typical phases of software development, thereby providing an improved understanding of the SDLC.

4.4 Typical Phases of Software Development

An important feature of the SDLC is that it is a comprehensive method that encompasses the five basic phases of software development, as depicted in Figures 4.1 and 4.2, namely investigation, analysis, design, implementation and maintenance.

Figure 4.1 illustrates these phases as being applied in a sequential, linear manner, an iterative approach, as indicated in Figure 4.2, is more common. Since an iterative process is essentially circular in nature, each phase receives input from and provides output to another. At a high level this circularity ensures the reassessment of the quality of each artefact.

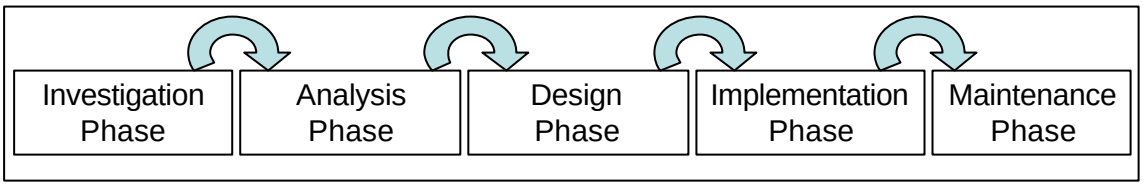

Figure 4.1: Typical Phases of Software Development

Virtually all systems development models incorporate these five phases. However, they do differ in how much time is spent in each phase, who does the work, and in the degree of formality involved (Post and Anderson, 2003).

It is important to realise that all of the activities involved are highly related and interdependent. Therefore, in practice, several developmental activities may occur at the same time (O'Brien, 2002). There are several variations of the SDLC, however, the five-step process described in this section represents the most common practice. It must be noted that some versions identify more than five phases, while others may use different terminology for individual phases.

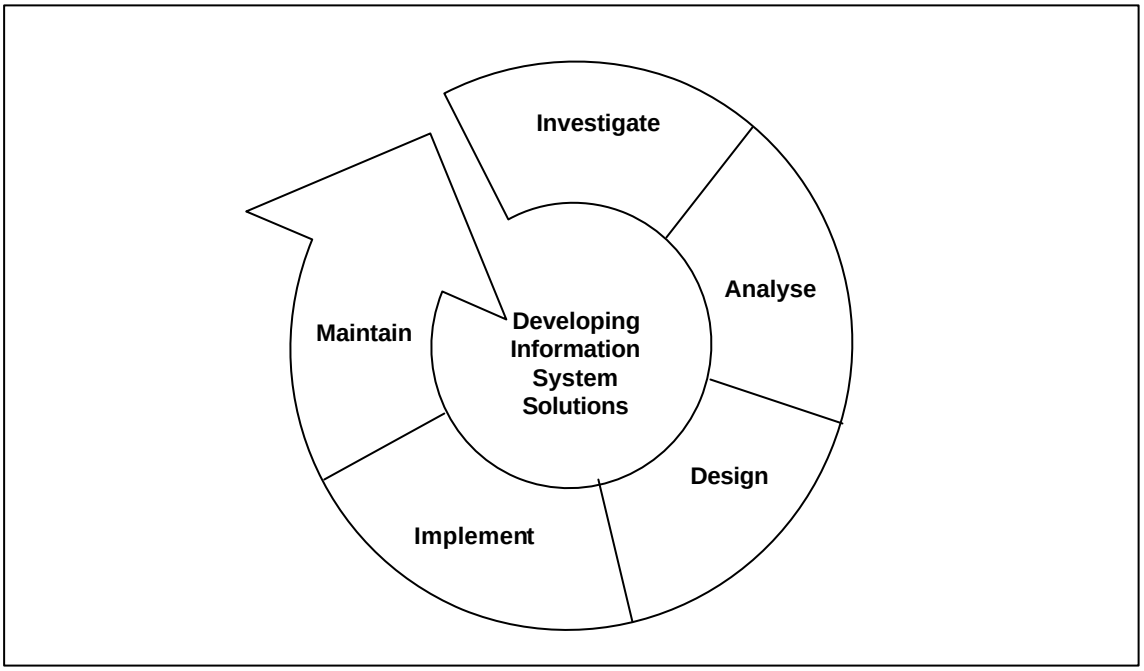

Figure 4.2: A Typical Cycle for Developing Information System Solutions (O'Brien, 2002)

4.4.1 The investigation phase

During the investigation phase, the objectives, constraints and scope of the project are specified. A preliminary cost-benefit analysis is performed to evaluate the perceived benefits and appropriate levels of cost for those benefits. It is during this phase that the objectives and general definitions of the requirements are established. There is an initial user definition activity followed by an evaluation and the initiation of necessary documents to formally commence the software development project. The documentation produced during this phase requires user involvement to define the project and its worth. Friedman and Wlosinski (2003) recommend that both a data sensitivity assessment and preliminary risk assessment take place during this initial phase. This phase concludes with a feasibility analysis which assesses the economic, technical and behavioural feasibilities of the project.

A feasibility study is a quick examination of the problems, goals, and expected costs of the system. The objective is to determine whether the problem can be solved with an information system. The problem may turn out to be more complex than originally thought or unsolvable with current technology. It may be better to wait for improved technology or lower prices (Post and Anderson, 2003).

4.4.2 The analysis phase

Several basic activities of systems analysis need to be performed whether developing a new application quickly or developing a long-term project. The analysis phase begins with a study of the documentation gained during the investigation stage. However, systems analysis is not a preliminary study, but an in-depth study of the end-user information needs. This phase produces a set of functional requirements that are used as the basis for the design of a new or improved information system. Assessments of the organisation, the status of the current systems and its capability to support the proposed systems are performed.

It is important during this phase to determine how the existing system works and where the problems are located. The common technique used is to decompose the system into smaller, workable chunks. Analysts begin to determine what the new system is expected to do, and how it will interact with existing systems. At the end of the analysis phase, a complete description of the business requirements should be available. The problems and needs are typically documented with text, data flow diagrams and other figures depending on the methodology followed (Post and Anderson, 2003).

4.4.3 The design phase

The systems analysis phase describes "what" a system should do to meet the information needs of users, whereas the systems design phase specifies "how" the system will accomplish this objective. Systems design consists of design activities that produce system specifications satisfying the functional requirements developed in the systems analysis phase (O'Brien, 2002).

The objective of systems design is to describe the new system as a collection of modules or sub-systems. This phase is typically divided into logical and physical design stages. The logical design is the blueprint for the desired solution. It is implementation independent, meaning that it contains no reference to specific technologies, vendors or products. The physical design, on the other hand, identifies specific technologies to support the desired solution. The system design phase will indicate how the new system will work by providing all the necessary details, including data inputs, system outputs, processing steps and database designs. The output of this stage consists of a complete technical specification of the new system (Post and Anderson, 2003).

4.4.4 The implementation phase

Once a new information system has been designed, it must be implemented. The implementation phase typically involves the acquisition of hardware and software, software development, testing of programs and procedures, development of documentation, and a variety of conversion alternatives. It involves the education and training of end users and specialists who will operate the new system (O'Brien, 2002).

Implementation is often a difficult and time-consuming process. However, O'Brien (2002) argues that it is vital in ensuring the success of any newly developed system, since even a very well-designed software application will fail if it is not properly implemented.

During the implementation phase, any required software components are developed or purchased. Once all components are tested individually, they are installed, integrated and tested as a system. Post and Anderson (2003) argue that although testing and quality control must be performed at every stage of development, a final systems test is required before entrusting the data of the company to the new system. A crucial stage in implementation is, therefore, final testing.

4.4.5 The maintenance phase

Once a system is fully implemented and is being used in business operations, the maintenance phase begins. Systems maintenance may be defined as the monitoring, evaluating and modifying of operational information systems to make desirable or necessary improvements (O'Brien, 2002).

The maintenance and review phase is the longest and typically most expensive phase of any software development life cycle. The needs of the organisation change, therefore, the systems that support the organisation must change. The pressures for change are so great in most organisations today that as much as 80% of the Management Information System (MIS) staff is devoted to modifying existing programs. These changes are most often time-consuming and difficult. When a programmer makes a minor change in one area, it may affect another area of the program, which in turn can cause additional errors or necessitate more changes (Post and Anderson, 2003).

4.5 Software Quality in the SDLC

Society has currently become increasingly dependant on technology, therefore, software applications must work. It is unsurprising that many software characteristics continue to grow in importance, including reliability, availability, security, safety and quality.

Software quality is a process, not a product. According to McGraw (2003), there is no substitute for instilling software quality as deeply into the software development process as possible, taking advantage of the engineering lessons software practitioners have learned over the years. He stresses that the particular software process followed is not as important as the act of thinking about reliability, security and performance throughout the software development process.

McGraw (2003) refers to three technical trends that are aggravating the software quality problem and making the impacts on business both more common and more serious than previous. He summarises these three key trends as follows:

- All modern software is exposed to the Internet;
- Extensible systems (for example, Java and .Net) are dangerous;
- System complexity is rising.

Quality is a key issue within the software industry. Software is expected to perform at high levels of reliability and security. The failure of such software can result in severe business consequences including (McGraw, 2003):

- Revenue loss when software fails or key data is stolen or compromised;
- Brand/reputation damage can destroy the market impact when software does not work as advertised, or security vulnerabilities impact consumer trust;
- Liability costs when consumers cannot complete online transactions, or when software embedded in airplanes, automobiles, pacemakers or nuclear reactors causes injury or death;
- Productivity loss when software malfunctions or ceases to function altogether.

Defective software costs the American economy an estimated $59.5 billion each year, according to a report issued in June 2002 by the NIST institute. Software users incurred 64% of the total and software developers 36%. The NIST institute suggests that improvements in testing could reduce this cost by about a third, but that testing improvements would not eliminate all software errors (McGraw, 2003).

McGraw (2003) suggests that the early identification of software risks is beneficial. The two primary reasons in support of this argument are:

- Most software defects are introduced early in the software life cycle;
- The earlier in the software life cycle a defect is uncovered and fixed, the less it costs to repair.

These reasons are illustrated in Figure 4.3 and 4.4 respectively.

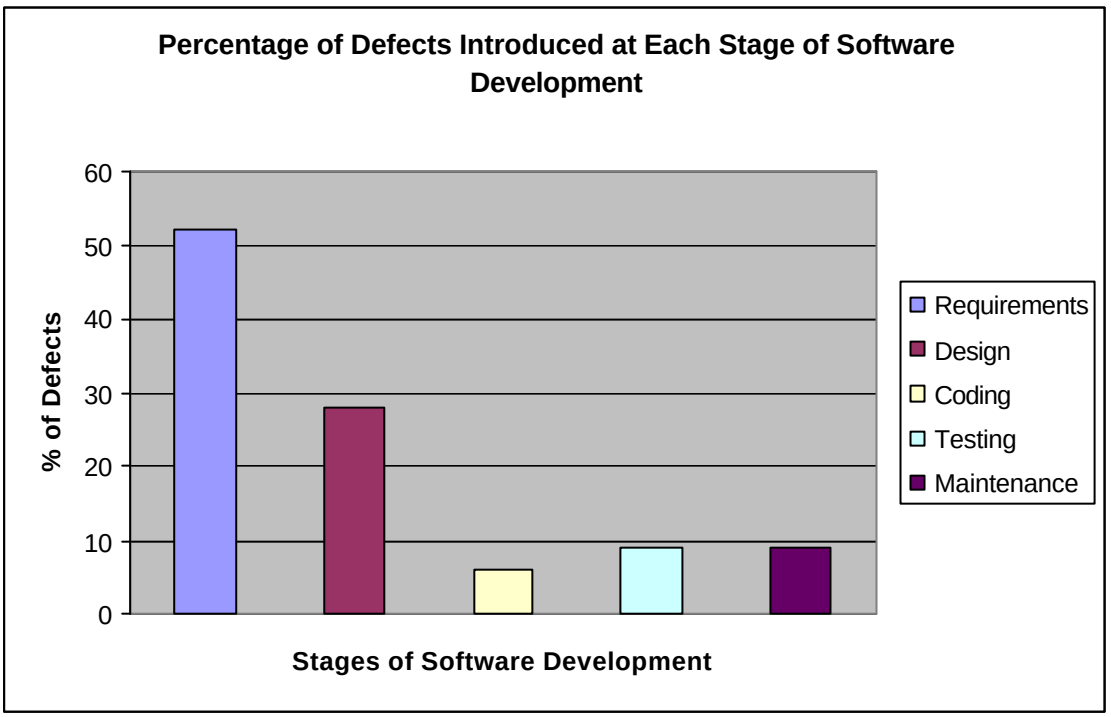

Figure 4.3: Percentage of Defects Introduced at Each Stage of Software Development (McGraw, 2003)

Figure 4.4 illustrates that it is evident that errors revealed in the initial stages of the software development process are less costly to fix than those revealed in the later stages. The likelihood of errors occurring in the later stages tends to diminish in proportion to the effort devoted to the initial stages. Figure 4.4 illustrates that it is evident that the cost of repairing errors is significantly lower during the requirements and design stages then in the later stages, or during the coding, testing and maintenance phases.

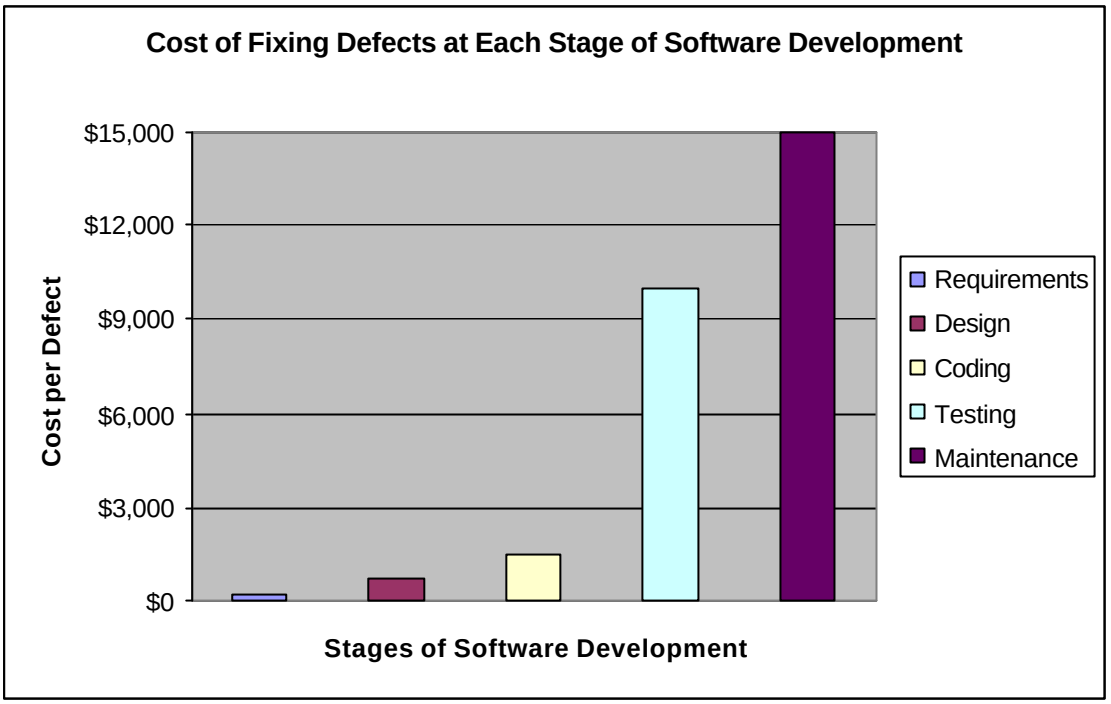

Figure 4.4: Cost of Fixing Defects at Each Stage of Software Development (McGraw, 2003)

One of the areas of software where errors can least be tolerated is that of security safeguards. One of the techniques employed to improve the reliability of software is Software Quality Assurance (SQA). The SQA process attempts to improve the quality of computer software products via the software development process. SQA involves the use of various reviews and the establishment of baselines throughout the SDLC. Tompkins and Rice, as early as 1985, argued that it is not sufficient to simply incorporate security safeguards in application systems. Safeguards should possess many of the software quality factors. Security concerns should be integrated into the SQA process in the same manner that security concerns should be incorporated into the SDLC process. The quality of security safeguards must be built-in, like software quality, and cannot be tested-in (Tompkins and Rice, 1985).

Businesses that value quality, according to Bessin (2004), become more effective at innovation, increase their competitive differentiation and greatly reduce their total cost of development and ownership. Quality, however, is not only the responsibility of each individual on the development team, but is the responsibility of the team as a whole. Teams must do everything they can to integrate workflows, establish traceability and simplify communication. A breakdown in the chain linking team members leads to data

loss, rework, lack of clarity and inefficiency and ultimately lowers software quality (Bessin, 2004). Gong, Yen and Chou (1998) suggest that software developers should select an appropriate development methodology and use suitable tools to maintain a high quality and low-cost software design and development environment.

Gong et al (1998) refer to the growing trend of adopting the Total Quality Management (TQM) philosophy to software development. They suggest that applying TQM to the software development process will help control software quality and productivity, lower costs and decrease cycle times. They provide through their research a way of integrating TQM into the software development process, and in so doing have focussed on the common phases of the SDLC (requirements analysis, design, development, testing and maintenance). Some of the major characteristics of TQM they focus on include:

- *Understandability* – a software project should be well-structured, concise, complete and consistent and thereby be understood by analysers, designers, programmers and users;
- *Reliability* – quality software is reliable and dependable. A reliable system performs correctly, completely and consistently;
- *Testability* – quality software should be testable, simple and easy to execute and maintain;
- *Modifiability* – this implies that quality software is general and flexible. A system of generic modules or routines should be re-used or tailored for other applications;
- *Portability* – quality software should be hardware independent. It should demonstrate a consistent performance on different computers with minimal modifications;
- *Efficiency* – a system with short turnaround time, faster response time and better throughput is recognised as the one with better quality;
- *Usability* – simplicity or conciseness is one of the fundamental perceptions about good quality.

Table 4.1 identifies the relationship of software quality factors to the life cycle phases in terms of where quality factors should be measured, and where the impact of poor quality is realised. Tompkins and Rice (1985) have not explicitly stated security as a

software quality factor, but this is implied through the correctness, reliability and integrity factors. They further suggest that software quality factors should be included in the functional requirements document and ultimately viewed as performance criteria (Tompkins and Rice, 1985).

SOFTWARE QUALITY FACTORS	SDLC					
	Requirements Analysis	Design	Code and Debug	System Test	Operation	Maintenance
Correctness (*)	#	#	#	X	X	X
Reliability (*)	#	#	#	X	X	X
Efficiency		#	#		X	
Integrity (*)	#	#	#		X	
Usability	#	#		X	X	X
Maintainability		#	#			X
Testability		#	#			X
Flexibility		#	#	X		X
Portability		#	#			
Reusability		#	#			
Interoperability		#			X	
LEGEND	# Where software quality factors should be measured; X Where impact of poor quality is realised * Security-related factors					

Table 4.1: Relationship of Software Quality Factors to Life Cycle Phases (Tompkins and Rice, 1985).

The business benefits of including quality-oriented activities in all phases of the software development cycle are both broad and deep. These measures facilitate innovation and lower costs by increasing predictability, reducing risk and eliminating rework. They can help differentiate a business from its competitors. Most important, continuously ensuring quality will always cost less than ignoring quality considerations. It is actually suggested that raising product quality costs virtually nothing if done correctly (Bessin, 2004).

4.6 Security in the SDLC

Tompkins and Rice (1985), as far back as 1985, found that inadequacies in the design and operation of computer applications were a frequent source of security vulnerabilities associated with information systems. This lead them to state that "security concerns should be an integral part of the planning, development and operation of a software application". Furthermore, they suggested that the SDLC methodology provides the structure to ensure that security safeguards are planned, designed, developed and tested in a manner that is consistent with the sensitivity of the information.

Tompkins and Rice (1985), found in the research that much of what needs to be done to improve security is not clearly separable from what is needed to improve the usefulness, reliability, effectiveness and efficiency of computer applications. However, they stress that while security concerns should be integrated into the SDLC, steps should be taken to ensure that the appropriateness, adequacy and reasonableness of security safeguards be separately identifiable activities within each stage of the SDLC. This means that system planners, developers and users should accomplish a series of security-related actions throughout the SDLC. The process for incorporating security safeguards within an application, however, is not substantially different from the SDLC activities.

Similarly, Jones and Rastogi (2004) state that to meet future demands, opportunities and threats associated with information security need to be "baked in" to the overall SDLC process. The reality is that information security is an afterthought for many organisations. This means that, most often, security is not an integral part of their business or information strategies, nor is it woven into their IT projects.

Jones and Rastogi (2004) suggest various reasons for this neglect, including:

- Security is not considered a business enabler or revenue generator. Its value is not appreciated until risks manifest themselves through attacks and compromises, by which time it is too late;
- In the absence of proper support from top management, security flounders and has few champions in an organisation;

- Developers are usually hired for their development or coding expertise and have minimal security knowledge. They are not given the training, tools, resources, time or the motivation to build secure systems;
- Software projects inevitably run up against deadlines, wherein companies focus their primary effort on the features of the software rather than on its security;
- Security is often perceived as a barrier to functionality, adding constraints and reducing flexibility;
- Security is often treated as a collection of tasks that can be completed at the end of the project, rather than as a process or a mindset employed throughout the project.

The cost of these lapses, regardless of the reasons behind them, can be high. For example, Microsoft experienced the effects of neglecting security in 2003 when security holes in their products were exploited, resulting in billions of dollars in recovery costs and lost productivity.

Jones and Rastogi (2004) are concerned that traditional firewall systems have become less effective in preventing or detecting Web-based attacks. They suggest that central to many successful system attacks currently are poorly developed systems and applications. Many of the security properties that are repeatedly outlined in government and other regulations, including accountability, unique user accounts and confidentiality, can be circumvented when software developers have not paid enough attention to security in the design, development, deployment and maintenance of their products. They argue that if the security considerations for systems were woven into the SDLC, and if the developers, project managers, and system architects were given adequate training, many of the security vulnerabilities that manifest themselves in software applications would never appear.

Security in application software is usually implemented in a form of so-called security services which are developed without considering the security requirements of the application system. Booysen and Eloff (1995) argue that security requirements are usually added to the system after development, because it is believed that the final system will suffer from:

- Loss of performance with the addition of security features;

- Loss of flexibility owing to restrictions and confinements in the behaviour of the target system;
- Higher costs to account for analysis of the security requirements, design and implementation of the security specifications, and maintenance of security in the application system.

Security plays an increasingly important role within systems' development. This can be attributed to the increase in the number of distributed applications. Breu, Burger, Hafner and Popp (2004) argue that security is a requirement that has to be considered at all stages of development and which needs particular modelling techniques to be captured. The development of secure systems clearly poses particular challenges to the development process. According to Breu et al (2004), these challenges include:

- The separation of security requirements and security measures;
- The traceability of security requirements throughout the development life cycle;
- The correctness and applicability of the security measures taken;
- The completeness of security requirements and measures.

Breu et al (2004) suggest that these challenges may be met by:

- Specifying the security requirements in context;
- Gathering potential threats related to the security requirements;
- Estimating the occurrence of every threat and its potential harm either quantitatively or qualitatively;
- Designing appropriate measures, taking into account the results of the risk analysis;
- Checking the chosen measures against the specified requirements.

The steps of threats and risk analysis support the transition from requirements to measures by gathering the potential threats related to the security requirements and by estimating the occurrence of each threat and its potential harm.

