# SecMVC

# A MODEL FOR SECURE SOFTWARE DESIGN BASED ON THE MODEL-VIEW-CONTROLLER PATTERN

by

**Michael Colesky**

2014

# SecMVC: A Model for Secure Software Design based on the Model-View-Controller Pattern

by

**Michael Robert Colesky**

**Research Thesis**

submitted 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: Dr Lynn Futcher**

**Co-supervisors: Dr Mariana Gerber & Prof. Johan van Niekerk**

2014

# Declaration

I, Michael Robert Colesky, hereby declare that the dissertation for the degree of Magister Technologiae in Information Technology is my own work, and that it has not previously been submitted for assessment or completion of any postgraduate qualification to another university or for another qualification.

[illegible]

Michael Robert Colesky

# Abstract

Current advances in the software development industry are growing more ubiquitous by the day. This has caused for security, not only in the broader sense, but specifically within the design and overall development of software itself, to become all the more important.

An evidently prevalent problem in the domain of software development is that ***software security is not consistently addressed during design, which undermines core security concerns, and leads to the development of insecure software***. This research seeks to address this issue via a model for secure software design, which is based on a software design pattern, namely, the Model-View-Controller (MVC) pattern.

The use of a pattern to convey knowledge is not a new notion. However, the ability of software design patterns to convey secure software design is an idea worth investigating. Following identification of secure software design principles and concepts, as well as software design patterns, specifically those relating to the MVC pattern, a model was designed and developed.

With the MVC pattern argued as being a suitable foundation for the model, the security-conscious MVC (SecMVC) combines secure software design principles and concepts into the MVC pattern. Together herewith, the MVC pattern's components in the MVC Compound pattern, namely: the Observer pattern, the Strategy pattern, and the Composite pattern, have provided further sub-models for less abstraction and greater detail.

These sub-models were developed, as a result of the SecMVC model's evaluation in the validation for this study, an expert review. Argued in the light of similar research methods, the expert review was chosen – along with a process that included the use of two expert participants to validate the SecMVC model. It was determined through the expert review that the SecMVC model is of sufficient utility, quality, and efficacy to constitute research value.

The research methodology process followed was design science, in which the SecMVC model, which includes its related sub-models, serves as the artefact and research output of this study.

This research study contributes evidence of the feasibility for integrating knowledge into software design patterns. This includes the SecMVC model itself. In addition, it argues for the use of an expert review, as an evaluative research method for such an artefact.

# Acknowledgements

*I would like to express the deepest gratitude to the following persons:*

- *My supervisors, Dr Lynn Futcher, Dr Mariana Gerber, and Prof. Johan van Niekerk, for their continued support, guidance, suggestions, and time invested;*
- *Dr Lynn Futcher, for her rigorous analysis and helpful feedback week after week;*
- *To my family, who supported me during my studies, and pressed me to do my best;*
- *To my colleagues in the School of Information Communication and Technology;*
- *The expert reviewers who contributed valuable insights and suggestions for the SecMVC model and its sub-model;*
- *The National Research Foundation (NRF) for providing funding which allowed me to write this dissertation as a full-time student;*
- *Dr Mariana Gerber for providing assistance and allocation with the NRF funding;*
- *The Nelson Mandela Metropolitan University for providing internal funding; and*
- *My editor, Dr Patrick Goldstone, for his time and contribution to the soundness of all the text in this document, and for his interest in assisting me in the proofreading of this dissertation. His Language-editing Certificate is shown in Appendix D.*

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| AEGIS | Appropriate and Effective Guidance for Information Security |
| AJAX | Asynchronous JavaScript and XML |
| API | Application Programming Interface |
| CAS | Code Access Security |
| CIA | Confidentiality, Integrity and Availability |
| CLASP | Comprehensive Lightweight Application Security Process |
| CLR | Command Line Runtime |
| COM | Component Object Model |
| CSSLP | Certified Secure Software Lifecycle Professional |
| CWE | Common Weakness Enumeration |
| DNS | Domain Name Service |
| DPSL | Design Pattern Specification Language |
| HTML | Hypertext Markup Language |
| ICT | Information and Communication Technology |
| IDE | Integrated Development Environment |
| IDS | Intrusion Detections System |
| IEC | International Electrotechnical Commission |
| IP | Internet Protocol |
| IPSec | Internet Protocol Security |
| IS | Information System |
| (ISC)$^2$ | International Information Systems Security Certification Consortium |
| ISO | International Standards Organisation |
| JSON | JavaScript Object Notation |

| | |
|---|---|
| LDAP | Lightweight Directory Access Protocol |
| MBASE | Model-Based Architecture and Software Engineering |
| MVC | Model-View-Controller |
| MVP | Model-View-Presenter |
| MVVM | Model-View-ViewModel |
| NMMU | Nelson Mandela Metropolitan University |
| NRF | National Research Foundation |
| NTDLL | Native Windows Library |
| OS | Operating System |
| OSI | Open Systems Interconnection |
| OSSA | Oracle Software Security Assurance |
| OWASP | Open Web Application Security Project |
| RMIAS | Reference Model of Information Assurance & Security |
| RPC | Remote Procedure Call |
| RUP | Rational Unified Process |
| $S^2e$ | Secure Software Engineering |
| SDL | Secure Development Lifecycle |
| SDLC | Software Development Life Cycle |
| SDU | Service Data Unit |
| SecMVC | Security-conscious Model-View-Controller |
| SQL | Structured Query Language |
| SSDM | Secure Software Development Model |
| SSL | Secure Sockets Layer |
| TCP | Transmission Control Protocol |
| TSPsecure | Team Software Process |
| UI | User Interface |
| UML | Unified Modelling Language |
| URI | Uniform Resource Identifier |
| WPF | Windows Presentation Foundation |
| XSS | Cross-Site Scripting |
| ZAWWW | South African World Wide Web |

# Chapter 1: Introduction

## 1.1. Introduction

The purpose of this research project is to investigate and explore the prospect of improved software design security, and therefore software development security through the use of security-conveying software design patterns. Where software design patterns typically demonstrate favourable software design practices, this research aims to show that such software design patterns could also promote good security habits; that is an aspect, which is ignored – to a far greater extent – than good coding practice (Hight, 2005, p. 1).

Research may be defined simply as a 'systematic investigation undertaken in order to discover new facts, [and] get additional information' (Hornby, Cowie, Gimson, & Lewis, 1986, p. 700). Similarly, the Collins English Dictionary defines research as a 'systematic investigation to establish facts or principles or to collect information on a subject' (Collins & Anderson, 2006). Definitions may, on the other hand, vary considerably; however, the gains in knowledge for both academia and industry can constitute research (Hussey & Hussey, 1997, p. 1; Tull & Hawkins, 1987, p. 26).

This research project has the potential to do the same, as secure design contributes to the development of favourable business solutions; and consequently, it has academic relevance.

Used on a regular basis as common solution-foundation guidelines, software design patterns provide a vessel for the relevant representation of good coding practice. This alone should benefit both students and organisations. As such, the proposed research is in line with the need to present a topic that is suitable and worth investigating. The topic, within the domains of secure software design and software design patterns, is where the problem for this research is identified. Stated clearly and concisely in Section 1.3, it is from this problem that the researcher's work is derived.

The structure, as described in this introduction, can be seen in Figure 1.1.

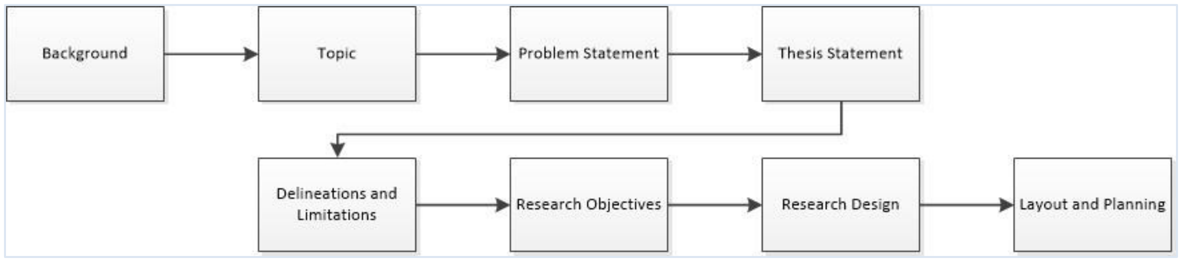


Figure 1.1: Flow of This Chapter

It is important to note that the proposition identified by this research, is of limited scope and specific focus. In order to delineate this research into manageable sections, it must be separated into a primary research objective, as well as secondary objectives in support of the primary objective. These are obtained through the problem; and they contribute to the motivation of this research, and as such, provide a candidate contribution to the solution.

The research objectives in this project are followed by the research design section. This discusses the specific aspects addressed by the research process, such as the paradigm and the methodology. Beyond this, there is the layout of the chapters and sections, insight into what has been achieved, according to this research, and finally the conclusion.