The NIST institute released a publication entitled *"Security Considerations in the Information System Life Cycle"* SP 800-64 (2004) which is in line with much of the research that has been conducted in this area. This document offers a framework for incorporating security considerations throughout a generic SDLC, including a "minimum" set of security considerations that need to be considered within each of the

five phases, namely the design, development, testing, operations and maintenance and disposal phases (Jones and Rastogi, 2004). The NIST institute states that building more secure systems requires:

- Well-defined security requirements and security specifications;
- Well-designed component products;
- Sound systems security engineering practices;
- Competent systems security engineers;
- Appropriate metrics for product/system testing, evaluation and assessment;
- Comprehensive system security planning and life cycle management.

Various classes of security-related risks and threats, according to Wang and Wang (2003), need to be considered during the design phase of the software development process. They divide the universe of software security risks and threats into three categories based on their target of attack, namely:

- *Application layer* – a class of security risks that focuses on attacking application software itself;
- *Platform layer* – include all risks and attacks that focus on the underlying platform or operating layer, such as attempts to gain unauthorised administrator access on Unix or Windows-based systems;
- *Network layer* – these threats and risks generally deal with the underlying telecommunication and network elements such as routers, switches and gateways.

It is important to software architects and designers that within any particular software project one or more of the risks may not be applicable to the particular application domain, hence their impact on the quality factors would need to be discounted due to their low likelihood of occurrence (Wang and Wang, 2003).

The operational software quality factors, from a software design point of view, are threatened by security risks as illustrated in Table 4.2. By taking into consideration security risks and threats, and their impact on the quality of the target system, software architects and designers need to select protection mechanisms via the application of appropriate security technologies and approaches to provide necessary safeguards.

LAYER	SECURITY RISKS	SOFTWARE QUALITY FACTORS				
		Correctness	**Reliability**	**Efficiency**	**Integrity**	**Usability**
Application layer	Credential theft				**	
	Functional manipulation	**	*		*	*
	Data theft/manipulation	*	*		**	*
	Application DoS		**	*	*	
Platform layer	Unauthorised admin. Access				**	
	System DoS		**	*		*
	Application Modification	**	*	*	*	
Network layer	Network DoS		**	*		*
	Network exposure /manipulation	*	**	*	**	
	Network Credential Theft				**	
LEGEND		* = negative effect, no irreparable damage; **=strong negative effect, causing irreparable harm				

Table 4.2: Individual Security Risks and Their Impact on Software Quality Factors (Wang and Wang, 2003)

Gregory (2003) argues that the failure of an organisation to involve information security in the life cycle will result in events that will be more costly and disruptive in the future. A wide variety of things can happen to information systems that lack the required interfaces and characteristics. Some example events include:

- *Orphan user accounts* – exist because the information system does not integrate with an identity management or single sign-on of an organisation;
- *Defaced web sites* – occur because systems were not built to security standards and instead include easily-exploited weaknesses;
- *Fraudulent transactions* – occur because an application lacked adequate audit trails and/or the processes required to ensure they are examined and the identified issues dealt with.

These problems are more costly to solve than the extra effort required to build, from a software design point of view, build the application correctly in the first place. Gregory (2003) further suggests that the proper use of the SDLC will contribute significant value in the form of software applications that are secure by design.

4.7 Conclusion

Many organisations use some variation of the SDLC to ensure that a carefully planned and repeatable process is used to develop software, however, most do not take security adequately into consideration. This results in the production of insecure systems (Tipton and Krause, 2006).

The SDLC begins with the identification of a requirement for software and ends with the formal verification of the developed software against that requirement. The primary purpose of the SDLC is to provide guidance and control over the development process. Security, as with quality, should be viewed as an integral part of the SDLC and not merely as an add-on or after-thought at the end of the development process. Most organisations have created their own customised versions of the SDLC. The details may vary from organisation to organisation, they all tend towards a common ultimate goal.

Researchers argue that incorporating security within software development processes is the most appropriate way of introducing security in the complex context of an organisation. The current literature fails so far to present a comprehensive framework that can integrate security with development, despite the abundance of software development approaches that have emerged. Many of them deal with few of the emerging aspects or have not yet been properly evaluated in the real world. Producing quality software requires personnel with substantial education, training and experience in both software development and information security.

None of the methodologies studied actually assist software developers in designing and building secure software applications. Small changes in the SDLC could substantially improve security without incurring significant overhead. There is, therefore, the need for a model or framework that attempts to guide the development team through the complete and detailed secure SDLC. It is important to get security right during the requirements, analysis, design and development phases. Research has shown that it takes ten times more time, money and effort to fix a bug in the development phase than in the design phase. The lesson is, therefore, to identify the security requirements and designs correctly as early in the SDLC as possible.

The security requirements associated with the development of an application system should be considered during the definition of user requirements and incorporated into the system during the design stages. Security must be at the heart of software specification, design and implementation to have meaningful, long-term impact. An expanded software development methodology is needed to provide an approach towards secure software development that addresses security requirements as part of the SDLC, while considering other functional requirements. A thorough risk analysis needs to be performed to determine the security requirements of a software application. This is the topic of the following chapter.

Chapter 5

Risk Analysis

5.1 Introduction

The previous chapter introduced the SDLC, including a number of alternative software development models and the typical phases of the SDLC. This chapter discusses risk analysis as a key process for the development of secure software applications.

Organisations today must deal with a multitude of information security risks. Terrorist attacks, fires, floods and other disasters can compromise or even destroy their information assets. The theft of trade secrets and the loss of information due to unexpected computer shutdowns can cause businesses to lose their competitive advantage. The achieving and maintaining of the trust of their customers must be foremost in the strategies of most organisations, especially those that do business online. Successful organisations typically build this trust by establishing safe and secure operating environments.

Security vulnerabilities in software applications render an otherwise secure environment insecure. Any software application, which has exploitable vulnerabilities, when added to a secure environment, affects the security of the total environment. Therefore, it is critical that application software be free from security vulnerabilities, especially in a networked environment. Security vulnerabilities in software arise from a number of development factors, but can generally be traced back to poor software development practices, new modes of attack, misconfigurations and unsecured links between systems. It has been said that the most secure computer system is one disconnected from the network, locked in a room and with its keyboard removed. However, such a machine is useless. Since perfect security is an unattainable goal with current technologies, producing secure software becomes a question of risk management.

An important part of the information security management process is the assessment of risks, and how they can be reduced to acceptable levels. It is necessary to take into account the specific needs and risks each information system, and to align these with the

business objectives and organisational and environmental aspects. Security risks must be identified, ranked, and managed throughout the SDLC. McGraw and Verdon (2004) support this notion by stressing the importance of integrating a high-level approach to iterative risk analysis throughout the SDLC. This, they argue, will help overcome many of the security vulnerabilities typically found in software applications today.

This chapter addresses some of the main approaches to risk analysis, and a detailed discussion of the various stages of risk analysis. It introduces some fundamental security risk concepts and their relationships. These are described in Section 5.2.

5.2 Security Risk Concepts and Relationships

Risk can be described as a threat that exploits some vulnerabilities that could cause harm to an asset. Peltier (2005) defines risk management as "the total cost to identify, control and minimise the impact of uncertain events". According to Spinellis, Kokolakis and Gritzalis (1999), managing risk can follow three strategies: risk reduction, risk transfer and risk acceptance. Risk acceptance means that there is an awareness of the risk, but there is a preference to accept its consequences instead of applying any form of countermeasure. This will apply in cases where the cost of the countermeasure is significantly higher than the impact of a security breach. Insurance is an example of risk transfer, where the risk is typically transferred to a third-party. This will apply in cases where the probability of the security breach is low, but the potential impact may be very high. Risk reduction can be achieved by reducing the probability of the threat, reducing the vulnerability, reducing the impact or effective recovering from the threat occurrences. Peltier (2005) advocates that effective risk management be totally integrated into the SDLC. This requires, however, that a risk analysis be performed during the initial stages of the SDLC.

The terms risk analysis and risk management are often used incorrectly. However, in the literature, there are two main schools of thought. The first recognises these as two separate processes, with risk management following risk analysis. The second school of thought reasons that risk management includes risk analysis, and the introduction of security controls. This research adopts the former school of thought. However, it does recognise risk assessment as an essential part of risk analysis and not as a separate

process. The terms controls, countermeasures and safeguards are used interchangeably throughout this work.

Risk analysis is defined by Peltier (2005) as *"a technique to identify and assess factors that may jeopardise the success of a project or achievement of a goal"*. The main objective of risk analysis is to identify and assess the risks to which the information systems and its assets are exposed, to select appropriate and justified security safeguards (Spinellis et al, 1999). It must be noted, however, that a risk analysis does not reduce risks. It merely identifies and assesses risk. The measuring of risk is not a simple task, since analysts are typically forced to estimate or predict future events, which are uncertain. Rigorous risk analysis relies heavily on an understanding of business impacts which, in turn, requires an understanding of laws and regulations, and the business to be supported (McGraw and Verdon, 2004). They further suggest that putting the right people together for an analysis is important. Risk analysis is not a science and knowledge and experience cannot be overemphasized. A broad knowledge of vulnerabilities, flaws and threats is a critical success factor. An important stage of the risk analysis process is the assessment of risk.

A risk assessment may be broadly defined as "the computation of risk". The risk algorithm computes risk as a function of assets, threats and vulnerabilities. According to NIST SP 800-64 (2004), a preliminary risk assessment should result in a brief, initial description of the basic security needs of the system. In practice, this need for information security protection is expressed in terms of the need for integrity, availability and confidentiality and other security needs that are applicable, for example, accountability and non-repudiation. A preliminary risk assessment should define the threat environment in which the product or system will operate.

With an effective risk assessment process in place, only those controls and safeguards that are actually needed will be implemented (Peltier, 2005). These controls and safeguards should be relative to the security requirements of the information system in question. Security requirements are concerned with the extent to which assets are to be protected from harm.

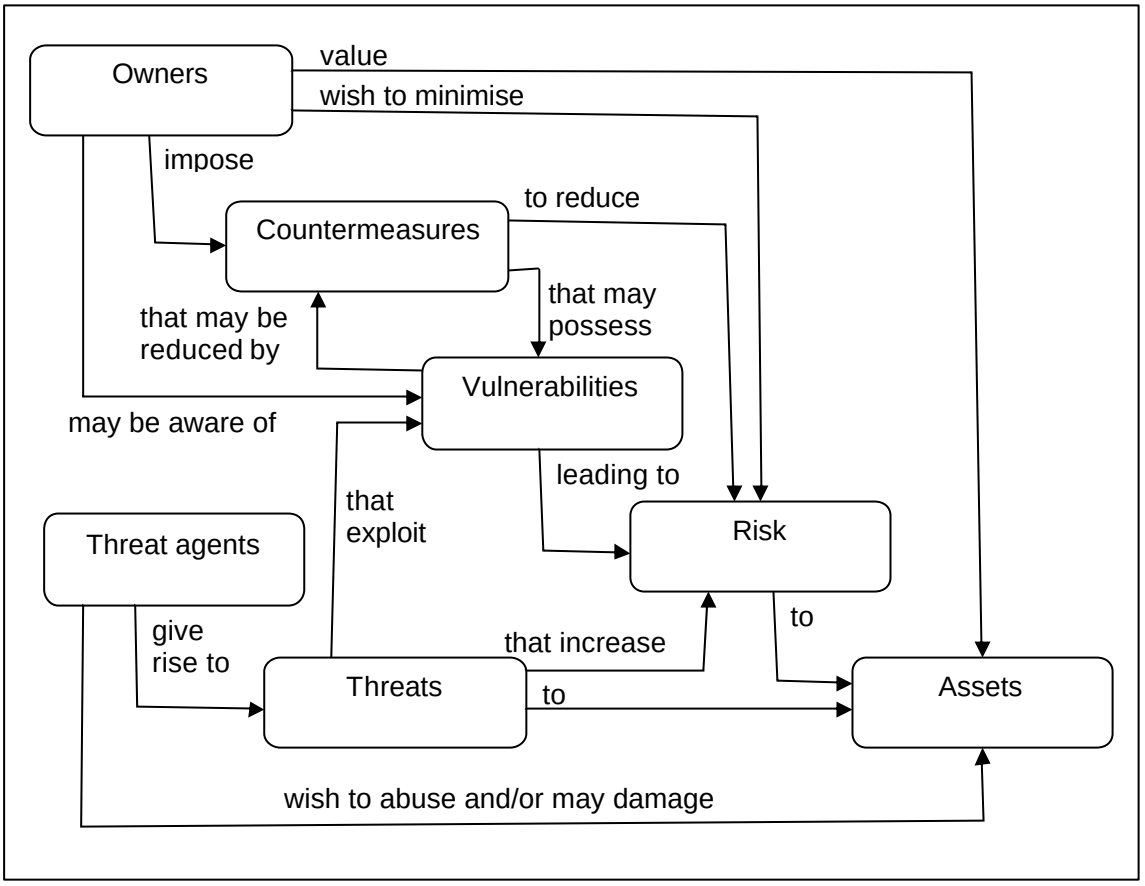

Figure 5.1: Security Concepts and Relationships (source : Common Criteria) cited in Task Force Report, 2004

Figure 5.1 provides a graphical view of some of the key security concepts and relationships. This diagram illustrates that information owners value their information assets. Therefore they wish to minimise any potential risk to them. Information owners would typically impose some kind of countermeasure or safeguard to reduce such risk. These countermeasures, however, may themselves possess some form of vulnerability, which may lead to further risk. This risk can be increased by threats that exploit existing vulnerabilities. Threat agents may possess certain capabilities and intentions creating threats. Threats utilise vulnerabilities in the system to focus their attacks. Adversaries use specific kinds of attacks or "exploits" to take advantage of particular vulnerabilities in the system. Without vulnerability, threat frequencies are immaterial, because there is no loss. Threats, impacts and vulnerabilities are all implicated in annualised exposures, therefore safeguards can lower risk by preventing, reducing or mitigating their effects (Anderson, 1991).

It is clear, from this discussion, that an important part of the risk analysis process is to determine what threats exist to a specific asset and the associated likelihood or probability of each threat. These threats must, therefore, be prioritised. Possible safeguards and controls must be selected as part of the risk management process. It is unwise to implement controls or safeguards simply because they seem to be the right thing to do. Each information system operates within a unique threat environment. It is recognised that information security controls must be determined, based on the risk to the system in the given environment (Landoll, 2006). Section 5.3 discusses various approaches to risk analysis.

5.3 Risk Analysis Approaches

There are a number of distinct approaches to risk analysis. However, these can be divided into two types: quantitative and qualitative, as discussed in Sections 5.3.1 and 5.3.2 respectively. Both quantitative and qualitative risk analysis methods are supported by standards and guides, like the Common Criteria Framework, ISO/IEC TR 13335, ISO/IEC 17799 and NIST 800 Special Publications.

5.3.1 Quantitative risk analysis

Quantitative risk analysis methods use mathematical and statistical tools to represent risk. For example, the Annual Loss Expectancy (ALE) or the Estimated Annual Cost (EAC) calculations require input in the form of numeric frequencies and monetary loss estimates. Quantitative risk analysis makes use of a single figure produced from these elements. This is calculated for an event by simply multiplying the potential loss by its probability. The equation is: **ALE = SLE × ARO**, where SLE is the Single Loss Expectancy, and ARO is the Annualised Rate of Occurrence (or the predicted frequency of a loss event happening). Therefore, it is theoretically possible to rank events in order of the calculated risk (ALE) and to make decisions based upon this (Security Risk Analysis Group, 2003).

The ALE-based methodologies are typically asset-driven. An asset-driven methodology directs most effort at compiling an inventory of data sources, destinations and modes of use. It commonly uses quantitative techniques and usually includes large-scale surveys.

Impact losses on business functions are revealed by modelling all possible paths from threats to assets via the vulnerabilities (Anderson, 1991).

The main problem with quantitative risk analysis is usually associated with the unreliability and inaccuracy of the data, since probability can rarely be precise. Despite this drawback, however, a number of organisations have successfully adopted quantitative risk analysis (Security Risk Analysis Group, 2003). Karabacak and Sogukpinar (2005), however, argue that risk analysis methods that use intensive quantitative measures are not suitable for the current information security environment, since these methods are not able to model such complex risk scenarios. They believe that risk analysis methods based on qualitative measures are far more suitable.

5.3.2 Qualitative risk analysis

This is by far the most widely used approach to risk analysis. Probability data is not required, and only estimated potential loss is used. Qualitative analysis allows information users and owners to avoid committing themselves to unqualified point values, by using relative rankings or heuristic estimates. Users often feel more comfortable with qualitative output (Anderson, 1991). An important drawback of this approach, however, is that it tends to yield inconsistent results. These results are usually dependent on the ideas of the people who conduct the risk analysis (Karabacak and Sogukpinar, 2005). Qualitative analysis does not remove any uncertainty from the model, it merely distinguishes between opinion and fact. Its practical effect is to streamline asset evaluation and threat analysis (Anderson, 1991).

5.3.3 Checklist-based approaches

The checklist method, also known as the simple questionnaire method, uses a series of questions to assess risk. There are a number of sources that provide security checklists, such as manuals from computer system vendors and publications from security organisations. Examples include BS7799 parts 1 and 2 (1999), ISO/IEC TR 13335 Part 4 (2000), IT-Baseline Protection Manual (GISA, 1997) and the NIST Handbook (1995). These generally list the questions and checklists by either functional areas, such as input, processing or output, or asset types, such as hardware, software and personnel. These generic checklists need to be converted to specific questions tailored for risk

analysis. The advantage of the checklist method is its simplicity in identifying major weaknesses.

Cho and Ciechanowicz (2001) use an evidential network to combine answers and uncertainties from a checklist-based risk analysis. They argue that the checklist approach is still useful today in that it is relatively easier and simpler than other risk analysis methods. The checklist method is a useful method in that it provides an overview of the security of the system in a reasonably short time period. It is the only applicable method to use where there is no risk analysis expertise nor organisational resource, such as budget and time, available to perform a detailed risk analysis. One concern in the checklist method, however, is how to manipulate the gathered answers to highlight areas that need management attention. The output of a checklist-based risk analysis without this capability, tends to be a lengthy list of answers to questions, which has very limited use to management and prevents quick decisions being taken to improve security (Cho and Ciechanowicz, 2001).

Checklists are used to some extent in the SecSDM, described in Chapter 6, thus it is necessary to highlight a number of issues and limitations with regards to their use. Landoll (2006) specifically refers to the following guidelines regarding the use of checklists:

- Checklists are a memory aid and should be used as a guide and a reminder to provide a complete and accurate analysis;
- Checklists help to ensure accuracy and completeness and can be used to simplify and improve various processes;
- Checklists can, however, drive the results instead of guiding the analyst if not used correctly;
- Checklists can be relied upon to the detriment of creativity;
- An over reliance on checklists can lead to tunnel vision and a breakdown in the analytical process necessary for effective risk assessment.

Section 5.4 briefly describes the specific risk analysis strategies, as advocated by ISO/IEC TR 13335-3 (1998).

5.4 Risk Analysis Strategies

The ISO/IEC TR 13335-3 (1998), from the viewpoint of level of detail and granularity, refers to the following four basic options for a risk analysis strategy:

- *Baseline approach*: a standard set of safeguards is applied to all information systems to achieve a baseline level of protection;
- *Informal approach*: a pragmatic risk analysis is conducted on all systems by exploiting the knowledge and experience of security professionals;
- *Detailed risk analysis*: refers to the detailed review of systems, including the identification and valuation of assets, an assessment of the levels of threats to those assets and associated vulnerabilities;
- *Combined approach*: the baseline and detailed approaches are balanced by applying detailed risk analysis to important systems, while protecting less important systems with a baseline approach.

The objective of baseline protection, according to ISO/IEC TR 13335-3 (1998), is to establish a minimum set of safeguards to protect all the information assets of an organisation. The appropriate baseline protection can be achieved through the use of safeguard catalogues. These suggest a set of safeguards to protect an information system from the most common threats, therefore a detailed assessment of threats, vulnerabilities and risks is not necessary.

A detailed risk analysis is done to identify the potential adverse business impacts of unwanted events, and the likelihood of their occurrence. The likelihood of occurrence is dependent on how attractive the asset is to a potential attacker, the likelihood of the threats occurring, and the ease with which the vulnerabilities can be exploited. A detailed risk analysis involves the identification and valuation of assets, the assessment of threats to those assets, and an assessment of the associated vulnerabilities. Its results lead to the identification, assessment and prioritisation of risk that are used to identify and select safeguards. This can be used to reduce the identified risks to an acceptable level (ISO/IEC TR 13335-3,1998).

The major advantage of a detailed risk analysis is that an extensive analysis is conducted for the specific situation, taking all threats, vulnerabilities and impacts into

account. A disadvantage is that it normally requires a considerable amount of time, effort and expertise to obtain the detailed results.

5.5 Risk Analysis Methodologies

The established risk analysis methodologies possess distinct advantages and disadvantages, and most share both some good principles and limitations, when applied to modern software design (McGraw and Verdon, 2004). This section briefly describes two of the more common risk analysis methodologies, namely CRAMM and OCTAVE.

5.5.1 CRAMM

Owing to the critical role of risk analysis in security management, a number of risk analysis methods have been developed since the early eighties. For example the Central Computer and Telecommunications Agency of the British government (CCTA) developed the CCTA Risk Analysis and Management Method (CRAMM) in 1985. Spinellis et al (1999) specifically advocate the use of CRAMM, because it is considered an effective and reliable method that has been extensively used since 1987. It is the mandatory risk analysis method for governmental organisations in the United Kingdom and has, therefore, been thoroughly tested. The CRAMM methodology involves three stages, namely:

- Asset identification and assessment;
- Threat and vulnerability identification and assessment;
- Countermeasure selection.

5.5.2 OCTAVE

This methodology was developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University. The Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE) method provides a process complete with guidelines, checklists, time-estimates and process descriptions for its three-phased process. The three phases include (Landoll, 2006):

- Asset-based threat profiles;
- Infrastructure vulnerability identification;
- Security and strategy plan development.

Sections 5.3, 5.4 and 5.5 discussed the various risk analysis approaches, strategies and methodologies respectively. Before explaining the risk analysis process in detail, it is necessary to address some of the common problems in risk analysis, as highlighted in Section 5.6.

5.6 Common Problems in Risk Analysis

Risk analysis is regarded as the point where the most difficulty arises within risk management. The measurement of risk is not a simple task because, it invariably includes the subjective judgment of an analyst. Risk analysis often forces the analyst to estimate or predict future events, which are uncertain. It is important, therefore, to consider these uncertainties associated with judgments made by the analyst.

Research has shown that many of the problems in risk analysis derive from an inadequate analysis of the elements of risk. Generally, surveys on risk analysis practice produce a pattern of complaints, including (Anderson, 1991):

- Formal risk analyses are time-consuming and expensive exercises and of dubious benefit;
- Methodologies always need to be adapted to fit a particular model, i.e., models are inappropriate;
- Quantitative methods are unsuitable for general data processing and communications analysis;
- Where quantitative methods are suitable, they are best conducted by employees with expertise in the application rather than in computer security;
- Risk analysers would be best served by tools that allow informal, qualitative analyses to be executed rapidly to compare several scenarios at a high level;
- The task of data collection is unacceptably time-consuming and represents a heavy investment in resources for the organisation;
- Risk analysis has to be performed by experienced IT staff, trained in the methodology and at least knowledgeable about computer security;
- System specification and documentation tools do not currently produce the "information about information" in an appropriate form for risk analysis;
- Most methodologies do not attempt to model the human environment of the system, although people pose the greatest risks;

- The notion of risk is applied at too high a granularity (entities like “file” and “data set” are too vague).