## 1.2. Background Information

The main objective of this research is to develop one or more models, in which software design patterns, specifically the Model-View-Controller (MVC) pattern, are a medium for introducing secure software design principles and concepts to the software design and following development phases. This research study considers software design a component of software development; therefore this area is also reflected. In order to meet the primary research objective, it is necessary to study two specific focus areas, namely: secure software development and software design patterns. These are further described in Sections 1.2.2 and 1.2.3, respectively.

### 1.2.1. Focus of this Research

The fields of study with which this research is concerned are to be found in the broader fields of software security and software development. As seen in Figure 1.2, these broader areas should first be studied – before addressing the more specific topic. The chosen focus of this research is based on the backgrounds of the researcher and the supervisors, and the strengths and interests thereof.

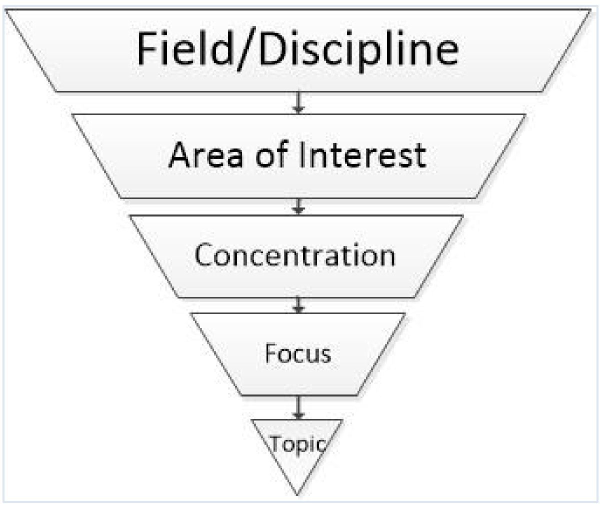


Figure 1.2: Topic Focus (adapted from Hofstee, 2006, p. 14)

Information security is ‘based on information being considered as an asset’ (ISO/IEC 27000, 2009, p. 6). Software security, on the other hand, is more concerned with the ‘protection of software and its resources’ (Sodiya, Onashoga, & Ajayi, 2006, p. 637). While both of these seek to protect confidentiality, integrity and availability, there is a clear distinction between them. This research is aligned within software security.

It may well be said that software security is a form of risk management (Viega & McGraw, 2006, p. 24). However, the focus area to be examined from within software security is secure software development. Therefore, in order to maintain its focus, this project does not include risk management in its scope.

Furthermore, in favour of minimising scope, this research also focuses its analysis on application security within the .NET framework, specifically. This has been chosen as the target framework for the following reasons:

Firstly, the School of Information and Communication Technology (ICT) at the Nelson Mandela Metropolitan University (NMMU) has a strong focus on .NET framework technologies. Secondly, because of this, the .NET framework is the area, in which the researcher is most knowledgeable; and therefore, best suited. Lastly, this framework is arguably one of the most current and well-known technologies in the field of software development. Secure software development is further discussed in Section 1.2.2.

Software development can be regarded as ‘proceeding from an analysis phase to a design phase and then to coding and testing’ (Haworth, 2002, p. 6). It is the design and implementation of a solution to an identified business problem. This research study considers security from the perspective of software development in general, with a specific focus on the design phase.

Even software development alone can be quite expansive; consequently the focus of this research, in addition to secure software design, is on software design patterns; generalised approaches addressing software requirements, which have stood the test of time, and been proven to be useful guidelines (Ampatzoglou, Frantzeskou, & Stamelos, 2012, p. 1). These are examined in Section 1.2.3. The following section, however, discusses secure software development; the broader containing area of secure software design.

### 1.2.2. Secure Software Development

Khan and Mustafa (2008, p. 1623) describe secure software development as, ‘the term largely associated with the process of producing reliable, stable, bug and vulnerability free software’. It is commonly associated with providing security consistently, at all the stages of development. Notably, secure software design focuses on the design phase. A Software Development Lifecycle, as described by Paul (2008a), is depicted in Figure 1.3.