Cho and Ciechanowicz (2001) suggest that imprecise inputs, too much focus on numerical values, and the tendency to use the same inputs over several years, are some of the factors that bring into doubt the value of risk analysis. However, despite all the problems mentioned, risk analysis is critical for preserving security, and the benefits of a well-performed risk analysis far outweigh any drawbacks (Cho and Ciechanowicz, 2001).

For the purposes of this study, it is necessary to reiterate the distinction between risk analysis and risk management. Whereas risk analysis is concerned with the identification, assessment and prioritisation of risk, risk management focuses on the selection and implementation of security safeguards. Bearing in mind that a risk is directly related to a threat that has the ability to exploit a vulnerability, thereby impacting the associated asset. Section 5.7 describes the risk analysis process in detail.

5.7 The Risk Analysis Process

Risk analysis is a useful tool for organisations to identify possible security holes in information systems and assist in providing appropriate countermeasures. It is an essential tool for the systematic management of information security. It provides useful information to management by identifying the potential risks, and forms the basis for improved decision-making with respect to security investment. It is therefore, for any business or organisation, important to have a clear understanding of the assets that need to be protected, the threats against which those assets must be protected, the vulnerabilities associated with the assets, and the overall risk to the assets from those threats and vulnerabilities.

There are a number of possibilities to reduce risk. ISO/IEC TR 13335-3 (1998) specifically refers to the following alternatives:

- Avoid the risk;
- Transfer the risk (for example, insurance);
- Reduce the threats;

- Reduce the vulnerabilities;
- Reduce the possible impacts;
- Detect, react to, and recover from unwanted events.

Which of these possibilities, or combination of them, is most appropriate depends on the circumstances.

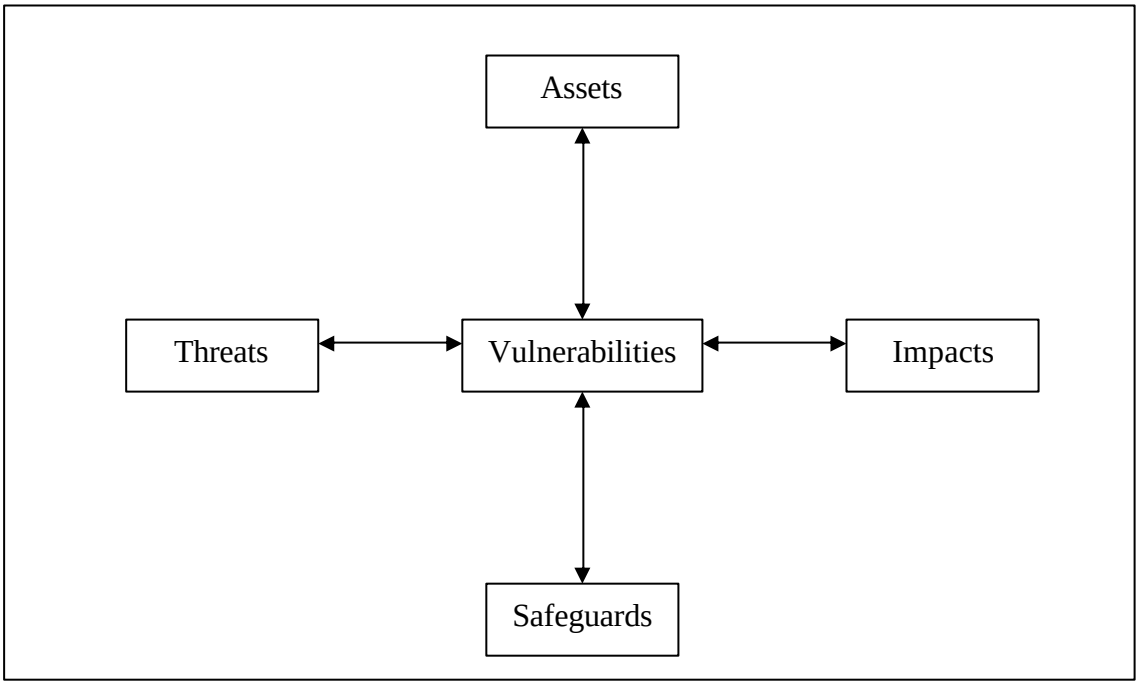

Figure 5.2: The Conventional Model used in Risk Analysis (Anderson, 1991)

This dissertation follows the guidelines of the detailed risk analysis approach as advocated by ISO/IEC TR 13335-3 (1998). This approach requires the detailed consideration of assets, threats and vulnerabilities, which ultimately will facilitate the selection of effective safeguards appropriate to the assessed risks. This approach is evident in the conventional model used in risk analysis, depicted in Figure 5.2.

The following sub-sections address the typical stages of the detailed risk analysis process, as described in ISO/IEC TR 13335-3 (1998), namely:

- The identification of assets;
- The valuation of assets;
- The threat assessment;
- The vulnerability assessment;

- The assessment of risks;
- The selection of safeguards;
- The implementation of safeguards.

5.7.1 The identification of assets

An asset is a component or part of a total system to which an organisation directly assigns value and, therefore, it requires protection. A key step in the risk analysis process is to identify the assets to be protected. From an information security perspective, the following assets may need protection (Bertine et al, 2004):

- Communications and computing services;
- Information and data, including software and data relating to security services;
- Equipment and facilities.

The information of an organisation is among its most valuable assets and is critical to its success. For the purposes of this study, the focus is on the protection of software applications and their associated information assets. The persons carrying out the risk analysis must be able to list the most important assets after consultation with the owners and users of the information.

The listing assets based on checklists and judgment should, according to Landoll (2006), yield an adequate identification of the critical assets of the organisation. If there are no documents or previous activities that have already classified the organisational assets, the process may be shortened by simply determining whether information is sensitive or not. It can then be stated that sensitive information requires protection and public data does not. This, however, may be an oversimplification for some organisations, especially those that must comply with information security regulations, such as HIPAA or the GLBA.

Another approach, that Landoll (2006) supports, is the categorisation of assets at three levels, namely critical, important and supportive. These levels are defined as:

- *Critical assets* – those assets that would prevent the organisation from accomplishing its core business functions if not protected;

- *Important assets* – those assets whose compromise would not prevent the organisation from accomplishing its core business in the short term, but would if the assets were not restored;
- *Supportive assets* – those assets whose compromise would not prevent the organisation from accomplishing its core business functions, but would affect the effectiveness or efficiency of day-to-day operations.

The next step in the risk analysis process, after having identified the assets requiring protection, is to assign some form of value to each of them.

5.7.2 The valuation of assets

The valuation of assets is an essential step in the overall risk analysis process. The next step in the risk analysis process, after having identified the assets requiring protection, is to assign some form of value to each of them. This value should be expressed in terms which are relevant to the asset and to the business entity involved. These values represent the importance of the assets to the business or organisation. According to ISO/IEC TR 13335-3 (1998), the values assigned should be related to:

- The cost of obtaining and maintaining the asset;
- The potential adverse business impacts resulting from its loss of confidentiality, integrity, availability, accountability, authenticity and reliability.

An example for such a valuation scale could be a distinction between:

- A 3-point scale (low, medium and high);
- A 5-point scale (negligible, low, medium, high and very high);
- A 7-point scale (negligible, very low, low, medium, high, very high and critical).

An organisation can define its own scale for determining its asset values. Landoll (2006) suggests, to simplify the process, that all critical assets are considered of high value, important assets have a medium value and supportive assets have a low value. This simplistic method comprises the classic qualitative approach.

Whitman and Mattord (2003) further recommend that the following data gathering questions be used to assist with the asset identification and valuation process:

- Which information assets are the most critical to the success of the organisation?
- Which information assets generate the most revenue?

- Which information assets generate the most profitability?
- Which information assets would be the most expensive to replace?
- Which information assets would be the most expensive to protect?
- Which information assets would be the most embarrassing or cause the greatest liability if revealed?

Other organisation-specific questions can add value to this process.

The next step in the risk analysis process involves a threat assessment because assets are subject to many kinds of threats, and threats have the potential to cause them harm.

5.7.3 The threat assessment

Peltier (2005) defines a threat as an undesirable event that could have an impact on the organisation. It is essential that no significant threat is overlooked in carrying out a threat assessment, since this could result in the failure or a weakness in the information system. A list of the most likely threats is helpful in performing a threat assessment, although one must be aware that threats are continually changing.

There are a number of different methods that can be used to create a complete list of threats. These include brainstorming, developing checklists and examining historical data. However, Peltier (2005) does caution that although the use of checklists is important, care must be taken that they do not negatively impact the free flow of ideas.

It is important to consider that a threat may arise from within the organisation when carrying out a threat assessment, for example, sabotage by an employee. The threat may arise from outside, for example, malicious hacking or industrial espionage. Common examples of threats include (Bertine et al, 2004):

- Unauthorised disclosure of information;
- Unauthorised destruction or modification of data, equipment or other resources;
- Theft, removal or loss of information or other resources;
- Interruption or denial of services;
- Impersonation or masquerading as an authorised entity.

Peltier (2005) classifies threats into three major categories, namely:

- *Natural threats* for example, floods, earthquakes, landslides and electrical storms;
- *Human threats*, which may be further sub-divided into:
  - *Unintentional acts* for example, errors and omissions, which statistically accounts for the largest loss to information assets;
  - *Deliberate acts* for example, fraud, malicious software and unauthorised access;
- *Environmental threats* for example, long-term power outages, pollution, chemical spills and liquid leakages.

Natural threats include equipment failures or disasters, such as fire and floods that can result in the loss of equipment and information. Natural threats usually affect the availability of processing resources and information (Killmeyer, 2006).

Corporate information can be easily accessed, compromised or destroyed by intentional, unintentional or natural threats. An intentional threat is one that is realised by someone committing a deliberate act. Intentional threats are unauthorised users who inappropriately access data and information that they are not granted permission to view or use. These unauthorised users can be internal or external to the organisation and can be classified as curious or malicious. Intrusion by malicious unauthorised users first results in a breach of confidentiality, which lead to breaches in integrity and possibly availability. An accidental threat is one with no premeditated intent, such as a system or software malfunction or a physical failure. Unintentional threats are typically caused by untrained or careless employees, who have not taken the necessary steps to ensure a secure environment. Such unintentional threats can include software developers who do not follow defined standards and procedures. This threat invariably exists in environments where no defined standards or procedures exist (Killmeyer, 2006).

It is important to recognise the various threat characteristics, as highlighted by ISO/IEC 13335-1 (2004) to perform a thorough threat assessment. These characteristics include:

- The source i.e. insider or outsider;
- The motivation (for example, financial gain or competitive advantage);

- The frequency of occurrence;
- The likelihood;
- The impact.

After considering the threat source (who and what causes the threat) and the threat target (i.e., what information assets may be affected by the threat), it is necessary to assess the likelihood of the threats. This likelihood should indicate the probability that a potential threat may be exercised against the asset under review. It will be useful to determine the impact that the threat may have on the overall organisation. At the completion of the threat assessment, there will be a list of the threats identified, the information assets they will affect, and measures of the likelihood of threats occurring on a 3-point, 5-point or 7-point scale as referred to in Section 5.7.2.

Threats are very real, and organisations are beginning to take the associated vulnerabilities, risks and potential losses to their information more seriously. Threats take advantage of vulnerabilities to cause destruction, interruption of operations, disclosure of information or denial of service, therefore, the next key step in the risk analysis process is the vulnerability assessment.

5.7.4 The vulnerability assessment

A threat needs to exploit an existing vulnerability to harm an asset. Nyanchama (2005) defines a vulnerability as a weakness in hardware or software that exposes an information system to attack, harm, interruption, or unauthorised exploitation. It may be a defect or weakness in system security procedure, design, implementation or internal control that an attacker can compromise. A vulnerability in itself, however, does not cause harm. It is merely a condition or set of conditions that will allow a threat to affect an asset.

Bertine et al (2004) classify vulnerabilities into four categories, namely:

- *Threat Model* vulnerabilities, which originate from the difficulty of foreseeing possible future threats;
- *Design and Specification* vulnerabilities, which arise from errors or oversights in the design of a system or protocol that make it inherently vulnerable;

- *Implementation* vulnerabilities are typically introduced by errors during system or protocol implementation;
- *Operation and Configuration* vulnerabilities originate from improper usage of options in implementations or weak deployment policies.

New security vulnerabilities are found almost daily. Each newly discovered vulnerability results in a frantic patch that potentially creates new vulnerabilities. This game of catch-up is endless. According to Cenzic (2003), the critical place to address security vulnerabilities is in the software development process.

Common development practices unfortunately leave software with many vulnerabilities (Task Force Report, 2004). These are generally caused by defective specification, design and implementation. McGraw and Verdon (2004) categorise software vulnerabilities as flaws (design-level problems) or bugs (implementation-level problems). Today, hackers attack software applications by seeking ways to manipulate input strings to gain super-user access, by stealing data or creating buffer overflows. They hunt for weaknesses in the many modules and components of complex systems, looking for hidden fields, embedded passwords, and available parameters to manipulate (Cenzic, 2003).

Issues and complexities in the current software development world lead to problems, with security vulnerability being one of the more significant by-products. Many security vulnerabilities result from defects that are unintentionally introduced in the software during design and development. According to a preliminary analysis done by the Computer Emergency Response Team (CERT) Coordination Centre, over 90% of software security vulnerabilities are caused by known software defects, and most software vulnerabilities arise from common causes. The top ten causes account for about 75% of all vulnerabilities. The overall specification, design and implementation defects in software must be reduced to significantly reduce software vulnerabilities, (Task Force Report, 2004).

According to Nyanchama (2005), vulnerabilities occur in software due to various factors, including poor coding practices and the complex nature of software itself. He

warns, however, that while good coding practices can reduce the number of defects, they cannot eliminate coding errors altogether. This can be attributed to various other factors including:

- Software engineering practices that predominantly emphasize functionality over safety;
- The labour-intensive nature of software engineering which makes software products susceptible to human error;
- The ongoing need to reduce software development costs;
- The practice of module reuse. An error impact is amplified whenever a faulty module is reused in a different application or different parts of an application.

The presence of a vulnerability does not cause harm in itself because there must be a threat present to exploit it. This relationship becomes clear when considering the following examples of common software vulnerabilities, as provided by ISO/IEC TR 13335-3 (1998):

- Unclear or incomplete specifications for developers which can be exploited by, for example, the threat of software failure;
- None or insufficient software testing which can be exploited by, for example, the threat of use of software by unauthorised users;
- A complicated user interface which can be exploited by, for example, the threat of user error;
- Lack of identification and authentication mechanisms, like user authentication which can be exploited by, for example, the threat of masquerading of user identity;
- Lack of audit-trail which can be exploited by, for example, the threat of use of software in an unauthorised manner;
- Well-known flaws in the software which can be exploited by, for example, the threat of the unauthorised use of software;
- Unprotected password tables which can be exploited by, for example, the threat of masquerading of user identity;
- Poor password management which can be exploited by, for example, the threat of masquerading of user identity;

- Incorrect allocation of user rights which can be exploited by, for example, the threat of use of software in an unauthorised manner;
- Uncontrolled downloading and using software which can be exploited by, for example, the threat of malicious software;
- Lack of effective change control which can be exploited by, for example, the threat of software failure;
- Lack of documentation which can be exploited by, for example, the threat of user error.

According to ISO/IEC TR 13335-3 (1998), it is important to assess how severe the vulnerabilities are, or how easily they may be exploited. A vulnerability should be assessed in relation to each threat that might exploit it in a particular situation.

It is clear, from this discussion, that a vulnerability assessment is necessary to identify any potential weaknesses. It identifies vulnerabilities that may be exploited by threats and assesses their likely level of weakness. A vulnerability, which has no corresponding threat, does not require the implementation of a safeguard, but should be recognised and monitored for changes. It is important to realise that an incorrectly implemented safeguard, or a safeguard being used incorrectly, could itself be a vulnerability.

It is necessary to address the assessment of risks as the next stage in the risk analysis process before discussing the selection and implementation of safeguards,

5.7.5 The assessment of risk

There would be no risk without vulnerabilities, A security risk, as defined by Landoll (2006), is the potential loss to organisational assets that will likely occur if a threat is able to exploit a vulnerability. The objective of this step is to identify and assess the risks to which the software application and its associated information assets are exposed to identify and select appropriate and justified security safeguards.

The basic equation for risk calculation is: **Risk = Asset × Threat × Vulnerability**.

This simple equation illustrates the principle that risk is calculated, based on an understanding of the asset value, the extent of the threat, and the likelihood of the threat

exploiting an existing vulnerability. It is tempting to think that determining risk is a simple calculation. However, the determination of the value of assets, the frequency of the threat, and the likelihood of a vulnerability existing is clouded by uncertainty (Landoll, 2006).

The result of this step should be a list of measured risks for each of the impacts of disclosure, modification, non-availability and destruction for the information system under consideration according to ISO/IEC TR 13335-3 (1998). The specific measures of risk will help identify which risks should be dealt with first when selecting safeguards.

5.7.6 The selection of safeguards

Safeguards must be selected, based on their effectiveness in addressing the indicated security risks (Landoll, 2006). The results of the risk assessment should be considered to select safeguards, which effectively protect against these risks. It is useful to consider the vulnerabilities that are to be protected against to identify the safeguards. Catalogues may prove helpful in the identification and selection of safeguards, although these will need to be customised to meet the specific security needs of the software application under consideration. Safeguard catalogues generally recommend a set of safeguards to protect against the most common threats.

The selection of safeguards should, according to ISO/IEC TR 13335-3 (1998), include a balance between operational (non-technical) and technical safeguards. Operational safeguards are those which provide physical, personnel and administrative security. Physical security safeguards include, for example, key-coded door locks, fire suppression systems and security guards. Technical security comprises hardware, software and communications safeguards. These safeguards are selected, according to the risks identified, to provide security functionality and assurance. Security functionality includes identification and authentication, logical access controls, audit trail/security logging, dial-back security, message authentication and encryption. Assurance documents the level of trust needed in security functions, and therefore, specifies the amount of checking and security testing required to confirm that level of trust (ISO/IEC TR 13335-3,1998).

The ISO/IEC TR 13335-4 (2000) suggests that the process of safeguard selection requires some knowledge of the type and characteristics of the information system under consideration, because this has a significant influence on the safeguards selected to protect it. A standalone workstation, for instance, clearly requires the implementation of different safeguards to that of a server or workstation connected to a network. The compatibility of the existing safeguards with the selected ones must be considered when selecting safeguards,. A particular safeguard may conflict with another or hinder its successful operation and the protection provided.

The ISO/IEC TR 13335-4 (2000) refers to the selection of safeguards according to security concerns and threats in the following ways:

- Identify and assess the security concerns. The requirements for confidentiality, integrity, availability, accountability, authenticity and reliability are considered. The strength and number of safeguards selected should be appropriate to the assessed security concerns;
- Typical threats are listed for each of the security concerns. Appropriate safeguards are suggested for each threat listed, according to the information system under consideration.

There are basically four aspects that a safeguard can address, namely impacts, threats, vulnerabilities and the risks themselves. The risk itself is addressed when the decision is made to transfer the risk rather than accept it. The impacts, threats and vulnerabilities are the main targets of safeguards. According to ISO/IEC TR 13335-4 (2000), they are addressed as follows:

- Safeguards can reduce the likelihood of a threat occurring. For example, consider the threat of data loss because of user errors - a training course can reduce the number of user errors;
- Safeguards can remove a vulnerability or make it less serious. For example, if an internal network is vulnerable to unauthorised access from an external network, the implementation of a firewall could make the connection less vulnerable;
- Safeguards can reduce or avoid the impact. For example, the adverse impact of non-availability of information can be reduced by making a copy of that information and storing it safely elsewhere.

The ISO/IEC TR 13335-3 (1998) suggests that various factors be considered when selecting safeguards for implementation, including:

- The ease of use of the safeguard;
- Transparency to the user;
- The help provided to the users to perform their functions;
- The relative strength of the safeguards;
- The types of functions performed (prevention, deterrence, detection, recovery, correction, monitoring, and awareness).

A further important aspect is cost. It is inappropriate to recommend safeguards which are more expensive to implement and maintain than the value of the information assets they are designed to protect (ISO/IEC TR 13335-3, 1998). Peltier (2006) proposes that a cost-benefit analysis be carried out to assist in identifying those safeguards that offer the maximum amount of protection at a minimum cost. How and where a safeguard is used will have a significant influence on the benefits gained from its implementation.

5.7.7 The implementation of safeguards

The proposed safeguards should be compared with the existing safeguards, prior to implementation, to assess whether they can be extended or upgraded. This can be less expensive than introducing new safeguards (ISO/IEC TR 13335-4, 2000).

Technical constraints, such as performance requirements, manageability and compatibility issues may hamper the use of certain safeguards. It could even be the case that a particular safeguard may decrease system performance. Aspects such as privacy legislation may demand that certain safeguards be in place (ISO/IEC TR 13335-4, 2000).

The documentation of safeguards is an important part of the information security documentation to ensure continuity and consistency. Much of the documentation, particularly on threats, vulnerabilities and risks, is very sensitive and must be protected from unauthorised disclosure. Therefore, it can be argued that security documentation is kept separate from the typical systems documentation developed during software development.

5.8 Criteria for Effective Risk Analysis

The perceived importance of security assessment in most information systems is such that risk analysis is, at best, a cyclic and, at worst, a once-off activity. Therefore, methodologies are needed to control and direct data collection and interpretation and to test completeness in the form of some "deliverable". Risk analysis, as an estimating task, has several major, independent sources of uncertainty.

The suitability of a methodology can be assessed, according to Anderson (1991), using a series of criteria with associated metrics, including:

- The ability to produce consistent results when applied to the same case by different analysers;
- Usability which is measured by the effort to learn and use as compared to the results obtained;
- Adaptability to different types of system configuration;
- Feasibility of acquiring input data economically;
- Completeness which is the acknowledgement of all relevant aspects of risk;
- Validity which is the correspondence of the output model to reality;
- Credibility which is the extent to which users feel they can rely on the results.

5.9 Conclusion

When developing a software solution, it is imperative that the first and driving factor for implementing a secure solution is the business need. An information security point of view requires that the particular business need is determined by performing some form of risk analysis. Technical security measures (for example, passwords), and non-technical measures (for example, secure operating procedures for personnel), are needed to meet these demands.

It has been determined that information security is concerned primarily with the confidentiality, integrity and availability of information. This implies that a secure software development environment should ensure that only authorised users have access to sensitive information, that the information is processed, stored and communicated correctly and that it is available when necessary. This will, when correctly maintained, result in a greater degree of assurance to the users of the information. The easier it

becomes for software applications to exchange information, the more difficult it is to protect them from various security risks.

A security risk is a measure of the adverse effects that can result if a security vulnerability is exploited, i.e., if a threat is realised. Risk can never be eliminated, however, one objective of security is to reduce risk to an acceptable level. This means it is necessary to understand the threats and vulnerabilities and to apply appropriate countermeasures (i.e., security services and mechanisms). Threats and threat agents change, but security vulnerabilities exist throughout the life of a system or protocol unless specific steps are taken to address vulnerabilities.

Risk Analysis	**Risk Identification**	Identification and valuation of information assets. Identification and assessment of threats. Identification of vulnerabilities.
	Risk Assessment	Quantitative using mathematical and statistical tools. Qualitative using relative rankings or heuristic estimates.
	Risk Prioritisation	According to the risk value determined during the risk assessment.
Risk Management	**Safeguard Selection**	Identification and selection of the relevant security services and mechanisms.
	Safeguard Implementation	Selection of the appropriate tools and components.

Table 5.1: The Distinctive Elements of Risk Analysis and Risk Management

It is important that a simplified approach is used to educate software developers in risk analysis and risk management, specifically in the software development arena, but not to the detriment of the results achieved. Table 5.1 highlights the most important elements that should be inherent in the approach taken. A good approach may be to find a balance between minimising the time and effort spent in identifying the safeguards, whilst ensuring that the high risk assets are appropriately protected. The incorporation of an initial quick and simple approach is likely to gain greater acceptance and, therefore, be more widely used.

Many of the security vulnerabilities currently found in software applications can be overcome by integrating a high-level approach to risk analysis into the SDLC. The following chapter proposes a model for secure software development, which integrates risk analysis into various stages of the SDLC.

Chapter 6

The Secure Software Development Model

6.1 Introduction

Chapter 5 introduced various risk analysis approaches, strategies and methodologies, and described the process in detail. It addressed common problems in risk analysis and recommended criteria for consideration. This chapter describes the phases of the Secure Software Development Model (SecSDM).

The problem of producing secure software is both a software engineering and security engineering problem. Software engineering addresses planning, tracking and measurement whilst security engineering addresses methods and tools needed to design, implement and test secure systems. Secure software development requires an integration of these two engineering approaches (Task Force Report, 2004).

The complex and highly connected computing environment of today triggers several security concerns. Security problems involving computers and software are frequent, widespread, and serious. The number and variety of attacks by persons and malicious software from outside the organisation, particularly via the Internet, are increasing rapidly. The evident separation between information security and software development has resulted in the production of vulnerable software applications.

In a climate where the protection of information is increasingly tied to the integrity of an organisation, security must be strongly coupled to the software development process ensuring that the desired level of security is achieved (Tipton and Krause, 2006). Organisations expect their data to remain confidential, to be available when required, and not be subject to unauthorised modification. This makes the security of information to be of the utmost importance (Tryfonas and Kiountouzis, 2002). It is necessary to develop an improved software development process to build better or more secure software. Security considerations must provide input into every phase of the SDLC.

Software developers generally ignore the idea of security. This leads to software applications that have many avoidable security weaknesses. Jurjens (2002) argues that software developers rely on their intuition in developing secure software and do without much systematic help or guidance. Therefore, it is unsurprising that security breaches continue to occur in software applications. Cenzic (2003) states that this is due to the lack of attention to various core problems, namely insecure software development and the lack of security testing. Cenzic (2003) suggests that human error, as a result from the lack of education or skill, has caused these issues to surface.

It is argued that building secure software begins with the effective education of software developers. These professionals need be educated to put security at both the heart of software design and at the foundation of its development process. This implies that software developers need to use improved processes that consistently produce secure software. The Task Force Report (2004) notes that there do not currently exist software development processes or practices that consistently produce secure software. They recommend that software producers adopt practices that can measurably reduce software specification, design and implementation defects and, therefore, minimise any potential risk.

The key to meeting the current demand for improved security, for the software industry, is to implement repeatable processes that reliably deliver measurably improved security. Therefore, a more stringent software development process that focuses on security is required. It should minimise the number of security vulnerabilities present in the SDLC and detect and remove these vulnerabilities as early in the life cycle as possible (Lipner and Howard, 2005). Tryfonas and Kiountouzis (2002) are in agreement, and suggest that new ways of addressing and resolving security issues, early within the SDLC, must be introduced in the software development arena.

This chapter describes a model for incorporating security into the SDLC. It proposes a more stringent software development methodology that both detects and removes vulnerabilities early in the life cycle, thereby minimising the number of security vulnerabilities in the live system. The SecSDM aims to draw attention to the importance of security in the SDLC and it is designed as an extension, not a replacement to pre-existing software development methodologies. Software developers should incorporate

the concepts from each phase of the SecSDM into the corresponding phases of the existing SDLC to ensure that security is appropriately considered and built into the software application. This type of inclusion should result in a robust end product that is more secure, easier to maintain and less costly to own. It is argued that although many researchers advocate that security needs to be integrated into the SDLC, few are able to describe a process to achieve this goal. Chapter 6 motivates an incremental process for incorporating information security into all phases of the SDLC, as illustrated in Figure 6.1.

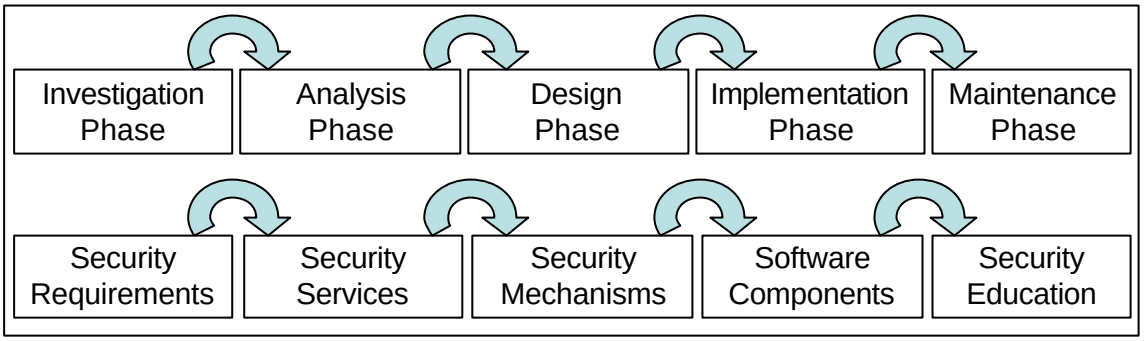

Figure 6.1: Security in the SDLC

6.2 Why the Need for a Secure Software Development Model?

Application security has been largely overlooked by many organisations protecting their assets, despite the fact that software applications have been a primary target for many hackers. It has been estimated, for instance, that nearly 90% of Web applications contain major security holes (Landoll, 2006). They are still vulnerable, regardless of the fact that most of these applications sit behind a firewall, and in some cases behind hardened operating systems.

Applications currently are networked and used in highly distributed environments. The current state of application security, however, reflects the fact that security has been an afterthought. Recently, protecting the confidentiality and integrity of data in transit and storage was the primary concern, and cryptography successfully addressed this problem. However, the provision of security within networked information systems extends beyond protecting data and cryptographic keys. Main (2004) argues that it is not enough to build perimeter defences around the network. The approach to application security must be driven by a clear and thorough understanding of the potential threats to the

application. Application security must, therefore, extend beyond traditional network and data security to address the attacks against the software itself. According to Main (2004), the design of security that is intrinsic and inseparable from the application is essential. Developers can weigh the investment required in providing adequate application security against the anticipated threats by planning for software protection as part of the creation of an application (Main, 2004).

Landoll (2006) states that organisations need to protect their information systems from unscheduled changes, third-party access, system-level and application-level vulnerabilities. Any of these threats could lead to disclosure or corruption of sensitive information, subversion of network systems or fraud (Landoll, 2006). It is important to provide software developers with an understanding of the vulnerabilities and exposures that could introduce security risks into software applications. The intended goal is to enable developers to write more secure code and to provide a greater level of assurance that software applications are not exposed to vulnerabilities when integrated with other systems and applications.

Common development practices, unfortunately, leave software with many vulnerabilities. These are caused primarily by defective specification, design and implementation processes. An inappropriate implementation may, for instance, contain vulnerabilities facilitating attacks, and an insecure design or life cycle model may allow Trojan horses in the source code (Leiwo, Kwok, Maskell & Stankovic, 2005). These software specification, design, and code defects are unknowingly injected by developers. This could be attributed to the fact that developers often use methods of coding that are inherently insecure. Logic modules, for example, are typically written in a manner that creates security issues when they are integrated. Applications are often installed that create vulnerabilities rather than eliminate them (Cenzic, 2003).

The supporting software must contain few, if any, vulnerabilities to have a secure infrastructure. This requires that software is built to sound security requirements and contain few specification, design or code defects. Software development processes must change to produce software with few defects. The Task Force Report (2004), reports that this requires that developers use methods that consistently produce secure software. This, in turn, requires development organisations to:

- Acquire a high-level of security expertise;
- Identify and adopt processes for producing low-defect, secure software;
- Consistently use this security expertise and these processes when they produce, enhance, maintain and rework software.

Procedures should be implemented which ensure that software development adheres to approved development and maintenance methodologies, to guarantee secure software development. A formal methodology for secure software development is necessary to ensure that appropriate controls are built into the application to provide confidentiality, integrity and availability of the information.

6.3 Support for Secure Software Development

Security is perhaps the final frontier for developers as they build applications for the modern enterprise. Microsoft has stated that it will deliver tools that enable developers to broadly cover the application life cycle, including tools that address design, coding, issue tracking, source code control and various forms of testing. Rick Samona, product manager for the .Net framework and developer tools at Microsoft, was quoted as saying that every organisation, small or large, must have an existing security design life cycle to ensure that security occurs during all phases. Organisations must provide their developers with the adequate training to write secure applications. According to Taft (2004), a recent Microsoft study demonstrated that 64% of developers are not confident in their ability to write secure applications. This is despite the fact that the .Net framework and Visual Studio.Net provide developers with the necessary tools and information to write secure applications.

Jones and Rastogi (2004) define a software development methodology that starts with an understanding of business objectives and security requirements. It ensures that security is designed into the application and thoroughly tested throughout the SDLC. Their methodology has interwoven security into every stage of the SDLC, however, the SecSDM provides a more structured approach to achieving a secure software product. The methodology of Jones and Rastogi (2004), like the SecSDM, is based on existing risk management practices. The following should be considered in the process of integrating security into the SDLC (Jones and Rastogi, 2004):

- Gain support from the highest levels within the organisation;
- Assess the current capability of the organisation with respect to secure coding;
- Develop a phased approach for integrating security into the SDLC;
- Ensure developers, project managers and system architects undertake formal training to secure coding;
- Ensure that secure coding reference materials are available;
- Establish internal metrics and key performance indicators.

Similarly, the Task Force Report (2004) suggests that any software development process must meet the following requirements to effectively produce secure software, namely:

- *Coverage* - a secure process must cover the full software life cycle, from the earliest requirements through design, development, delivery, maintenance, enhancement and retirement;
- *Definition* - the process must be precisely defined so that it can be taught, supported, verified, maintained, enhanced and certified;
- *Integrity* - the process must establish and guard the integrity of the product throughout its life cycle;
- *Measures* - the process must include measures to verify that the developers are capable of consistently and correctly using the process, that the correct process was properly used to develop the product, and that the process consistently produces secure software products;
- *Tailoring* - the process must permit tailoring and enable verification that such tailoring does not compromise the security of the resulting products;
- *Usability* - the process must be usable by properly trained and qualified software developers, security specialists, managers and other professionals. It must be economical and practical to use for developing a wide and defined range of software product types;
- *Ownership* - the process must be owned, supported, widely available, and regularly updated to reflect changing threat conditions and improvements in knowledge and technology;

- *Support* - the process must be fully supported with training programs, course material, instruction guides and supporting tools;
- *State of the Practice* - the process must include the use of quality practice methods for design, development, test, product measurement, and product documentation.

Gregory (2003) advocates that one of the key steps to integrating security into the SDLC is to ensure that all the stakeholders who need security information has it available, in a useful, phase specific format. The improvement of software engineering practices and processes will lead to secure software; software released with fewer defects; lower development and maintenance costs and an enhanced reputation for the final product.

6.4 Phases of the Secure Software Development Model

A key notion underlying the creation of the SecSDM, is the inclusion of security activities throughout the SDLC. Various security design principles, as recommended by Tipton and Krause (2006), were taken into account in developing this model. These include:

- Avoid security for its own sake and focus on the overall capability and the associated risk factors;
- Address the key security areas of identification, authentication, authorisation, confidentiality, integrity, availability, accountability and non-repudiation;
- Build multiple layers of controls;
- Strive for transparent security;
- Keep security simple;
- Consider the life cycle of the software application;
- Favour mature and proven security technologies.

These principles clearly support the need to ensure that appropriate security concerns are addressed throughout the SDLC to minimise the associated risks. The main security concerns to be addressed at each phase of the SDLC, according to the SecSDM, are as follows:

- *Investigation Phase*: This determines the security requirements of the software application by executing a simple risk analysis exercise;
- *Analysis Phase*: This determines the security services to be used to satisfy the security requirements;
- *Design Phase*: This determines how the security services will be implemented;
- *Implementation Phase*: This identifies and implements the tools and components;
- *Maintenance Phase*: This educates users in the correct operation of the software application in a secure manner.

This section describes the SecSDM as a simple, ten-step process for integrating security into each phase of the SDLC as depicted in Appendix A. These key steps include:

- *Investigation Phase;*
  - STEP 1: Information asset identification and valuation;
  - STEP 2: Threat identification and assessment;
  - STEP 3: Risk (asset/threat) identification;
  - STEP 4: Determine the level of vulnerability;
  - STEP 5: Risk assessment;
  - STEP 6: Risk prioritisation.
- *Analysis Phase;*
  - STEP 7: Identify the relevant security services and level of protection required to mitigate each risk.
- *Design Phase;*
  - STEP 8: Map security services to security mechanisms;
  - STEP 9: Summary of findings.
- *Implementation Phase;*
  - STEP 10: Map security mechanisms to .Net and other security components.
- *Maintenance Phase;*
  - The maintenance of software is made easier and more manageable through the structured approach provided by the SecSDM.

An important consideration in developing this incremental process has been to develop a useable process that will lessen the burden for software developers who are not specialists in information security.

The investigation phase, as described next, incorporates six of the ten steps. This highlights the fact that the security requirements of a software application need to be defined early in the life cycle.

6.4.1 The investigation phase

One of the primary concerns with traditional software development methodologies is that the security needs of a given system are often not determined until into the implementation phase. This results in late and expensive attempts to incorporate security into the work in progress. Security requirements are normally stated in terms of how to achieve security and not in terms of the problem to be solved. This leaves it unclear how the security requirements may affect functional requirements (Haley, Laney & Nuseibeh, 2004).

Early determination of the security requirements assists in the development of software applications which better meet the needs of all the stakeholders. The question posed is how to state the security requirements of a particular system or software application? ISO/IEC TR 13335-3 (1998) suggests that information security requirements are stated in terms of confidentiality, integrity, availability, accountability, authenticity and reliability of information. It is necessary to perform some form of a risk analysis to determine the security requirements of a particular system. The conducting of a security risk analysis determines the security vulnerabilities and shortcomings of the proposed software application. It identifies countermeasures that will ensure that the security of the information is maintained under all circumstances (Booysen and Eloff, 1995). It is assumed, for the purposes of the proposed risk analysis approach, that the software application being developed is at a relatively high risk. The guidelines of the detailed risk analysis approach, as recommended by ISO/IEC TR 13335-3 (1998) are followed and were described in Chapter 5.

The proposed risk analysis process carried out during the investigation phase takes the form of a step-by-step process. Its purpose is to identify the information assets, their

associated threats and vulnerabilities, and rank them according to those assets that need the most protection.

The primary aim of the risk analysis is to ensure that the most important information assets that require protection from potentially harmful threats are identified early in the SDLC. Existing security policies should be analysed during the investigation phase, and any legal issues that can impact the design of the software examined. The outcome of this phase is a set of security requirements that describes the risks that need to be addressed throughout the rest of the life cycle.

STEP 1 : Information asset identification and valuation

A key step in preparing for a risk analysis is to identify the assets to be protected. The first step, according to the SecSDM, is to identify the key information assets pertaining to the software application being developed. It provides a foundation to the risk analysis process and simultaneously initiates an awareness and understanding of the information assets that need protection.

An asset is, by definition, a component or part of a total system to which an organisation directly assigns value and, therefore, requires protection. Critical assets can be the data stored, processed and transmitted by IT facilities, such as software and hardware products, systems or applications. Information is among the most valuable assets of an organisation and is, therefore, critical to its success. All information is valuable to an organisation, but some is more critical and sensitive than others. Information must be available to be useful. Connectivity makes information available when and where it is needed and is the nature of doing business today. This very availability puts information at risk. Therefore, it is necessary to identify the most important information assets so that additional controls or safeguards can be implemented to provide adequate protection.

Software developers need to both determine and understand what information assets require protection to develop secure software applications. Different industries and different systems have varying information protection requirements. For example, healthcare organisations stress the confidentiality of patient records, whereas banking is more concerned about the integrity of monetary transactions. The software development

team needs to understand and capture what the adequate protection of information is, in their specific context (Tipton and Krause, 2006).

INVESTIGATION PHASE *STEP 1a : Information Asset Identification*	
Which information assets are the most **critical** to the **success** of the proposed software application?	*1. Customer orders*
	2. Product specifications
	3. Supplier contract details
Which information assets pertaining to the proposed software application generate the most **revenue**?	*1. Customer orders*
	2. Product specifications
	3. Supplier product details
Which information assets pertaining to the proposed software application generate the most **profitability**?	*1. Customer orders*
	2. Product specifications
	3. Supplier product details
Which information assets pertaining to the proposed software application would be the most **expensive to replace**?	*1. Product specifications*
	2. Supplier contract details
	3. Customer orders
Which information assets pertaining to the proposed software application would be the most **expensive to protect**?	*1. Product specifications*
	2. Supplier contract details
	3. Customer orders
Which information assets pertaining to the proposed software application would be the most **embarrassing** or cause the greatest **liability if revealed**?	*1. Customer orders*
	2. Employee salary details
	3. Product specifications

Table 6.1: Investigation Phase - Information Asset Identification

The listing of assets, according to Landoll (2006), based on checklists and judgement, yields an adequate identification of the important assets of the organisation. These information assets can include, for example, personal information, employee salary information, customer contact information or financial information. Software developers are required to list, to assist with the asset identification process and in consultation with the relevant stakeholders, the three information assets that apply to each of the data gathering questions, as recommended by Whitman and Mattord (2003). For example, Table 6.1 illustrates that customer orders, product specifications and supplier contract details are the three information assets identified as the most critical to

the success of the software application in question. This list of questions can be updated to include other organisation-specific questions that will add value to this process.

Those information assets that pertain to the questions posed in Table 6.1 require special consideration. It is recommended that the five most important information assets that need protection be identified to simplify the process. These can be confirmed through consultation with the owners of the information assets identified. Once an agreement is reached, on the five most important information assets, these need to be listed and labelled Asset A, Asset B, Asset C, Asset D and Asset E respectively, as illustrated in Table 6.2. According to this example, the five most important information assets are customer orders, product specifications, supplier contract details, supplier product details and employee salary details.

INVESTIGATION PHASE *STEP 1b : Information Asset Valuation*					
Information Assets	**Asset Impact Value**				
	0 Negligible	**1 Low**	**2 Medium**	**3 High**	**4 Critical**
Asset A *Customer Orders*					*X*
Asset B *Product Specifications*					*X*
Asset C *Supplier Contract Details*				*X*	
Asset D *Supplier Product Details*				*X*	
Asset E *Employee Salary Details*				*X*	

Table 6.2: Investigation Phase - Information Asset Valuation

The next step in the process is to assign values to each of these five information assets. Ideally, the value assigned is expressed in terms which are relevant to the asset and to the business entity involved. These values represent the business importance of the assets and will typically be obtained by interviewing the information owners and its key users. This is necessary to determine the impact value and sensitivity of the information in use, stored, processed or accessed. Although for many organisations it could be more appropriate to establish this value in monetary terms, for simplicity it is recommended that a 5-point Lickert scale is used to establish this value. The SecSDM requires that an

asset impact value between 0 and 4 (where 0=negligible and 4=critical) is assigned to each of the five information assets, based on its financial value or worth to the organisation. Landoll (2006) refers to this as classification-based valuation, whereby assets are classified into one of several levels that indicate their qualitative value. For many organisations the qualitative approach provides adequate asset valuation with less effort than quantitative asset valuation approaches. The 5-point Lickert scale used by the SecSDM is based on the classification levels as recommended by Landoll (2006). These are defined as follows:

- *0 (NEGLIGIBLE)* indicates an insignificant or no impact on human life or the continuation of the operation of critical business functions;
- *1 (LOW)* indicates a slight impact on human life or the continuation of the operation of critical business functions;
- *2 (MEDIUM)* indicates that compromise of the asset would have moderate consequences that would impair the operation of a critical business function for a short time;
- *3 (HIGH)* indicates that the compromise of the asset would have serious consequences that could impair the operation of a critical business function;
- *4 (CRITICAL)* indicates that compromise of the asset would have grave consequences leading to a loss of life or serious injury to people and/or long-term disruption to the operation of a critical business function.

The SecSDM provides a table which maps the most critical assets (rows) against the asset impact values (columns). For each of the information assets (A, B, C, D and E) identified, software developers are required to indicate its asset impact value by simply ticking the appropriate cell in the table provided. Only five cells (i.e., one asset impact value per information asset), must be marked off as illustrated in Table 6.2.

The identification and classification of information assets in this manner helps determine the value of the most important information assets, and provides an awareness with respect to their value. The classification of certain information assets in this way brings about awareness of its value to users authorised to handle that information. The most important information assets are identified and assigned values to each, based on their worth to the organisation, the next step requires the identification of the various threats that may cause harm to these assets.

STEP 2 : Threat identification and assessment

A threat can be defined as any undesirable event that can have a negative impact on an organisation. Threat identification, in the software development context, addresses those threats with the potential of causing the maximum damage to the information assets pertaining to the particular software application. It is necessary to perform the identification and assessment of threats during the investigation phase of the SDLC, because it is used to identify risks and to guide subsequent design, coding and testing decisions.

A wide variety of threats face the information systems of an organisation. Each has the potential to attack any of the information assets identified in Step 1. Experts find it increasingly challenging to accurately address all relevant threats due their volume and their ever-changing nature. The following questions, as proposed by Whitman and Mattord (2003), need to be considered to assist with the identification of threats,:

- Which threats represent the most danger to the information assets pertaining to the proposed system?
- How much would it cost to recover from a successful attack?
- Which of the threats would require the greatest expenditure to prevent?

The answers to these questions can help identify those threats that will have the most severe impact on the information assets identified in Step 1.

Checklists help software developers with the recording of items to be checked or remembered. Therefore, a checklist of the most likely threats is helpful in performing a threat assessment, although one must be aware that threats are continually changing. In order to simplify the threat identification and assessment process, a checklist of the most common threats is provided by the SecSDM, based on those referred to in ISO/IEC TR 13335-3 (1998). These questions are provided in Table 6.3 and include:

- Theft of information for example, the illegal disclosure of information;
- Use of system by unauthorised users for example, the unauthorised access by competitors;
- Use of system in an unauthorised manner for example, the unauthorised collection of data;
- Masquerading of user identity for example, malicious hacking;

- Malicious software attacks for example, viruses, worms and denial of service attacks;
- User errors for example, invalid or inaccurate data entry;
- Repudiation for example, the denial of having performed a transaction;
- Technical software failures or errors for example, bugs, code problems and unknown loopholes.

Software developers can add any additional threats to the standard list provided. However, it is important that software developers are informed that security-related checklists must be used with a corresponding process, such as the one described in this chapter, to be useful (Task Force Report, 2004).