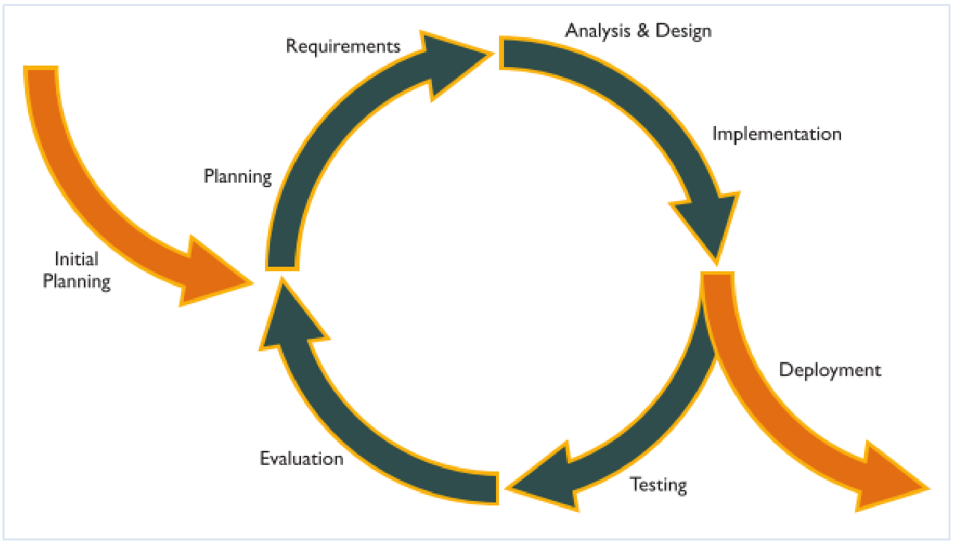


Figure 1.3: Software Development Life Cycle (Paul, 2008a)

All of these stages are important; and there are numerous frameworks, which seek to address them (Khan & Mustafa, 2008, pp. 1623–1624). However, typically these frameworks approach software development and security as mutually exclusive processes. An extensive, though non-exhaustive list of frameworks for secure software development would include:

- The Microsoft Secure Development Lifecycle (SDL);
- The Oracle Software Security Assurance (OSSA) framework;
- The Comprehensive Lightweight Application Security Process (CLASP);
- McGraw's Seven Touch Points;
- Team Software Process (TSPsecure);
- The Secure Software Development Model (SSDM);
- Appropriate and Effective Guidance for Information Security (AEGIS);
- The Rational Unified Process (RUP);
- The Model-Based Architecture and Software Engineering (MBASE) security extension; and
- Secure Software Engineering ($S^2e$).

Experts suggest that secure software development should be approached from the perspective of every stage of the lifecycle (Breu, Burger, Hafner, & Popp, 2004, p. 1; Jones & Rastogi, 2004, p. 1). This is necessary to ensure that security is not merely attached to an already otherwise complete solution; but rather, it is incorporated from the bottom up.

The issue which arises in this area is the lack of consideration for security at the earliest stages of development. Frequently, it is only seen to be a requirement during or after implementation; and security is frequently considered to be nothing more than a diversion from productivity (Jürjens, 2002, p. 2). Programmers, instead, rely on coding habits alone to ensure that an application is robust; however, often this is not enough (Khan & Mustafa, 2008, p. 1622). When security is deliberately ignored, in order to cut costs, organisations risk far greater costs in the long run (Mouratidis, Giorgini, & Manson, 2005, p. 610).

Software is made without security in mind – for two predominant reasons (Paul, 2008a, p. 1). One is the lack of appreciation for the benefits of a secure solution, specifically with regard to the software development process. The other is the lack of experience and skill of many developers tasked with secure software development.

An understanding of software security's significance is best accomplished through security education, training, and the raising of awareness (Paul, 2008b). Treating the lack in secure design and coding skills requires much of the same. Training courses can be taken; and the initial education could be modified; although this is not always feasible. Alternative approaches, therefore, need to be considered to supplement this. One such medium is the integration of security considerations into software design patterns. This research study considers this option within the more specific secure software design context.

### 1.2.3. Software Design Patterns

Ampatzoglou et al. (2012, p. 1) describe software design patterns as 'documented solutions to common design problems that are expected to enhance software quality'. They are guidelines frequently utilised for generally effective solutions, featuring not only the approach to a software design problem, but also the specific context in which it occurs (Yoshioka, Washizaki, & Maruyama, 2008, pp. 35–36).

These patterns are considered by both industry professionals and students. When taught to students, they provide the necessary skills and experience needed to develop robust, extensible, and reliable solutions (Freeman, Robson, Bates, & Sierra, 2004, pp. 85–87).

Software design patterns are generally depicted visually, as for example in Unified Modelling Language (UML) 2.0, and applied helpful software design principles (Freeman et al., 2004, p. 9). When shown via visual representation, class diagrams are often used. These are typically available for illustrating the structure of an object-oriented design. These are applicable – irrespective of whether the design pattern is a class pattern or an object pattern (Freeman et al., 2004, p. 591).

These patterns are not, however, limited to software design and implementation only. Possible variations include: architecture, application, domain-specific, business process, organisational, and user interface design patterns (Freeman et al., 2004, pp. 604–605).