Danger of a threat is something difficult to assess. Danger may be simply the likelihood of a threat attacking the information asset, or it may represent the amount of damage the threat could create. It may also represent the frequency with which an attack can occur (Whitman and Mattord, 2003). It is necessary, as part of the threat assessment process, to determine the likelihood, frequency and potential impact that each of the common threats may have on the software application in question. This may be performed, according to the SecSDM, by assigning each of the threats listed above to one of the following likelihood/frequency/impact levels, namely:

- *LOW* if the loss of confidentiality, integrity or availability could be expected to have a limited adverse affect on the organisational operations, assets or individuals. For example, an interruption with no financial loss. This could also indicate a low likelihood and low frequency of occurrence;
- *MEDIUM* if the loss of confidentiality, integrity or availability could be expected to have a serious adverse affect on the organisational operations, assets or individuals. For example, a short interruption that results in a limited financial loss to a single business unit. This could also indicate a medium likelihood and medium frequency of occurrence;
- *HIGH* if the loss of confidentiality, integrity or availability could be expected to have a severe or catastrophic adverse affect on the organisational operations, assets or individuals. For example, the shutdown of a critical business unit that leads to a significant loss of business, corporate image or profit. This could also indicate a high likelihood and high frequency of occurrence.

INVESTIGATION PHASE *STEP 2 : Threat Identification and Assessment*				
Common Threats (ISO/IEC TR 13335-3:1998)	**Level of Likelihood/Frequency/Impact**			
	LOW	**MED**	**HIGH**	**N/A**
Theft of information (Deliberate e.g. Illegal information disclosure)			*X*	
Use of system by unauthorised users (Deliberate or Accidental e.g. Unauthorised access by competitors)		*X*		
Use of system in an unauthorised manner (Deliberate or Accidental e.g. Unauthorised data collection)		*X*		
Masquerading of user identity (Deliberate e.g. Malicious Hacking)			*X*	
Malicious software attacks (Deliberate or Accidental e.g. viruses, worms, DoS)			*X*	
User errors (Deliberate or Accidental e.g. Invalid/inaccurate data entry)			*X*	
Repudiation (Deliberate e.g. Denial of having performed transaction)		*X*		
Technical software failures or errors (Deliberate or Accidental e.g. Bugs, code problems, unknown loopholes)			*X*	
Other …….. ……..				
Other …….. ……..				

Table 6.3: Investigation Phase - Threat Identification and Assessment

The SecSDM provides a table, as illustrated in Table 6.3, mapping the most common threats (rows) against the likelihood, frequency and potential impact (columns). Software developers are required, for each of the most common threats listed, to indicate the likelihood, frequency and potential impact of the threat in question by checking the appropriate cell in the table provided. Threats that are deemed to be not applicable, requires software developers to check an 'X' in the 'N/A' column. Only eight cells (i.e., one level per threat) must be checked on this table unless additional threats as identified by the software developers are added.

Having identified and assessed the various threats that may have a negative impact on the information assets of the organisation, it is then necessary to identify the most critical risks for which the software application must provide protection. This is described in Step 3.

STEP 3 : Risk (asset/threat) identification

Risk identification requires that the most critical asset/threat relationships are identified to ascertain which risks are most likely to impact the proposed system (Whitman and Mattord, 2003). This is done by simply considering the five most important information assets, as identified in Step 1, and the most common threats listed in Step 2. Those assets with high or critical asset impact values (i.e., 3 or 4) and those threats recognised to have a potentially high impact will contribute significantly to the criticality of the risk.

The SecSDM provides a table, as illustrated in Table 6.4, which maps the most common threats listed in Step 2 (rows) against the most important information assets, identified in Step 1 (columns). Software developers are required to determine the eight most critical risks (i.e., asset/threat relationships) by checking a letter from 'a' to 'h' in the appropriate cell. Each of the asset/threat relationships ('a' to 'h') refer to the corresponding risk ('A' to 'H') in the steps that follow. Software developers, in determining these eight risks ('A' to 'H'), must consider both the asset impact value of the particular asset (as indicated in Step 1) and the likelihood, frequency and potential impact of the particular threat (as indicated in Step 2). For example, if Asset A has an asset impact value of 4 and the likelihood, frequency and potential impact of the 'theft of information' on Asset A is HIGH, then the corresponding cell is labelled 'a'. Only eight cells (labelled 'a' to 'h') must be checked on this table unless additional threats, as identified by the software developers, are added.

INVESTIGATION PHASE *STEP 3 : Risk (Asset/Threat) Identification*					
ASSET/THREAT RELATIONSHIP					
Common Threats (ISO/IEC TR 13335-3:1998) (refer to Step 2)	**Information Assets (refer to Step 1)**				
	Asset A	**Asset B**	**Asset C**	**Asset D**	**Asset E**
Theft of information	*a*	*b*			*e*
Use of system by unauthorised users		*f*			
Use of system in an unauthorised manner					*g*
Masquerading of user identity	*c*				
Malicious software attacks	*d*				
User errors		*h*			
Repudiation					
Technical software failures or errors					
Other ……					
Other ……					

Table 6.4: Investigation Phase - Risk (Asset/Threat) Identification

This step results in the identification of at least eight of the most prominent risks. The following step in the process requires that the level of vulnerability for each critical risk be determined.

STEP 4 : Determine the level of vulnerability

A vulnerability is a weakness or fault in a system that an attack exploits. However, the presence of a vulnerability does not cause harm in itself, because there must be a threat present to exploit it. Jurjens (2002) states that in practice, security is not compromised by breaking the dedicated security mechanisms, but by exploiting the weaknesses or vulnerabilities in the way they are used. Therefore, as part of the risk analysis process, it is important to be able to determine the level of vulnerability for each risk.

INVESTIGATION PHASE *STEP 4 : Determine Level of Vulnerability*			
Risks (refer to Step 3)	**Level of Vulnerability**		
	LOW	**MEDIUM**	**HIGH**
	The asset is quite well protected against the threat, therefore the vulnerability is low.	*The asset is exposed to some degree and is not that well protected, therefore the vulnerability is medium.*	*The asset is exposed to a large degree and is not well protected at all, therefore the vulnerability is high.*
Risk A (Asset/Threat Relationship 'a')	*X*		
Risk B (Asset/Threat Relationship 'b')		*X*	
Risk C (Asset/Threat Relationship 'c')	*X*		
Risk D (Asset/Threat Relationship 'd')	*X*		
Risk E (Asset/Threat Relationship 'e')		*X*	
Risk F (Asset/Threat Relationship 'f')		*X*	
Risk G (Asset/Threat Relationship 'g')		*X*	
Risk H (Asset/Threat Relationship 'h')		*X*	

Table 6.5: Investigation Phase – Determine Level of Vulnerability

It is necessary to consider the likelihood that the risk may materialise, taking the current situation and controls into account, to determine the level of weakness or vulnerability for each risk ('A' to 'H'), as identified in Step 3. The SecSDM provides a table, as illustrated in Table 6.5, which simply maps each risk (rows) against the various levels of vulnerability (columns). The three main levels of vulnerability provided by this model are defined as:

- *LOW* indicates that the asset is fairly well protected against the threat, therefore, its vulnerability is low;

- *MEDIUM* indicates that the asset is exposed to some degree and is not that well protected, therefore, its vulnerability is medium;
- *HIGH* indicates that the asset is exposed to a large degree and is not well protected at all, therefore, its vulnerability is high.

For example, if Risk A is associated with Asset A which is relatively well protected against the associated threat (for example, 'theft of information') then the level of vulnerability may be low and the corresponding cell should be checked. Only eight cells (one per risk 'A' to 'H') must be checked on this table.

This step identifies the vulnerabilities related to each asset/threat relationship. The following step in the risk analysis process requires that a risk assessment be carried out to determine the specific extent of each risk ('A' to 'H'), identified in Step 3. This value is determined, according to the SecSDM, by taking into account the asset impact value, level of vulnerability and potential likelihood for each risk identified. This process is described in more detail in Step 5.

STEP 5 : Risk assessment

The asset impact value of the particular information asset, the level of vulnerability and the likelihood, frequency and potential impact of the particular threat must be considered, for each risk identified, to carry out a risk assessment. These are matched in a table to determine the specific measure of risk on a scale of 1 to 8. These specific values are placed in the matrix as illustrated in Table 6.6, according to those recommended by ISO/IEC TR 13335-3 (1998). The appropriate row in the table is identified by the asset impact value of the particular information asset, as identified in Step 1. Similarly, the appropriate column is identified by the likelihood, frequency and potential impact of the threat and corresponding level of vulnerability. The risk-level is determined by the likelihood, frequency and potential impact of the threat (as determined in Step 2) and the corresponding level of vulnerability (as determined in Step 4). For example, if the particular information asset, associated with the risk in question, has an asset impact value of 3, the likelihood, frequency and potential impact of the threat is 'high', and the level of vulnerability is 'low', then the measure of risk is 5. In the example provided in Table 6.6, the information asset 'Customer Orders' has an asset impact value of '4', a likelihood of the threat 'Theft of information' is 'high', and the level of vulnerability is 'low', therefore the measure of risk is '6'.

INVESTIGATION PHASE *STEP 5 : Risk Assessment*										
RISK A (as per Asset/Threat Relationship 'a' in Step 3): *Customer Orders/Theft of information*										
		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset impact value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6 *X*	7	8

Table 6.6: Investigation Phase – Risk Assessment

The specific risk values, as determined for each risk ('A' to 'H'), are valuable in assessing and prioritising those risks that require individual attention throughout the rest of the SDLC. The process of prioritising risks is discussed in Step 6.

STEP 6 : Risk prioritisation

The prioritisation of risks during the investigation phase serves as a guideline for the analysis, design and implementation phases of the SDLC. This is achieved, according to the SecSDM, by simply listing each risk ('A' to 'H'), identified in Step 3, and its corresponding risk value as determined in Step 5. The description of each risk must refer specifically to the particular asset/threat relationship, as identified in Step 3. Table 6.7 illustrates an example of the risk values that have been determined for each risk identified. The primary goal of the following phases is to document a method through which these risks may be reduced to an acceptable level.

INVESTIGATION PHASE *STEP 6 : Risk Prioritisation*		
	Risk Description (Asset/Threat relationship)	**Risk Value**
Risk A	*Customer orders/Theft of information*	*6*
Risk B	*Product Specifications/Theft of information*	*7*
Risk C	*Customer orders/Masquerading of user identity*	*6*
Risk D	*Customer orders/Malicious software attacks*	*6*
Risk E	*Employee salary details/Theft of information*	*6*
Risk F	*Product specifications/Use of system by unauthorised users*	*6*
Risk G	*Employee salary details/Use of system in unauthorised manner*	*5*
Risk H	*Product specifications/User errors*	*7*

Table 6.7: Investigation Phase – Risk Prioritisation

At the end of the investigation phase, the software developer would have identified, assessed and prioritised all potential risks pertaining to the important information assets to the software application being developed. During the analysis phase, the prioritised list of risks ('A' to 'H'), determined during the investigation phase, are mapped to the appropriate security services. This process is described in more detail in Step 7.

6.4.2 The analysis phase

The risk sensitivity of a particular software application determines the extent of the security services employed. It is meaningful for the analysis phase to focus on the security risks ('A' to 'H'), identified during the investigation phase. During the analysis phase, the appropriate security services are selected that would most likely mitigate the security risks identified. It is important, however, that this is carried out independently of any implementation details. The output of this phase is a refined set of security requirements.

The ISO 7498-2 (1989) standard provides the basis for information security in software applications through five basic security services, namely:

- *Identification and authentication*, which refers to the ability to identify and uniquely authenticate all users of a system;
- *Authorisation/access control*, which refers to the ability to allow or prohibit users from accessing the information assets of an organisation;
- *Confidentiality,* which refers to the ability to ensure that information assets are only available to those who are authorised to access them;
- *Integrity*, which refers to the ability to ensure that information is complete and uncorrupted and that it has not been altered in any way;
- *Non-repudiation/non-denial,* which refers to the ability to ensure that users do not deny their actions.

These five security services provide the basis for ensuring the security of any software application and it is important to relate them to the risks ('A' to 'H') identified during the investigation phase.

STEP 7 : Identify the relevant security services and level of protection required to minimise each risk

The analysis phase, according to the SecSDM, requires that appropriate security services be selected that will help reduce the risks ('A' to 'H'), as identified during the investigation phase, to an acceptable level.

Typically, the common threats listed in Step 2, will require the following security services (listed in brackets):

- Theft of information (identification and authentication, access control);
- Use of system by unauthorised users (identification and authentication, access control);
- Use of system in an unauthorised manner (access control, data confidentiality);
- Masquerading of user identity (identification and authentication, access control);
- Malicious software attacks (access control, data integrity);
- User errors (access control, data integrity);
- Repudiation (identification and authentication, non-repudiation);
- Technical software failures or errors (data integrity).

The software developers are required to map each of the eight risks ('A' to 'H'), as identified during the investigation phase, to the envisaged security services. For each risk, multiple security services can be identified. The particular level of protection required must be indicated by placing a B (for Basic), S (for Standard) or ES (for Extra Strong) in the relevant cells, as illustrated in Table 6.8. It must be noted, however, that not all the security services are required to address each individual risk, nor are all security services applicable to all risks.

ANALYSIS PHASE *STEP 7 : Identification of Security Services*					
Risks	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Risk A	*S*	*S*			
Risk B	*ES*	*ES*			
Risk C	*S*	*S*			
Risk D		*S*		*S*	
Risk E	*S*	*S*			
Risk F	*S*	*S*			
Risk G		*S*	*S*		
Risk H		*ES*		*ES*	

Table 6.8: Analysis Phase – Identification of Security Services

Step 7 results in the appropriate level of protection being selected to reduce the risks ('A' to 'H'), identified during the investigation phase, to an acceptable level. Section 6.4.3 describes the process of selecting the appropriate security mechanisms through which the security services identified in Step 7, should be implemented.

6.4.3 The design phase

A primary source of security problems is often an excessively complex design that cannot be easily or correctly implemented, maintained nor audited. The purpose of the design phase is to convert the information security requirements into information security specifications that can be used by programmers to develop the security-relevant code (Tompkins and Rice, 1985).

It is during the design phase of the SecSDM that the security services need to be translated into security mechanisms. These mechanisms are determined according to the

security services identified in the analysis phase. For example, if confidentiality is a required security service, then encryption can be used as the security mechanism. It is determined, during the design phase, through which security mechanisms the security services identified in the analysis phase will be implemented.

It is important to note the distinction between a security service and a security mechanism in considering the various activities of the design phase. A security service is a measure which can be put in place to address a threat (for example, the provision of confidentiality) and a security mechanism is a means to provide a service (for example, encryption). This distinction becomes more apparent when the relationship between the various security services and mechanisms is considered.

The five security services referred to by X.800 and the ISO 7498-2 (1989) standard are supported by eight security mechanisms, previously discussed in Chapter 3, namely:

- *Encipherment mechanisms* are commonly known as encryption or cipher algorithms. These mechanisms can help provide identification and authentication, data confidentiality and data integrity services;
- *Digital signature mechanisms* can be used to provide non-repudiation, origin authentication and data integrity services;
- *Access control mechanisms* can be used to provide access control services;
- *Data integrity mechanisms* can be used to provide data integrity, non-repudiation and origin authentication services;
- *Authentication exchange mechanisms*, known as authentication protocols, can be used to provide entity authentication;
- *Traffic padding* describes the addition of 'pretend' data to conceal the volumes of real data traffic. It can be used to help provide traffic flow confidentiality but is only effective when the added padding is enciphered;
- *Routing control mechanisms* may be used to provide data confidentiality and integrity;
- *Notarisation mechanisms* may be used to guarantee the integrity, origin and/or the destination of transferred data. They provide for non-repudiation.

Security mechanisms cannot be 'blindly' inserted into a software application in the hope of providing the required level of security. The overall system development process

needs to take the various security concerns and risks into consideration to ensure the appropriate use of the selected security mechanisms.

The mapping of security services to the appropriate security mechanisms is required for all eight risks ('A' to 'H'), identified during the investigation phase. This process is described in Step 8.

STEP 8 : Map security services to security mechanisms

DESIGN PHASE *STEP 8 : Mapping of Security Services to Security Mechanisms*					
RISK A (as per Asset/Threat Relationship 'a' in Step 3): *Customer Orders/Theft of information*					
Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption	*S*				
Digital Signatures					
Access Control Mechanisms		*S*			
Data Integrity Mechanisms					
Authentication Exchange	*S*				
Traffic Padding					
Routing Control					
Notarisation					

Table 6.9: Analysis Phase – Mapping of Security Services to Security Mechanisms

The software developers, with this basic knowledge and understanding of security services and mechanisms, for each risk ('A' to 'H'), as identified during the investigation phase, are required to indicate the specific security mechanisms (rows) that would be implemented to support the security services (columns), as identified in Step 7. The specific level of protection required must be indicated by checking the level protection identifier in the corresponding cells, as illustrated in Table 6.9.

It is important to note that not all security services and mechanisms are required to address each individual risk ('A' to 'H') , and it is not necessary to plot each cell in the table provided. A table, as illustrated in Table 6.9, must be completed for each of the risks identified during the investigation phase.

Step 9 describes the process of consolidating the security services and mechanisms identified in Steps 7 and 8 respectively to provide input into the implementation phase of the SDLC.

STEP 9 : Summary of findings

DESIGN PHASE *STEP 9 : Summary of Findings*																				
Security Mechanisms	**Security Services**																			
	Identification & Authentication				Access Control				Data Confidentiality				Data Integrity				Non-repudiation			
Encipherment/ Encryption	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Digital Signatures	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Access Control	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Data Integrity Mechanisms	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Authentication Exchange	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Traffic Padding	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Routing Control	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Notarisation	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H

Table 6.10: Analysis Phase – Summary of Findings

The SecSDM provides a table, as illustrated in Table 6.10, which permits the consolidation of the findings of Steps 7 and 8. Software developers are required to map the various security mechanisms (rows) to the appropriate security services (columns) for each risk ('A' to 'H'), as identified during the investigation phase. This is done by highlighting the relevant cells. For example, if Risk B requires the security service of

'identification and authentication' through the security mechanism 'encryption', then this cell is highlighted. At least one cell must be highlighted for each risk ('A' to 'H'), indicating the security mechanism through which the selected security service will be implemented. Table 6.10 illustrates that for Risk A, the security service of 'identification and authentication' will be implemented through the encryption and authentication exchange mechanisms. Similarly, it illustrates that 'access control' is implemented through access control mechanisms.

The risk sensitivity of the system is determined and the most appropriate security services and mechanisms to be employed are identified. These mechanisms need to be implemented. The security mechanisms identified can be implemented through appropriate software security tools and components, for example, those inherent in the .Net framework. The .Net framework, according to Taft (2004), provides developers with the necessary tools and information to write secure applications. A detailed discussion of the implementation phase is provided in the following section.

6.4.4 The implementation phase

The implementation phase can be referred to as the 'build and test phase'. The high-level objectives of this phase should include (Tipton and Krause, 2006):

- Build secure environments to foster system development integrity and to protect organisational assets;
- Promote secure coding practices to ensure the security quality of the end product;
- Enforce formal code review procedures to introduce checks and balances into the code development process;
- Thoroughly test all security components to validate the security design and to ensure they meet the necessary security requirements.

It is during the implementation phase that all software components are coded by the developers. Jones and Rastogi (2004) stress the importance of ensuring that the developers are knowledgeable about security risks and skilled in secure coding standards. Landoll (2006) refers to a coding standard as a set of rules and guidelines that programmers are expected to follow to increase the quality and the security of the code

produced. Any organisation that writes software applications should have a coding standard, and ensure that their employees have the necessary training to use such standards. The absence of a coding standard, according to Landoll (2006), greatly increases the likelihood that the code produced will contain security flaws. These coding standards should be reviewed for clarity, completeness and consistency.

The choice of programming language can impact the security of a software product. The best programming languages are ones where all actions are defined and reasonable features are included to reduce mistakes, where memory is managed appropriately, and where the use of pointers is discouraged. Languages like C and C++ have inherent characteristics that can lead to security vulnerabilities. Languages such as JAVA and C# have been proven to be better for developing secure software. It must be noted, however, that the use of a particular language does not guarantee or deny security. Secure applications can in theory be written in C with due care and substantial effort, and insecure applications can be written in Java and C# (Task Force Report, 2004).

The implementation of security mechanisms depends on the programming language used, the coding standards and best practices adhered to, and the personal programming style of the programmer. The programmer must ensure that all security-relevant code is understandable, auditable, maintainable and testable (Tompkins and Rice, 1985). The SecSDM does not currently recommend the use of specific .Net security components to implement the various security mechanisms but it does describe the process of mapping the security mechanisms summarised in Step 9 to the various .Net and other security components as recommended by the software developer. This is carried out in Step 10.

STEP 10 : Map security mechanisms to .Net and other security components

Siponen et al (2005) suggest that an implementation priority list is needed which indicates the priority of the security mechanisms to be implemented to ensure that the correct security features are implemented. This model provides a table, as illustrated in Table 6.11, where for each security mechanism (rows), its relevant risks must be indicated by checking the appropriate risk ('A' to 'H'). This can be transferred directly from Step 9. Software developers are encouraged to indicate the specific .Net or other components through which the various security mechanisms will be implemented. For

the purposes of this example, the security mechanism 'notarisation' has been excluded from the table because of space constraints.

An important part of the implementation phase is testing. Testing is often seen as a way of 'testing in' security which is unacceptable. The role of security testing is to verify that the system design and code can withstand attack. Testing ensures that countermeasures are correctly implemented and that code is developed following coding standards and best practices. Security testing should follow a security test plan. This test plan should include unit testing, integration testing, quality assurance testing and penetration testing (Jones and Rastogi, 2004). The testing of the software to validate that it meets the security requirements as determined during the investigation phase is essential to produce secure software. This testing should include serious attempts to attack and break its security and scan for common vulnerabilities (Task Force Report, 2004).

The dilemma arises about when to stop testing. Testing can stop when there are no known security vulnerabilities that compromise the security goals as determined during the investigation, analysis and design phases. Tipton and Krause (2006) argue that security testing differs from functional testing in the SDLC. Security testing should focus on the functions that invoke security mechanisms and on the least-used aspects of the mechanisms. This is because the least-used functions often contain flaws that can be exploited. Security testing usually includes a number of negative tests, whose expected outcomes demonstrate unsuccessful attempts to circumvent system security. Functional testing focuses on those functions that are most commonly used.

Testing procedures must be consistent, repeatable and reusable. This ensures that software can be considered to be reliably secure. Currently, reliability of a software application means that it will function as planned and will attempt to ensure the security of the application, the data, and the computing environment as a whole (Cenzic, 2003).

IMPLEMENTATION PHASE *STEP 10 : Mapping of Security Mechanisms to .Net and other components*			
Security Mechanisms	**Risks**	**.Net security components**	**Other components (including own)**
Encipherment/ Encryption	Risk A	…….	…….
	Risk B		
	Risk C	…….	…….
	Risk D		
	Risk E	…….	…….
	Risk F		
	Risk G	…….	…….
	Risk H		
Digital Signatures	Risk A	…….	…….
	Risk B		
	Risk C	…….	…….
	Risk D		
	Risk E	…….	…….
	Risk F		
	Risk G	…….	…….
	Risk H		
Access Control Mechanisms	Risk A	…….	…….
	Risk B		
	Risk C	…….	…….
	Risk D		
	Risk E	…….	…….
	Risk F		
	Risk G	…….	…….
	Risk H		
Data Integrity Mechanisms	Risk A	…….	…….
	Risk B		
	Risk C	…….	…….
	Risk D		
	Risk E	…….	…….
	Risk F		
	Risk G	…….	…….
	Risk H		
Authentication Exchange Mechanisms	Risk A	…….	…….
	Risk B		
	Risk C	…….	…….
	Risk D		
	Risk E	…….	…….
	Risk F		
	Risk G	…….	…….
	Risk H		
Traffic Padding	Risk A	…….	…….
	Risk B		
	Risk C	…….	…….
	Risk D		
	Risk E	…….	…….
	Risk F		
	Risk G	…….	…….
	Risk H		
Routing Control	Risk A	…….	…….
	Risk B		
	Risk C	…….	…….
	Risk D		
	Risk E	…….	…….
	Risk F		
	Risk G	…….	…….
	Risk H		

Table 6.11: Implementation Phase – Mapping of Security Mechanisms

Operations staff are typically responsible for the entire computing environment, while developers must consider security throughout the software development process. The introduction of a new software application can create significant risk to this environment. Security testing is one way of ensuring that applications are developed in a way to minimise this risk, so that they can be implemented with a higher degree of confidence. Developers must test and retest for security to learn new coding techniques and quickly gain the benefits associated with strong security development procedures (Cenzic, 2003).

6.4.5 The maintenance phase

Gregory (2003) views the maintenance phase as another iteration of the entire life cycle. During this phase, it is important to find ways to evaluate the security of the system to ensure that the system is as secure as intended. New exposures may develop as users, data and equipment change. Those users affected by the security features that have been implemented must be educated and trained to ensure their commitment.

The goal state defined for the software application must be audited to ensure that it complies with policies, standards and regulations used by an organisation. A security report should contain findings of the audit and security tests. System documentation is created during any software development process. This is particularly necessary for security related code. It is advisable to keep the documentation of the security related code in a separate development document, owing to its sensitivity. The sensitivity of this code and documentation is such that it should be well protected and access to it restricted (Booysen and Eloff, 1995).

The SecSDM ensures that all relevant security-related information is kept in a security portfolio, as provided in Appendix A. This helps improve the auditability of the software application in question, because security-related decisions are traceable to the appropriate phase as proposed by this secure software development approach. The integration of information security into the SDLC as described in this section, and the tight integration between the various phases will help ensure that the final product meets the information security requirements, identified during the initial phases.

6.5 Conclusion

There are very few structured approaches to effectively integrate information security and software development. It is not easy for a software developer to have an expert knowledge of all the available security tools and their particulars, so as to choose the most appropriate ones for a specific software application under development. There is, therefore, an evident need for a framework to help determine the most appropriate security features and components to be incorporated into software applications (Tryfonas and Kiountouzis, 2002).

Organisations and companies are starting to recognise the importance of security in the life cycle as an integrated end-to-end process. In the past, integrating security into the software life cycle process has typically been haphazard. It requires both trained experts and dedicated resources. Securing software requires an extensive body of knowledge unknown to developers. Good programmers try to produce high quality software but they most often lack the knowledge and training necessary to develop, assess and/or improve the reliability, safety and security of software. A software development team that is not aware about delivering secure systems, will not deliver a secure product. Similarly, a software development team that follows a process that does not encompass good security discipline will not deliver a secure product.

It is necessary to build an improved software development process that integrates security during all stages to build better and more secure software. McGraw (2003), states that the key to building secure and reliable software lies in developing software under an iterative risk management process and applying tools and processes in a manner that is consistent with the business purpose of the software itself. Changing the process to deliver more secure software, however, is not the most difficult challenge. The real challenge is changing the perceptions and attitudes of all those involved in the software development process.

Software development does not occur in isolation. It is necessary to involve concerned users, system developers and information security specialists. Security is tightly interwoven in the software development process by applying the SecSDM. Software

developers are encouraged to consider security from the earliest point of the SDLC, and to build critical security milestones and events into their development timelines.

Security, as with quality, should be viewed as an integral part of the SDLC and not just as an add-on at the end of the development process. It is imperative that security is a well-thought-out process at all stages, from system inception and design through implementation and deployment. Any failure to consider information security adequately during the design phase of systems development can result in implementation vulnerabilities. The types of controls that will ultimately be incorporated into a software application should be determined based upon the potential loss or harm that could be suffered if the data or application were modified, destroyed or disclosed or is caused to become unavailable due to unauthorised or undesirable events (Tompkins and Rice, 1985).

Tipton and Krause (2006) stress the importance of software developers being educated in secure coding practices to ensure that the end product has the required security functionality. However, it is argued that this alone will not guarantee the security of a software application. Software developers need to be educated in information security to avoid adding security features for its sake. Security is a feature and will have an impact on both the cost and schedule of the project. However, Howard and LeBlanc (2003) suggest that like any feature, the later it is added in, the higher the cost and the higher the risk to the project schedule. It is possible to predict the schedule impact by performing a security risk analysis early in the SDLC.

Meaningful security is easier to achieve when security issues are considered as part of a routine development process, and safeguards are integrated into the software application during its design. A retrofit security is undoubtedly more expensive than its integration into the software development process from the beginning. Similarly, safeguards that are integral to a software application, are usually easier to use and less visible to the user (Booysen and Eloff, 1995).

The SecSDM, as discussed in this chapter, was tested at a South African university using third-year Information Technology students to establish its effectiveness. These

students were encouraged to integrate the various steps, as described by this model, into their software development projects. Chapter 7 addresses some of the key findings.

Chapter 7

Secure Software Development in Practice

7.1 Introduction

Over the years, it has become increasingly concerning that software developers tend to ignore the idea of integrating security into software applications. For this reason, the Secure Software Development Model (SecSDM), as described in Chapter 6, was developed. However, for any new model, it is necessary to determine whether it actually facilitates the achievement of the primary goal for which it was established. In this case, the development of more secure software applications.

To establish the effectiveness of the SecSDM, results were gathered from both 2005 and 2006 third year Information Technology (IT) project students at a South African university. The study took the form of a questionnaire and was completed by two groups of students. The first group consisted of 2005 project students, while the second group consisted of 2006 project students. All variables were kept consistent as far as possible, except for the fact that the 2006 project students were introduced to, and worked through, the SecSDM. Therefore, the study attempted to evaluate the effectiveness of the SecSDM.

Some of the typical applications developed by the 2005 and 2006 project students, as part of their practical course component, included:

- Managing Bookings and Information Online (for example, the tourism industry, computer laboratories, sports clubs);
- Student Accommodation System;
- Managing Patient Information and Appointments for Health Services;
- Biometric Security System;
- Stock Control and Tracking System;
- Supply Chain Management System;
- Workflow System for Handling Petitions;
- Event Planning System;

- Online Payment System;
- Online Voting System;
- Online Applications for Tertiary Institutions;
- Online Gaming System;
- Online Information System for the Department of Environmental Health;
- Internet-Enabled Robotics Laboratory;
- Small Business Application (Point-of-Sale, Debtors, Creditors, Stock Control);
- Retail Management System;
- Spatial Information Management Systems;
- Facilities Management System.

A total of 31 students participated in the study in 2005. It is important to note, however, that for these students, information security was not specifically emphasized, or taught, during the requirements gathering, analysis, design nor implementation phases of their projects. The 2006 students, on the other hand, were introduced to the SecSDM during the course of the academic year. These students were encouraged to integrate the various steps, as described by the SecSDM, into their development projects. A total of 20 students participated in the study in 2006.

Although the projects developed at third-year level are typically group projects, each individual was required to take responsibility for a specific sub-system. Therefore, the perception and understanding of the various individuals within a group, with respect to the security of the application, could differ. Consequently, it was necessary that each individual, within each group, complete the information security questionnaire, as presented in Appendix B.

The objective of the study was to establish whether the SecSDM, actually encouraged students to integrate information security into their software development projects, at the same time providing them with a deeper insight and understanding of the various security aspects that may have impacted their applications.

The following section describes the questionnaire that was completed by both the 2005 and 2006 students.

7.2 The Information Security Questionnaire

The primary aim of the information security questionnaire was to establish the extent to which IT project students consider and incorporate information security into their development projects.

The questionnaire was divided into five sections, with each section focussing on a particular phase, or phases, of the SecSDM as follows:

- Section 1 focused on questions relating to the investigation phase;
- Section 2 addressed the analysis phase;
- Section 3 was concerned with both the design and implementation phases;
- Section 4 dealt with the maintenance phase;
- Section 5 aimed to determine general opinions of the students with respect to information security.

The following sub-sections briefly describe each section of the questionnaire, as presented in Appendix B.

7.2.1 Section 1 – The investigation phase

The purpose of this section was to determine whether the students performed any form of risk analysis, either formal or informal, during the life cycle of their projects. Therefore, the first three questions related to the identification and assessment of assets, threats and vulnerabilities. The fourth question pertained directly to the extent to which a risk analysis was carried out. Respondents simply indicated their answer by circling 'Yes', 'No' or 'Not Sure' for each of the questions posed. Those respondents who indicated that they had performed some form of risk analysis, were encouraged to provide brief details in this regard.

7.2.2 Section 2 – The analysis phase

The primary aim of this section was to determine whether the students included any of the five security services within their applications. Respondents simply indicated their answer by circling 'Yes', 'No' or 'Not Sure' for each of the questions posed. For those respondents who indicated that they had included a specific security service, it was

necessary to determine whether this was purely for the sake of including the service, or whether the reason for inclusion related to a specific risk anticipated. If based on a formal reason, respondents were encouraged to provide brief details in this regard. Otherwise, respondents were required to provide a motivation for its inclusion.

7.2.3 Section 3 – The design and implementation phases

The main intention of this section was to determine whether the students incorporated any of the eight security mechanisms within their applications. Respondents simply indicated their answer by circling 'Yes', 'No' or 'Not Sure' for each of the questions posed. For those respondents who indicated that they had included a specific security mechanism, it was necessary to determine whether this was implemented through the use of a .Net component, some other component, or whether they wrote their own component. This was indicated by circling either 'a', 'b' or 'c' respectively. For those respondents who indicated that they had included a specific security mechanism, it was necessary to determine whether they perceived the selected component to be "strong" enough to protect the associated assets, or not.

7.2.4 Section 4 – The maintenance phase

Section 4 specifically addressed questions pertaining to user procedures, documentation and auditing. Respondents simply indicated their answers by circling 'Yes', 'No' or 'Not Sure' for each of the questions posed. Those respondents who indicated that they had some form of user education process in place, with respect to using the application in a secure way, were required to provide relevant details. The same applied to those students who indicated that they had produced any form of security-related documentation.

7.2.5 Section 5 – General opinions

The questions posed in this section were of a general nature. The respondents were required to indicate their opinion with respect to the extent to which security services and components had been considered and integrated into their software applications. This was indicated by simply circling 'a' for 'too few', 'b' for 'too many' or 'c' for 'adequate'. Lastly, respondents were asked whether they believed there to be a need for a new or improved methodology for integrating information security into their software

development projects. Once again, respondents simply indicated their answers by circling 'Yes', 'No' or 'Not Sure'.

7.3 Results of Information Security Questionnaire

The results of the questionnaire, as discussed in Section 7.2, provides important information to support the need for a new or improved methodology for integrating information security into software development projects, and to motivate improvements to the SecSDM. In interpreting the results, it is important to bear in mind that, whereas the 2005 students simply followed the more traditional SDLC, the 2006 students were provided with an overview to information security and introduced to the SecSDM. In addition, the hand-in of an information security portfolio (see Appendix A), reflecting each security deliverable as defined by the SecSDM, formed an essential part of their system documentation. The detailed results, comparing the results from the 2005 and 2006 project groups, are provided in Appendix C.

7.3.1 The investigation phase

During the investigation phase, it was necessary to determine the security requirements of the software application under development. According to the SecSDM, this may be achieved by identifying the most important information assets, threats and vulnerabilities pertaining to the application in question. The results relating to the investigation phase are illustrated in Figure 7.1.

Figure 7.1 clearly illustrates an increase of 56% from 2005 (39%) to 2006 (95%), in the number of respondents who indicated that they had identified and assessed the important information assets associated with the application under development. Similarly, increases of 35% and 28% were recorded for the identification of threats and vulnerabilities respectively. Despite these significant increases, however, the number of respondents who indicated that they had carried out some form of risk analysis, only increased by 14%. This possibly suggests that many of the respondents were not aware that the identification of assets, threats and vulnerabilities, actually formed part of a risk analysis.

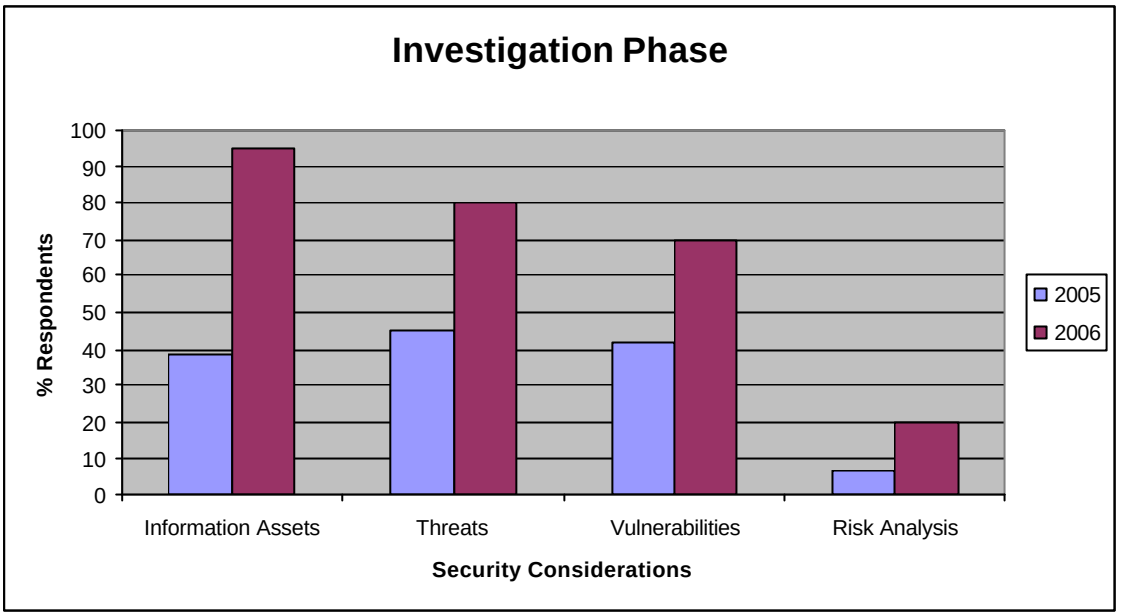

Figure 7.1: Integration of Security into the Investigation Phase

7.3.2 The analysis phase

The questions pertaining to the analysis phase were aimed at determining the extent to which students incorporated the five security services into their software applications. The results relating to this phase are illustrated in Figure 7.2.

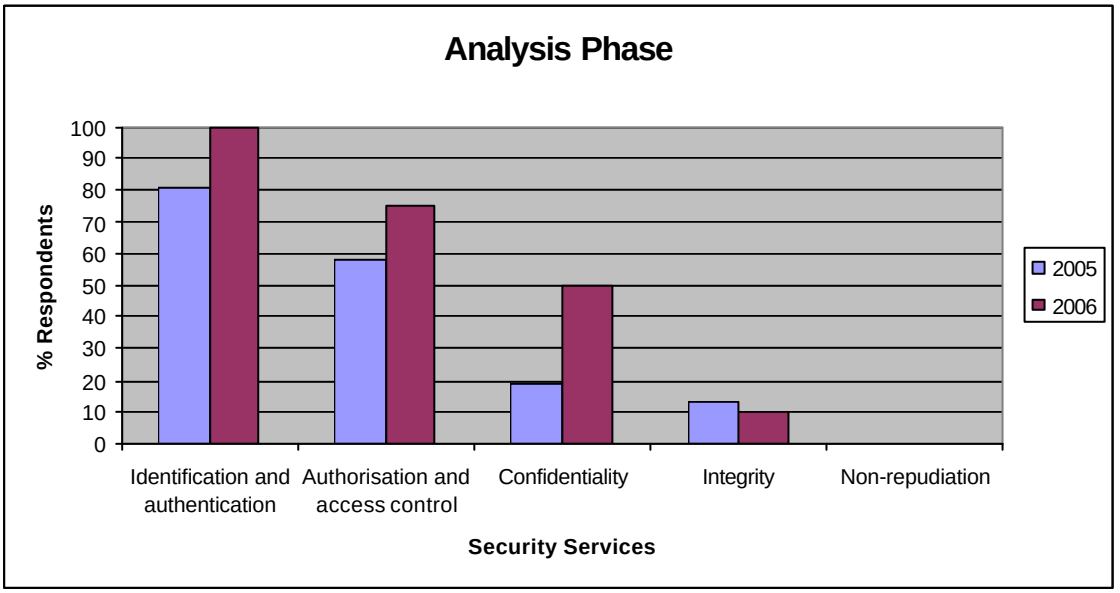

Figure 7.2: Integration of Security into the Analysis Phase

From Figure 7.2, it can be seen that significant increases were recorded from 2005 to 2006 for the following security services:

- Confidentiality (31%);
- Identification and authentication (19%);
- Authorisation and access control (17%).

This suggests that many more students incorporated these three security services into their software applications in 2006 than in 2005. However, the results for integrity and non-repudiation were not significant. In addition, the identification of each of the security services could be related to a specific reason identified in the investigation phase.

7.3.3 The design and implementation phases

The questions relating to the design and implementation phases aimed to establish the extent to which the students included the eight security mechanisms within their software applications. For those students who indicated that they had included a particular security mechanism, it was necessary to determine whether it was implemented using a .Net or other component. The results relating to the investigation phase are illustrated in Figure 7.3.

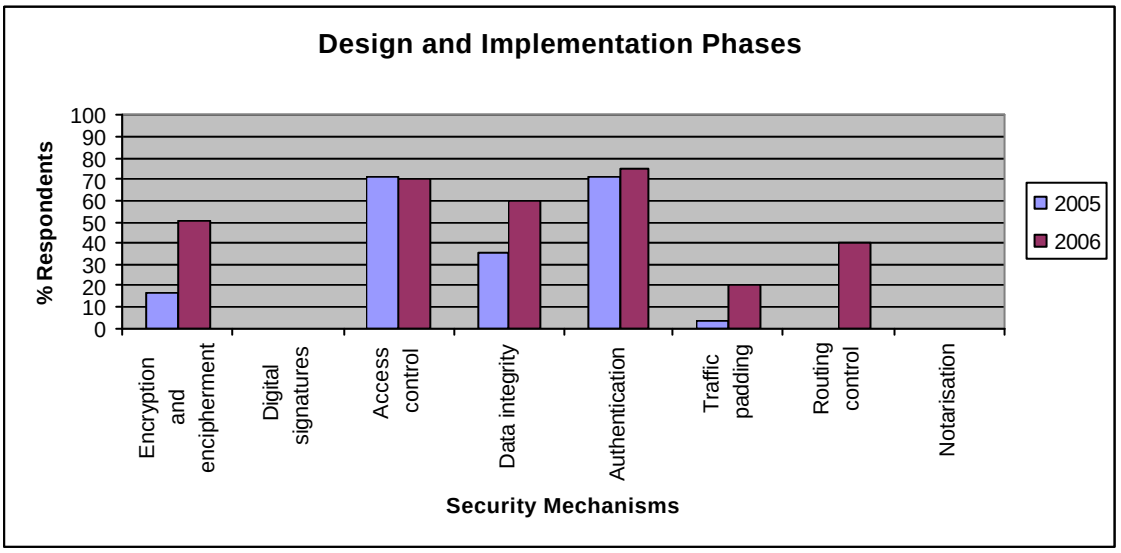

Figure 7.3: Integration of Security into the Design and Implementation Phases

Significant increases were recorded from 2005 to 2006 for the following security mechanisms:

- Routing control (40%);
- Encryption and encipherment (34%);

- Data integrity (25%);
- Traffic Padding (17%).

It is necessary, however, to interpret these results in relation to the figures recorded in 2005. The figures recorded in 2005 for traffic padding (3%) and routing control (0%) were significantly low. This contributed towards the significant increases of 17% and 40% respectively.

On the other hand, if one considers the access control and authentication security mechanisms, a decrease of 1% and an increase of 4% was recorded from 2005 to 2006 for each respectively. These insignificant results may be contributed to the fact that these are the two security mechanisms that students traditionally include in their software applications, with figures of 71% being recorded for each in 2005.

The figures recorded for digital signatures and notarisation for both 2005 and 2006 was 0. The reason for the non-inclusion of these two security mechanisms, could possibly be attributed to the nature of the software applications that the students develop, and the lack of knowledge that students may have with respect to these particular security mechanisms.

For encryption and encipherment, the results suggest that students tend to implement this security mechanism using .Net components. However, a number of students indicated that they wrote their own components for many of the other security mechanisms. In terms of the actual implementation of the various security mechanisms, the results suggest that when using .Net components, the students are fairly confident that the selected components are "strong" enough to protect the associated information assets.

7.3.4 The maintenance phase

The questions relating to the maintenance phase were concerned primarily with whether any user procedures had been defined for using the software application in a secure manner, whether any security-related detail had been documented, and whether or not it would be possible to audit the information security aspects of the software application developed. The results relating to the maintenance phase are illustrated in Figure 7.4.

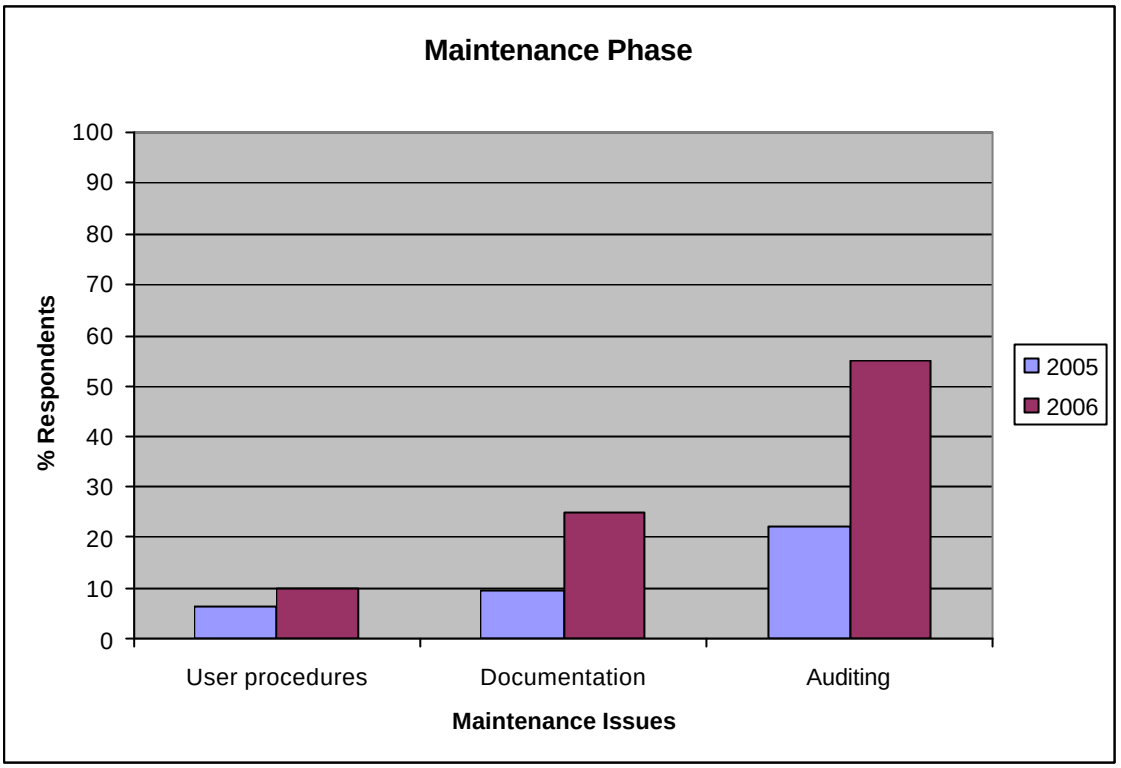

Figure 7.4: Integration of Security into the Maintenance Phase

The main concern in terms of the figures recorded, was that very few students tend to pay attention to including user procedures for using the application in a secure way. Figures recorded in this regard were 6% in 2005 and 10% in 2006. This could be attributed to the fact that very few of the applications developed are actually commercialised nor used in a "live" environment upon completion.

In addition, the number of respondents who indicated that they had documented some security-related detail with respect to their applications, was 10% in 2005 and 25% in 2006. The main concern was that many of the 2006 students responded negatively in this regard, although all groups completed the information security portfolio, as shown in Appendix A. This suggests that students did not take into account the fact the information security portfolio is a security-related document within itself.

However, a significant increase of 32% was recorded from 2005 to 2006 for the question relating to auditing. This suggests that many more students felt confident that the information security aspects of their applications were auditable to determine

whether too few, too many or adequate security services and components had been considered and integrated into their applications.

7.3.5 General Opinions

The general security questions posed were based on the students' opinions as to whether they perceived there to have been too few, too many or adequate security services and components considered and integrated into their software applications. In addition, they indicated whether they recognised the need for a new or improved methodology for integrating information security into their software development projects. The results relating to these opinions are illustrated in Figure 7.5.

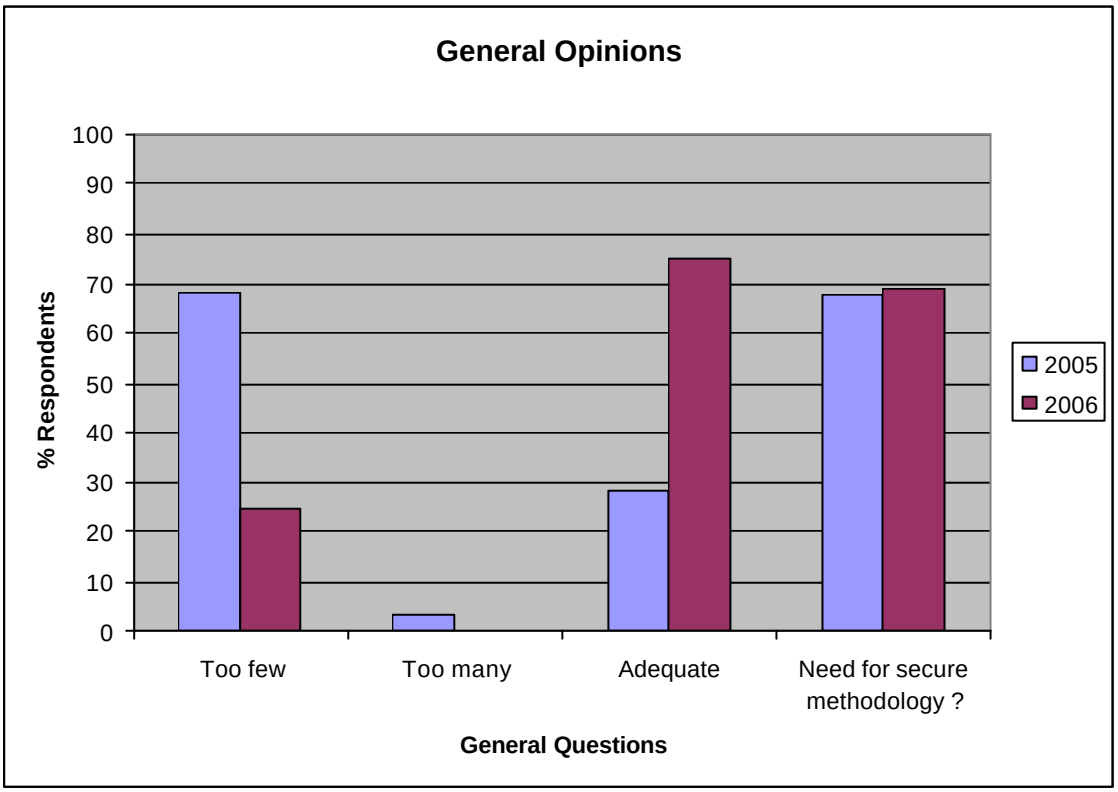

Figure 7.5: General Opinions

A significant decrease of 43% was recorded from 2005 (68%) to 2006 (25%) for those students who perceived there to be too few security services and components considered and integrated into their projects. Consequently, there was a significant increase of 47% for those students who perceived their projects too have adequate security services and components. Although the increase from 2005 (68%) to 2006 (69%) was insignificant,

it is encouraging that a relatively high percentage of students recognised the need for a new or improved methodology for integrating information security into their projects.

7.4 Significant Findings

Some of the key findings with respect to how each step of the model was carried out by the respondents include:

- *STEP 1: Information asset identification and valuation;*

  Respondents were not specific with respect to the identification of information assets. Several respondents, for instance, indicated the database, in totality, as a key information asset. This would imply that all tables within the database have the same security requirements. This is not normally the case since encrypting the entire database, for instance, would have severe performance implications. The valuation of the information assets identified, on the other hand, was well understood by the respondents.

- *STEP 2: Threat identification and assessment;*

  Respondents could identify well with the various threats, and therefore were able to complete this section with relative ease. No additional threats were identified by the students.

- *STEP 3: Risk (asset/threat) identification;*

  It is clear that several respondents misinterpreted this step as, in many cases, more than eight blocks were labelled ('a' to 'h) when identifying the most critical risks (i.e., asset/threat relationships). This resulted in duplicate labels which negatively impacted on the steps that followed.

- *STEP 4: Determine the level of vulnerability;*

  The key problems here resulted primarily from the problems inherited from Step 3. For instance, where duplicate labels were provided when identifying the key asset/threat relationships in Step 3, it was difficult for students to correctly transfer these to Step 4.

- *STEP 5: Risk assessment;*

  A number of respondents neglected to consider the asset as an essential part of the risk identified. For these respondents, they were then unable to transfer the appropriate asset value from Step 1.

- *STEP 6: Risk prioritisation;*

  As with Step 5, a number of respondents neglected to consider the asset as an essential part of the risk identified. However, respondents generally understood the concept of simply transferring the risk and risk value from Step 5.

- *STEP 7: Identify the relevant security services and level of protection required to mitigate each risk;*

  The tendency here was for respondents to haphazardly label blocks B (for Basic), S (for strong) and ES (for extra strong), without taking the typical mappings, as supplied in the table provided, into consideration.

- *STEP 8: Map security services to security mechanisms;*

  As with Step 7, the tendency was for respondents to haphazardly label blocks B (for Basic), S (for strong) and ES (for extra strong), without taking the typical mappings, as supplied in the table provided, into consideration.

- *STEP 9: Summary of findings;*

  Respondents who had worked logically through the model, were able to complete this section with ease since it simply required them to transfer the recommendations from Step 8.

- *STEP 10: Map security mechanisms to .Net and other security components;*

  Although most respondents were able to indicate the specific risks for which each security mechanism should be implemented, very few actually listed the components used.

It is clear that the use of 'A', 'B, 'C', 'D' and 'E' to indicate the most important information assets and the use of 'a', 'b', 'c', 'd', 'e', 'f', 'g' and 'h' to indicate the most critical asset/threat relationships caused some confusion for the respondents. This was made increasingly confusing by referring to each risk as 'A', 'B', 'C', 'D', 'E', 'F', 'G' and 'H'. Therefore, it is recommended that in further improvements to the model, that assets be referred to as A1, A2, A3, A4 and A5 respectively. Similarly, that the common threats be referred to as T1, T2, T3, etc. and the most critical risks as R1, R2, R3, etc. This would mean that a particular risk (for example, R1) would be associated with a specific asset (for example, A1) and a common threat (for example, T3).

Very few respondents provided details where requested. However, the main areas of concern for most students, with respect to security, appeared to have been the

identification and authentication of users, as well as access control. This was generally implemented through the use of passwords to identify the particular users of the application. Access to various components or features was then restricted based on the role of these users. The tendency was for most students to store these passwords in the database in an encrypted form, and not in plain text. Although this is encouraging, it is important that students consider other aspects of security as promoted by the SecSDM.

7.5 Conclusion

The results from the information security questionnaire, as discussed in this chapter, clearly suggest that by following the SecSDM, developers were forced into a more structured approach to incorporating security into their software applications. In addition, most security-related decisions taken during the SDLC could be related back to the previous phases. Although problems do exist with the SecSDM in its current form, it is believed that with a few improvements, it could become very effective in ensuring that security concerns are not added as an afterthought during the implementation phase, but rather based on a specific risk anticipated during the investigation phase.

Chapter 8

Conclusion

8.1 Introduction

The increasing growth and use of the Internet to conduct business has brought about great changes in the computing environment for businesses, organisations and individuals. The risks to all have increased by these new technology enriched environments which have become part of our everyday lives.

Intense time-to-market requirements has put very real pressure on organisations to produce software faster, which reduces the opportunity and incentive to ensure the software is robust and reliable. The real source of the software security problem, however, is the lack of integration between security and development. This means that typically developers do not understand security and security professionals do not understand software development. Developers see security as a hurdle, while security professionals see software development as reckless (McGraw and Viega, 2001).

Information security is typically defined as the preservation of confidentiality, integrity and the availability of information. While confidentiality ensures that information is accessible only to those authorised to have access, integrity is concerned with safeguarding the accuracy and completeness of such information and any related processing methods. Information is an asset which, like other business assets has value and therefore needs to be suitably protected. Information security protects information from a wide range of threats to ensure business continuity, minimise business damage and maximise return on investments and business opportunities.

The Task Force Report (2004) states that the security of software may be increased by:

- Enhancing the education and training of present and future software developers to put security at the heart of software design and at the foundation of the development process;
- Developing, sharing and skilfully using processes and practices to improve the quality and security of software, so that systems are more resilient to attack.

Therefore, the quality of software applications depends on an adequate supply of proficient and up-to-date software developers. Professional software developers must learn to create software that is sufficiently trustworthy to be used by non-professionals. To prepare software developers for successful careers in software development, they must not only be skilled in the various aspects of software development, but also in information security. In this way software developers will be better equipped to produce secure software applications that end users will be able to trust and use with confidence. The SecSDM, as described in this dissertation, will provide software developers with a structured approach to integrating security into their software applications.

8.2 Summary

Software plays a key role in the modern world and is expected to perform at high levels of reliability and security. However, it is evident that there exists a lack of trust with respect to software applications, especially in the highly networked environment that has become part of our everyday lives. Since people are increasingly concerned about the security and reliability of current information systems, they are reluctant to entrust them with their personal information. Not surprisingly, users are demanding more secure software applications.

The reality, however, is that information security is an afterthought for many software developers. This means that information security does not normally form an integral part of the software development process. From the discussion in Chapter 2, it is clear that security must be strongly coupled with the software development process to ensure that the desired level of quality is achieved. This, in turn, should result in people becoming more confident to use these applications. Therefore, by carefully considering the security aspects of software applications, one can increase their safety and effectiveness in meeting the expectations of their users.

Chapter 3 discusses the various standards and best practices relating to information security and software development (for example ISO 7498-2, ISO/IEC 17799 and ISO/IEC TR 13335). The ISO 7498-2 (1989) standard played a key role in the development of the SecSDM as it specifically describes the Basic Reference Model for Open Systems Interconnection (OSI). This standard provides the basis for information

security through five security services. These, in turn, are supported by eight security mechanisms. The ISO/IEC 17799 (2005) standard, on the other hand, provides best business practice, guidelines and general principles for implementing, maintaining and managing information security. Although ISO/IEC TR 13335 also provides guidelines for the management of IT security, ISO/IEC TR 13335-3 (1998) specifically advocates a structured approach to performing a detailed risk analysis. This approach forms the basis of the investigation phase of the SecSDM.

Chapter 4 addresses the fact that existing software development methodologies neglect to provide a structured approach to integrating security into each stage of the software development process. This severely impacts the quality and trustworthiness of the final product. Therefore, it is important that reliability, security and performance be considered throughout all phases of the SDLC. Small changes in the SDLC could substantially improve the security, reliability and trustworthiness of the software application without incurring significant overhead.

To secure software applications, it is important to adopt a disciplined approach that incorporates all aspects of software development, from requirements gathering through to design, implementation and maintenance. An important feature of the SecSDM, as described in Chapter 6, is that it provides a comprehensive process that not only encompasses the five basic phases of software development, but it also provides a formal approach to integrating security into each of these phases.

Chapter 5 further suggests that the earlier in the SDLC that risks are identified, the better. A detailed risk analysis involves the identification and valuation of assets, the assessment of threats to those assets, and an assessment of the associated vulnerabilities. The results of these risk analysis activities are then used to assess the specific risks pertaining to the particular software application, and consequently justify appropriate safeguards to minimise these risks. The SecSDM, as described in Chapter 6, explicitly dictates that a risk analysis be performed during the investigation phase of the SDLC. The results of this activity provide critical input to the further phases of the SDLC, thereby facilitating a more secure software product.

To conclude this research project, it was necessary to carry out an initial study to test the effectiveness of the SecSDM in ensuring the development of a secure software product. This study took the form of a questionnaire. The results of this study are discussed in Chapter 7.

8.3 Meeting the Objectives

A literature study was carried out to develop an understanding of software quality and Trustworthy Computing and, in particular, how these relate to information security and software development. In this way, the secondary objective pertaining to this research topic was achieved in Chapter 2.

Similarly, a literature study was undertaken to gain the knowledge required to support the development of the SecSDM, and to ensure that it conforms to current standards and best practices. Based on the standards and best practices studied, a set of criteria for secure software development was established. Chapter 3 addresses those pertaining to information security and software development, with a specific focus on the ISO 7498-2 (1989) standard, the ISO/IEC 17799 standard and the ISO/IEC TR 13335 technical report. Therefore, it can be argued that the secondary objective relating to standards and best practices was realised.

To develop a model for secure software development, it was necessary to undertake an assessment of current software development methodologies. Special attention was given to the inherent weaknesses of the methodologies studied, especially in terms of the way in which they provide for security. Therefore, the secondary objective pertaining to the SDLC was met through the discussions provided in Chapter 4.

The identification and valuation of assets, the assessment of threats to those assets, and an evaluation of the associated vulnerabilities is necessary to ensure the security of a software application. Therefore, a further literature study was undertaken to gain an understanding of risk analysis and its importance as the first and most critical stage of a secure software development model. Therefore, it can be argued that this secondary objective was met through the risk analysis discussion in Chapter 6.

An information security questionnaire, as provided in Appendix B, was completed by 2005 and 2006 students at a South African university. Chapter 7 reports on these results and provides evidence that the SecSDM can work in practice. The secondary objective pertaining to this analysis was therefore achieved through the evidence provided.

It is argued that the primary objective of this research project was met through the development of a secure software development model as described in Chapter 6. The secondary research objectives afforded essential input into the development of this model. The SecSDM is based on the ISO 7498-2 (1989) standard, which deals with the underlying security services and mechanisms that form an integral part of the model. The risk analysis advocated by this model is based on the approach as suggested by the ISO/IEC TR 13335-3 (1998) technical report. In addition, the SecSDM conforms to various guidelines and principles as advocated by ISO/IEC 17799 (2005) and ISO/IEC TR 13335-3 (1998).

8.4 Future Research

Future research is required to further test the effectiveness of the SecSDM. It is recommended that this be carried out at other tertiary institutions, and within the software development industry itself. The results of this research will provide the input required to make further adjustments and enhancements to this model. It is hoped that the final result will be a secure software development model that can be used both for educating software developers in secure software development, and in the actual development of secure applications.

List of References

Anderson, A. M. (1991). Comparing risk analysis methodologies. In *Proceedings of IFIP. Information Security.* North Holland.

Bertine, H., Chadwick, D., Euchner, M. & Harrop, M. (2004, Oct). *Security in telecommunications and information technology* (Technical Report). International Telecommunication Union.

Bessin, G. (2004, Jun). *The business value of software quality.* (http://www.106.ibm.com/developerworks/rational/library/4995.html).

Booysen, H. A. & Eloff, J. H. (1995). A methodology for the development of secure application systems. In *Proceedings of IFIP. Information Security – the next decade.* (pp. 230-243). Chapman and Hall.

Breu, R., Burger, K., Hafner, M. & Popp, G. (2004, May). Towards a systematic development of secure systems. *Systematic Development*, pp.5-13.

Cenzic. (2003, Oct). *Enabling security in the software development life cycle* (Technical Report).

Cho, S. & Ciechanowicz, Z. (2001). Checklist-based risk analysis with evidential reasoning. In *Proceedings of IFIP. Trusted information: the new decade challenge.* Kluwer Academic Publishers.

Friedman, M. & Wlosinski, L. (2003, May). *Integrating security into the systems development life cycle* (Technical Report). Centre for Information Technology.

Futrell, R. T., Shafer, D. F. & Shafer, L. I. (2002). *Quality software project management*. Prentice Hall.

Galin, D. (2004). *Software quality assurance from theory to implementation*. London : United Kingdom: Pearson Education Limited.

Gong, B., Yen, D. C. & Chou, D. C. (1998). A manager's guide to total quality software design. *Industrial Management and Data Systems*, pp.100-107.

Gregory, P. H. (2003). *Security in the software development life cycle* (Technical Report). The Hart Gregory Group Inc.

Security Risk Analysis Group (2003). *Introduction to risk analysis.*

Haley, C. B., Laney, R. C. & Nuseibeh, B. (2004, March). Deriving security requirements from cross-cutting threat descriptions. In *Proceedings of the 3rd international conference on aspect-oriented software development* (pp. 112-121). New York : United States of America: ACM Press.

Howard, M. & LeBlanc, D. (2003). *Writing secure code : Practical strategies and techniques for secure application coding in a networked world*. Microsoft Press.

ISO. (1989). *ISO 7498-2: Information Processing Systems - Open System Interconnection - Basic Reference Model - Part 2: Security Architecture.*

ISO. (1995). *ISO/IEC 12207 : Information Technology - Software Life Cycle Processes.*

ISO. (1998). *ISO/IEC TR 13335-3 : Information Technology – Guidelines for the Management of IT Security. Part 3 : Techniques for the management of IT security.*

ISO. (2000). *ISO/IEC TR 13335-4 : Information Technology – Guidelines for the Management of IT Security. Part 4 : Selection of safeguards.*

ISO. (2004). *ISO/IEC 13335-1 : Information Technology - Security Techniques - Management of Information and Communications Technology Security. Part 1: Concepts and models for information and communications technology security management.*

ISO. (2005). *ISO/IEC 17799 : Information Technology - Code of Practice for Information Security Management.*

Jones, R. L. & Rastogi, A. (2004, Nov). Secure coding - building security into the software development life cycle. *Application Program Security,* pp.29-38.

Josang, A., van Laenen, F., Knapskog, S. J. & Vandewalle, J. (1997). How to trust systems. In *Proceedings of IFIP*. *IT Security in Research and Business.* Chapman & Hall.

Jurjens, J. (2002, May). Using UMLSec and goal trees for secure systems development. *Communications of the ACM, 48* (5), pp.1026-1030.

Karabacak, B. & Sogukpinar, I. (2005). Isram : Information security risk analysis method. *Computers and Security, 24*, pp.147-159.

Killmeyer, J. (2006). *Information security architecture : An integrated approach to security in the organisation*. New York : United States of America: Auerbach Publications.

Kuppuswami, S., Vivekanandan, K., Ramaswamy, P. & Rodrigues, P. (2005). The effects of individual XP practices on software development effort.

Landoll, D. J. (2006). *The security risk assessment handbook : A complete guide for performing security risk assessments.* New York : United States of America: Auerbach Publications.

Laudon, K. C. & Laudon, J. P. (2004). *Information systems and the internet : A problem-solving approach.* Orlando : United States of America: Dryden Press.

Leiwo, J., Kwok, L., Maskell, D. L. & Stankovic, N. (2004, May). A technique for expressing it security objectives. *Information and Software Technology,* pp.31-37. (Available online at www.sciencedirect.com)

Lipner, S. & Howard, M. (2005). The trustworthy computing security development lifecycle. *27*. (cited on 15th April 2005)

Maciaszek, L. A. (2001). *Requirements analysis and system design : Developing information systems with UML*. Pearson : Addison Wesley.

Main, A. (2004, May). Application security - building in security during the development stage. *Application Program Security*, pp.31-37.

McGraw, G. (2003). Making essential software work : Why software quality management makes good business sense.

McGraw, G. & Verdon, D. (2004). Risk analysis in software design. *IEEE Security and Privacy*, pp.32-37.

McGraw, G. & Viega, J. (2001, Aug). Securing software : Practice safe coding.

Microsoft. (2005). *Overview of information security at Microsoft* (Technical Report). Microsoft Corporation.

Mundie, C., de Vries, P., Haynes, P. & Corwine, M. (2002, Oct). *Trustworthy computing - a Microsoft white paper* (Technical Report.). Microsoft Corporation.

Nerur, S., Mahapatra, R. & Mangalara, G. (2005, May). Challenges of migrating to agile methodologies. *Communications of the ACM, 48* (5), pp.73-78.

NIST (1996, Sept). Generally accepted principles and practices for securing information technology systems. *NIST Special Publication 800-14*. (http://csrc.nist.gov/publications/nistpubs/800-14/800-14.pdf).

NIST (2004, June). Security considerations in the information system development life cycle. *NIST Special Publication 800-64*. (http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf).

NIST (2004, June). Engineering principles for information technology security: a baseline for achieving security. *NIST Special Publication 800-27 Revision A*. (http://csrc.nist.gov/publications/nistpubs/800-27A/SP800-27-RevA.pdf).

Nyanchama, M. (2005, July). Enterprise vulnerability management and its role in information security management. *Information Security Management*, pp.29-56.

O'Brien, J. A. (2002). *Management information systems - managing information technology in the e-business enterprise* (Fifth ed.). McGraw Hill.

Peltier, T. R. (2005). *Information security risk analysis*. New York : United States of America: Auerbach Publications.

Peters, J. F. & Pedrycz, W. (2000). *Software engineering : an engineering approach*. Wiley.

Pfaffenberger, B. (2002). *Computers in your future*. New Jersey : United States of America: Prentice Hall.

Post, J. & Anderson, D. L. (2003). *Management information systems : Solving business problems with information technology*. McGraw Hill.

Security Risk Analysis Group (2003). *Introduction to risk analysis.*

Schultz, D. J. (2000). A comparison of five approaches to software development.

Siponen, M. (2006, August). Information security standards focus on the existence of process, not its content. *Communications of the ACM, 49* (8), pp.97-100.

Siponen, M., Baskerville, R. & Kuivalainen, T. (2005). Integrating security into agile development methods. In *Proceedings of the 38th Hawaii International Conference on System Sciences.*

Spinellis, D., Kokolakis, S. & Gritzalis, S. (1999). Security requirements, risks and recommendations for small enterprises and home-office environments. *Information Management and Computer Security*, pp.121-128.

Taft, D. K. (2004, Dec). Microsoft aids secure coding. *eWeek*.

Task Force Report. (2004, April). *Improving Security Across the Software Development Life Cycle* (Technical Report.). National Cyber Security Summit.

Tipton, H. F. & Krause, M. (2006). *Information Security Management Handbook* (Fifth ed., Vol. 3). New York : United States of America: Auerbach Publications.

Tompkins, F. G. & Rice, R. (1985). Integrating security activities into the software development life cycle and the quality assurance process. In *Proceedings of IFIP. Computer Security. The practical issues in a troubled world.* (pp.65-105). North Holland.

Tryfonas, T. & Kiountouzis, E. (2002). Information systems security and the information systems development project. In *Proceedings of IFIP. Security in the information society: visions and perspectives.* Kluwer Academic Publishers.

Van Vliet, H. (2000). *Software engineering - principles and practice* (Second ed.). New York : United States of America: Wiley.

Von Solms, R. & Meyer, L. R. (1995). Information security accreditation - the ISO 9000 route. In *Proceedings of IFIP. Information security – the next decade.* (pp.40-49). Chapman and Hall.

Wang, H. & Wang, C. (2003, Jun). Taxonomy of security considerations and software quality : addressing security threats and risks through software quality design factors.

Whitman, M. & Mattord, M. (2003). *Principles of Information Security.* Thomson Course Technology.

Appendix A

The Secure Software Development Model (SecSDM)

INVESTIGATION PHASE

STEP 1 : Information Asset Identification and Valuation (Whitman & Mattord, 2003). For example: customer orders, service requests, employee salaries, EDI documents, etc.

STEP 1a : Information Asset Identification

Which information assets are the most **critical** to the **success** of the proposed software application?	1. 2. 3.
Which information assets pertaining to the proposed software application generate the most **revenue**?	1. 2. 3.
Which information assets pertaining to the proposed software application generate the most **profitability**?	1. 2. 3.
Which information assets pertaining to the proposed software application would be the most **expensive to replace**?	1. 2. 3.
Which information assets pertaining to the proposed software application would be the most **expensive to protect**?	1. 2. 3.
Which information assets pertaining to the proposed software application would be the most **embarrassing** or cause the greatest **liability if revealed**?	1. 2. 3.

STEP 1b : Information Asset Valuation

From the information provided above, select and prioritise the FIVE most important information assets pertaining to the proposed software application and indicate their respective Asset Impact Values (where 0=negligible, 1=low, 2=medium, 3=high, 4=critical) by placing an 'X' in the appropriate cells (one 'X' per asset).

Information Assets	**Asset Impact Value**				
	0 Negligible	**1 Low**	**2 Medium**	**3 High**	**4 Critical**
Asset A ……....					
Asset B ……....					
Asset C ……....					
Asset D ……....					
Asset E ……....					

INVESTIGATION PHASE

STEP 2 : Threat Identification and Assessment (Whitman & Mattord, 2003)

A wide variety of threats face an organisation's information and it's information systems. Each of these threats has the potential to attack any of the information assets previously identified. The following questions need to be addressed:

- *Which threats represent the most danger to the information assets pertaining to the proposed software application?*
- *How much would it cost to recover from a successful attack ?*
- *Which of the threats would require the greatest expenditure to prevent ?*

In order to answer these questions, for each of the common threats listed below, estimate it's level of likelihood, frequency and potential impact to the information assets pertaining to the proposed software application (by placing an 'X' in the appropriate cell).

Common Threats (ISO/IEC TR 13335-3:1998)	**Level of Likelihood/Frequency/Impact**			
	LOW	**MED**	**HIGH**	**N/A**
Theft of information (Deliberate e.g. Illegal information disclosure)				
Use of system by unauthorised users (Deliberate or Accidental e.g. Unauthorised access by competitors)				
Use of system in an unauthorised manner (Deliberate or Accidental e.g. Unauthorised data collection)				
Masquerading of user identity (Deliberate e.g. Malicious Hacking)				
Malicious software attacks (Deliberate or Accidental e.g. viruses, worms, DoS)				
User errors (Deliberate or Accidental e.g. Invalid/inaccurate data entry)				
Repudiation (Deliberate e.g. Denial of having performed transaction)				
Technical software failures or errors (Deliberate or Accidental e.g. Bugs, code problems, unknown loopholes)				
Other …….. ……..				
Other …….. ……..				

INVESTIGATION PHASE					
STEP 3 : Risk (Asset/Threat) Identification (Whitman & Mattord, 2003) *By identifying the most critical asset/threat relationships one is able to ascertain the risks most likely to impact the proposed software application.* *Identify the EIGHT most critical risks (asset/threat relationships) by placing a letter from 'a' to' h' in the appropriate cell.*					
ASSET/THREAT RELATIONSHIP					
Common Threats (ISO/IEC TR 13335-3:1998) (refer to Step 2)	**Information Assets (refer to Step 1)**				
	Asset A	**Asset B**	**Asset C**	**Asset D**	**Asset E**
Theft of information					
Use of system by unauthorised users					
Use of system in an unauthorised manner					
Masquerading of user identity					
Malicious software attacks					
User errors					
Repudiation					
Technical software failures or errors					
Other ……					
Other ……					

INVESTIGATION PHASE			
STEP 4 : Determine Level of Vulnerability *In order to determine the level of vulnerability for each risk (asset/threat relationship 'a' to 'h') as identified in Step 3, you need to consider the likelihood that the risk may materialise, taking the current situation and controls into account. This can be done by plotting the level of vulnerability for each risk (asset/threat relationship 'a' to 'h') by placing an 'X' in the appropriate cell.*			
Risks (refer to Step 3)	**Level of Vulnerability**		
	LOW	**MEDIUM**	**HIGH**
	The asset is quite well protected against the threat, therefore the vulnerability is low.	*The asset is exposed to some degree and is not that well protected, therefore the vulnerability is medium.*	*The asset is exposed to a large degree and is not well protected at all, therefore the vulnerability is high.*
Risk A (Asset/Threat Relationship 'a')			
Risk B (Asset/Threat Relationship 'b')			
Risk C (Asset/Threat Relationship 'c')			
Risk D (Asset/Threat Relationship 'd')			
Risk E (Asset/Threat Relationship 'e')			
Risk F (Asset/Threat Relationship 'f')			
Risk G (Asset/Threat Relationship 'g')			
Risk H (Asset/Threat Relationship 'h')			

INVESTIGATION PHASE

STEP 5 : Risk Assessment (matrix from ISO/IEC TR 13335-3:1998)

In order to determine the actual measure of risk, the relevant Asset Impact Value, level of vulnerability and the likelihood of the threat must be considered for each risk identified in Step 3. The appropriate row in the matrix is identified by the Asset Impact Value of the particular asset (simply carry this over from Step 1). The appropriate column is identified by the likelihood, frequency and potential impact of the particular threat and the level of vulnerability. Simply transfer the level of vulnerability as determined in Step 4, and the likelihood, frequency and potential impact of the particular threat, as determined in Step 2. This must be performed for each risk as identified in Step 3 by placing an 'X' in the appropriate cell.

RISK A (as per Asset/Threat Relationship 'a' in Step 3):

…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

RISK B (as per Asset/Threat Relationship 'b' in Step 3):

…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

RISK C (as per Asset/Threat Relationship ‘c’ in Step 3):

…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

RISK D (as per Asset/Threat Relationship ‘d’ in Step 3):

…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

RISK E (as per Asset/Threat Relationship ‘e’ in Step 3):

…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

RISK F (as per Asset/Threat Relationship ‘f’ in Step 3):
…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

RISK G (as per Asset/Threat Relationship ‘g’ in Step 3):
…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

RISK H (as per Asset/Threat Relationship ‘h’ in Step 3):
…….

……..

		Likelihood/Frequency/Potential Impact of Threat								
		LOW			MEDIUM			HIGH		
		Level of Vulnerability			Level of Vulnerability			Level of Vulnerability		
		LOW	MED	HIGH	LOW	MED	HIGH	LOW	MED	HIGH
Asset Impact Value	Negligible 0	0	1	2	1	2	3	2	3	4
	Low 1	1	2	3	2	3	4	3	4	5
	Medium 2	2	3	4	3	4	5	4	5	6
	High 3	3	4	5	4	5	6	5	6	7
	Critical 4	4	5	6	5	6	7	6	7	8

INVESTIGATION PHASE		
STEP 6 : Risk Prioritisation *For each risk identified in Step 5, transfer the risk description and the risk value to the table below. This step completes the risk analysis part of the Investigation Phase.*		
	Risk Description (Asset/Threat relationship)	**Risk Value**
Risk A		
Risk B		
Risk C		
Risk D		
Risk E		
Risk F		
Risk G		
Risk H		

ANALYSIS PHASE

STEP 7 : Identification of Security Services
Typically the following threats would require the services as indicated in the table below.

Threats	Security Services				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Theft of information	X	X			
Use of system by unauthorised users	X	X			
Use of system in an unauthorised manner		X	X		
Masquerading of user identity	X	X			
Malicious software attacks		X		X	
User errors		X		X	
Repudiation	X				X
Technical software failures or errors				X	

STEP 7 : Identification of Security Services
With this knowledge, map each of the EIGHT risks (asset/threat relationships) identified in the Investigation Phase to the envisaged services. For each risk, more than one service may be identified. Indicate the level of protection required by placing a B (for Basic), S (for Standard) or ES (for Extra Strong) in the relevant cells. Not all services will be required to address each individual risk.

Risks	Security Services				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Risk A					
Risk B					
Risk C					
Risk D					
Risk E					
Risk F					
Risk G					
Risk H					

DESIGN PHASE					
STEP 8: Mapping of Security Services to Security Mechanisms (according to ISO 7498-2) Typically the security services are implemented through the security mechanisms as indicated in the table below.					
Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption	X		X	X	
Digital Signatures	X			X	X
Access Control Mechanisms		X			
Data Integrity Mechanisms				X	X
Authentication Exchange Mechanisms	X				
Traffic Padding			X		
Routing Control			X	X	
Notarisation				X	X

DESIGN PHASE

STEP 8 : Mapping of Security Services to Security Mechanisms (according to ISO 7498-2)
For each risk, identify the specific mechanisms that would be implemented to support the security services required. Indicate the level of protection required by placing a B (for Basic), S (for Standard) or ES (for Extra Strong) in the relevant cells. Not all security services and mechanisms will be required to address each individual risk.

RISK A (as per Asset/Threat Relationship 'a' in Step 3):
…….

……..

Security Mechanisms	Security Services				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

RISK B (as per Asset/Threat Relationship 'b' in Step 3):
…….

……..

Security Mechanisms	Security Services				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

RISK C (as per Asset/Threat Relationship 'c' in Step 3):
…….

……..

Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

RISK D (as per Asset/Threat Relationship 'd' in Step 3):
…….

……..

Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

RISK E (as per Asset/Threat Relationship 'e' in Step 3):
…….

……..

Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

RISK F (as per Asset/Threat Relationship 'f' in Step 3):
…….

……..

Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

RISK G (as per Asset/Threat Relationship ‘g’ in Step 3):
…….

……..

Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

RISK H (as per Asset/Threat Relationship ‘h’ in Step 3):
…….

……..

Security Mechanisms	**Security Services**				
	Identification & Authentication	**Access Control**	**Data Confidentiality**	**Data Integrity**	**Non-repudiation**
Encipherment/ Encryption					
Digital Signatures					
Access Control Mechanisms					
Data Integrity Mechanisms					
Authentication Exchange					
Traffic Padding					
Routing Control					
Notarisation					

DESIGN PHASE

STEP 9 : Summary of Findings

In the table below indicate which security services and mechanisms would be required to address the specified risks (A, B, C, D, E, F, G, H). Simply place an 'X' in the appropriate cells.

Security Mechanisms	**Security Services**																			
	Identification & Authentication				**Access Control**				**Data Confidentiality**				**Data Integrity**				**Non-repudiation**			
Encipherment/ Encryption	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Digital Signatures	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Access Control Mechanisms	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Data Integrity Mechanisms	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Authentication Exchange	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Traffic Padding	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Routing Control	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H
Notarisation	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H	E	F	G	H

IMPLEMENTATION PHASE

STEP 10 : Mapping of Security Mechanisms to .Net and Other Components

For each security mechanism, indicate the risks (A, B, C, D, E, F, G, H) for which it is relevant, by placing an 'X' next to the appropriate risk (transferred directly from Step 9). In addition, identify the specific .Net or other components through which the security mechanisms will be implemented.

			.Net security components	**Other components**
Security Mechanisms	Encipherment/ Encryption	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		
		Risk H		
	Digital Signatures	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		
		Risk H		
	Access Control Mechanisms	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		
		Risk H		
	Data Integrity Mechanisms	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		
		Risk H		
	Authentication Exchange Mechanisms	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		
		Risk H		
	Traffic Padding	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		
		Risk H		
	Routing Control	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		
		Risk H		
	Notarisation	Risk A		
		Risk B		
		Risk C		
		Risk D		
		Risk E		
		Risk F		
		Risk G		

Appendix B

Information Security Questionnaire

INFORMATION SECURITY QUESTIONNAIRE

Purpose : The aim of this questionnaire is to establish the extent to which information technology project students at the NMMU consider and incorporate information security into their 3rd year development projects.

Use of Results : The results of this questionnaire will provide essential information required to support the need for a new methodology for integrating information security into software development projects.

Instructions : Answer(as honestly as possible) all questions in each of the sections below as pertaining to your 3rd year development project. Simply circle YES or NO or NOT SURE in the appropriate column. In some sections you are required to circle option a, b or c. Please provide additional detail where possible.

STUDENT INFORMATION

Student Name : …… Student Number :…….

Signature :…….. Date completed : ……/……/20…..

Stream (TA = technical applications, SD = software development)	TA	SD

Brief Project Description ……..…
……..
……..
……..
……..
……..
……..

SECTION 1 (Investigation Phase)

1.1 Did you identify and assess the importance of all the **information assets** associated with the application you developed?	YES	NO	NOT SURE
1.2 Did you identify any of the **threats** that may compromise your application, specifically the information assets ?	YES	NO	NOT SURE
1.3 Did you identify any **vulnerabilities** within the deployment environment that may negatively impact on the efficient running of your application?	YES	NO	NOT SURE
1.4 Did you perform any other form of **risk analysis**, either formal or informal? If yes, provide brief details ……..... …… …… …… …… …… ……	YES	NO	NOT SURE

SECTION 2 (Analysis Phase)			
2.1 Did you include any **identification/authentication** service (e.g. passwords) in the application?	YES	NO	NOT SURE
If yes, did you include this service as a result of a specific risk anticipated?	YES	NO	NOT SURE
➢ If yes, was this based on any formal reason (provide details) …… …… ➢ If no, why did you include it? …… ……			
2.2 Did you include any **authorisation/access control** service (e.g. access rights depending on role of users) in the application?	YES	NO	NOT SURE
If yes, did you include this service as a result of a specific risk anticipated?	YES	NO	NOT SURE
➢ If yes, was this based on any formal reason (provide details) …… …… ➢ If no, why did you include it? …… ……			
2.3 Did you include any **confidentiality** service (e.g. encryption) in the application?	YES	NO	NOT SURE
If yes, did you include this service as a result of a specific risk anticipated?	YES	NO	NOT SURE
➢ If yes, was this based on any formal reason (provide details) …… …… ➢ If no, why did you include it? …… ……			
2.4 Did you include any **integrity** service (e.g. hashing) in the application?	YES	NO	NOT SURE
If yes, did you include this service as a result of a specific risk anticipated?	YES	NO	NOT SURE
➢ If yes, was this based on any formal reason (provide details) …… …… ➢ If no, why did you include it? …… ……			
2.5 Did you include any **non-repudiation** service (e.g. digital signatures) in the application?	YES	NO	NOT SURE
If yes, did you include this service as a result of a specific risk anticipated?	YES	NO	NOT SURE
➢ If yes, was this based on any formal reason (provide details) …… …… ➢ If no, why did you include it? ……			

……			
SECTION 3 (Design and Implementation Phases)			
3.1 Did you include any form of **encryption/encipherment**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough to protect the associated information assets?	YES	NO	NOT SURE
3.2 Did you include any form of **digital signatures**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough to protect the associated information assets?	YES	NO	NOT SURE
3.3 Did you include any form of **access control**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough to protect the associated information assets?	YES	NO	NOT SURE
3.4 Did you include any form of **data integrity**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough to protect the associated information assets?	YES	NO	NOT SURE
3.5 Did you include any form of **authentication**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough to protect the associated information assets?	YES	NO	NOT SURE
3.6 Did you include any form of **traffic padding**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough to protect the associated information assets?	YES	NO	NOT SURE

3.7 Did you include any form of **routing control**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough to protect the associated information assets?	YES	NO	NOT SURE
3.8 Did you include any form of **notarisation**?	YES	NO	NOT SURE
If yes, did you			
a) use a .NET component,	a		
b) use some other component, or	b		
c) write your own?	c		
If yes, are you confident that the component is "strong" enough protect the associated information assets?	YES	NO	NOT SURE

SECTION 4 (Maintenance Phase)			
4.1 Did you define any **user procedures** for using the application in a **<u>secure</u>** way?	YES	NO	NOT SURE
If yes, did you specify any **user education processes** in this regard?	YES	NO	NOT SURE
If yes, provide brief details : ……			
……			
……			
……			
4.2 Did you **document** any information security related detail during the analysis, design or implementation phases?	YES	NO	NOT SURE
If yes, provide brief details : ……			
……			
……			
……			
4.3 Would it be possible to **audit** the information security aspects of the application you developed, to determine whether too few , too many or adequate security services and components were considered and integrated into the application?	YES	NO	NOT SURE

SECTION 5 (General Opinions)			
5.1 In your opinion, would you say that :			
a) too few	a		
b) too many	b		
c) adequate	c		
security services and components were considered and integrated into your application?			
5.2 Do you believe that there is a need for a new, or improved, methodology for integrating information security into software development projects?	YES	NO	NOT SURE

Thank you for completing this questionnaire!

Nelson Mandela Metropolitan University

for tomorrow

Appendix C

Results of Information Security Questionnaire

Note to reader:

When examining the results of the information security questionnaire as reported in this Appendix, it is necessary to refer back to Appendix B for the applicable questions. Each question number used in this Appendix refers to the particular question as provided in Appendix B.

SECTION 1 (Investigation Phase)				
Question as per Appendix B		2005 (%)	2006 (%)	DIFF (%)
1.1	**Information Assets**			
	Yes	39	95	56
	No	23	5	-18
	Not Sure	38	0	-39
1.2	**Threats**			
	Yes	45	80	35
	No	35	15	-20
	Not Sure	20	5	-14
1.3	**Vulnerabilities**			
	Yes	42	70	28
	No	32	25	-7
	Not Sure	26	5	-21
1.4	**Risk Analysis**			
	Yes	6	20	14
	No	77	50	-27
	Not Sure	17	30	14

SECTION 2 (Analysis Phase)				
Question as per Appendix B		2005 (%)	2006 (%)	DIFF (%)
2.1	**Identification and Authentication**			
	Yes	81	100	19
	Yes	72	85	13
	No	12	10	-2
	Not Sure	16	5	-11
	No	19	0	-19
	Not Sure	0	0	0
2.2	**Authorisation and Access Control**			
	Yes	58	75	17
	Yes	67	87	20
	No	11	13	2
	Not Sure	22	0	-22
	No	39	25	-14
	Not Sure	3	0	-3
2.3	**Confidentiality**			
	Yes	19	50	31
	Yes	50	90	40
	No	0	10	10
	Not Sure	50	0	-50
	No	74	50	-24
	Not Sure	7	0	-7
2.4	**Integrity**			
	Yes	13	10	-3
	Yes	75	100	25
	No	0	0	0
	Not Sure	25	0	-25
	No	81	90	9
	Not Sure	6	0	-6
2.5	**Non-repudiation**			
	Yes	0	0	0
	Yes	0	0	0
	No	0	0	0
	Not Sure	0	0	0
	No	90	85	-5
	Not Sure	10	15	5

SECTION 3 (Design and Implementation Phases)				
Question as per Appendix B		2005 (%)	2006 (%)	DIFF (%)
3.1	**Encryption and encipherment**			
	Yes	16	50	34
	a (.Net)	80	70	-10
	b (other)	20	30	10
	c (own)	0	0	0
	Yes	0	60	60
	No	0	0	0
	Not Sure	100	40	-60
	No	84	45	-39
	Not sure	0	5	5
3.2	**Digital signatures**			
	Yes	0	0	0
	a (.Net)	0	0	0
	b (other)	0	0	0
	c (own)	0	0	0
	Yes	0	0	0
	No	0	0	0
	Not Sure	0	0	0
	No	97	95	-2
	Not sure	3	5	2
3.3	**Access control**			
	Yes	71	70	-1
	a (.Net)	41	50	9
	b (other)	9	0	-9
	c (own)	50	50	0
	Yes	18	57	39
	No	18	14	-4
	Not Sure	64	29	-35
	No	23	25	2
	Not sure	6	5	-1
3.4	**Data integrity**			
	Yes	35	60	25
	a (.Net)	64	50	-14
	b (other)	9	0	-9
	c (own)	27	50	23
	Yes	45	58	13
	No	9	8	-1
	Not Sure	46	33	-13
	No	42	40	-2
	Not sure	23	0	-23

SECTION 3 (Design and Implementation Phases) *(continued)*				
Question as per Appendix B		2005 (%)	2006 (%)	DIFF (%)
3.5	**Authentication**			
	Yes	71	75	4
	a (.Net)	23	27	4
	b (other)	4	26	22
	c (own)	73	47	-26
	Yes	14	40	26
	No	18	13	-5
	Not Sure	68	47	-21
	No	23	20	-3
	Not sure	6	5	-1
3.6	**Traffic padding**			
	Yes	3	20	17
	a (.Net)	0	0	0
	b (other)	0	0	0
	c (own)	100	100	0
	Yes	0	0	0
	No	0	50	50
	Not Sure	100	50	-50
	No	58	60	2
	Not sure	39	20	-19
3.7	**Routing control**			
	Yes	0	40	40
	a (.Net)	0	0	0
	b (other)	0	50	50
	c (own)	0	50	50
	Yes	0	75	75
	No	0	0	0
	Not Sure	0	25	25
	No	81	45	-36
	Not sure	19	15	-4
3.8	**Notarisation**			
	Yes	0	0	0
	a (.Net)	0	0	0
	b (other)	0	0	0
	c (own)	0	0	0
	Yes	0	0	0
	No	0	0	0
	Not Sure	0	0	0
	No	68	45	-23
	Not sure	32	55	23

SECTION 4 (Maintenance Phase)				
Question as per Appendix B		2005 (%)	2006 (%)	DIFF (%)
4.1	**User procedures**			
	Yes	6	10	4
	Yes	50	100	50
	No	50	0	-50
	Not Sure	0	0	0
	No	77	70	-7
	Not Sure	17	20	3
4.2	**Documentation**			
	Yes	10	25	15
	No	81	70	-11
	Not Sure	9	5	-4
4.3	**Auditing**			
	Yes	23	55	32
	No	35	35	0
	Not Sure	42	10	-32

SECTION 5 (General Opinions)				
Question as per Appendix B		2005 (%)	2006 (%)	DIFF (%)
5.1	**Security of system**			
	a (too few)	68	25	-43
	b (too many)	4	0	-4
	c (adequate)	28	75	47
5.2	**Need for a secure software development methodology?**			
	Yes	68	69	1
	No	0	6	6
	Not Sure	32	25	-7

FINAL THOUGHT

What lies behind us and what lies before us are tiny matters compared to what lies within us!

-William Morrow-