Security patterns are one example of pattern usage outside the application domain, while remaining applicable to a solution's design. These security patterns exist for each stage of the software development lifecycle (Yoshioka et al., 2008, pp. 36–37). Problems have been noted, however, with regard to the analysis stage security patterns.

Analysis stage security patterns are described as being particularly abstract; and they pose a barrier to less experienced developers (Yoshioka et al., 2008, p. 43). On the other hand, in the design phase, common use of UML to depict design patterns promotes sufficient understanding. Making use of simple, or frequently used, software design patterns to convey secure software design principles and concepts specifically, is therefore, a feasible supplement to these separate approaches.

Yoshioka et al. (2008, p. 43) note that specifically for implementation patterns, the problem and solution should both be explicit. Despite this, the surveyed security patterns identified possessed a strong focus on attack patterns in the analysis phase, and a lack of security specification in the design and implementation phase (Yoshioka et al., 2008, pp. 43–44). As such, these gaps must be filled.

As mentioned, security patterns are better utilised in specific implementations. However, because security patterns are not appropriate; and since they are less effective for abstracted concepts, these gaps may be filled by generalised design patterns with secure software design principles and concepts integrated. The following section addresses the combined focus areas in the research topic.

### 1.2.4. Research Topic

As mentioned above, this research focuses primarily on the development of one or more models for, and subsequent demonstration of, the ability of software design patterns to integrate secure software design principles and concepts. The areas discussed above contribute accordingly.

Secure software development, specifically focusing on secure software design and the .NET framework, provides the principles and concepts, which are to be introduced. On the other hand, software design patterns are the medium whereby these are presented to developers. This leads to the topic being defined formally as:

***Integrating secure software design principles and concepts into a model using software design patterns.***

With the research topic stated concisely, Section 1.3 discusses the problem, and then presents the problem statement in a similar fashion.

## 1.3. Problem Statement

Software security protects information, resources, and other assets. Assets are of value, and as such protecting them is important. The need for specific consideration for security in a software development project is an important issue which should be conveyed. Traditionally, its importance is addressed through security education, training and awareness, although this is not always deemed to be worth the expense. Without this, security is often only addressed during or after implementation, instead of during the design phase.

The problem statement for this research is as follows:

***Software security is not consistently addressed during design, which undermines core security concerns, and leads to the development of insecure software.***

Given due consideration, failing to provide developers with the skills and experience needed, makes any attempt to provide for security ineffectual. However, developers already in the software development industry might not be willing to sacrifice the time, the costs, and the additional pressures of additional education and training. This further contributes to insufficient provisioning for security at the design phase, even when sufficient analysis has taken place.

## 1.4. Delineation and Limitations

The scope of any project must be controlled, to ensure that it is realistic and can demonstrate feasible progress over a limited period of time. This research should maintain its boundaries, in order to prevent addressing too wide a scope.

The specific focus of this research lies within the development of the models themselves, ***which integrates secure software design principles and concepts in a .NET framework context into a model using the MVC pattern as a basis.*** No attempt is made to construct a new software design or security pattern. While components of the MVC pattern may be addressed separately, a single comprehensive model is considered to be more concise. Only the Model-View-Controller, Observer, Composite and Strategy patterns are considered.

Furthermore, in order to maintain a realistic scope, this research focuses only on design-specific aspects of the software development lifecycle, keeping in mind the significance of the need for security in every stage. The aim of this study is ultimately to improve coding security for industry, in particular. An artefact, in the form of one or more models representing a software design pattern and secure software design principle and concept integration, is produced by this research.

As mentioned in the previous sections, this research focuses specifically on application security in the .NET framework context.

## 1.5. Research Objectives

Following from the problem and delineation for this research, the primary research objective is:

***To develop a model, based on the MVC pattern, which addresses software security through secure software design.***

The sub-objectives necessary to meet the primary research objective are:

- To identify secure software design principles and concepts from the secure software development literature;
- To analyse software design patterns from the literature and the MVC pattern's suitability for this research study;
- To integrate the identified secure software design principles and concepts into the MVC pattern;
- To validate the proposed model, verifying its utility, quality and efficacy.

The focus of this research project is on integrating general security principles and concepts pertaining to secure software design into the MVC pattern, and representing these issues in the security-conscious MVC (SecMVC) model.

## 1.6. Research Design

This research project aims to create an artefact, to verify it through publication, and then to validate it by way of an expert review. As such, this research makes use of deductive reasoning, as described by Saunders, Lewis and Thornhill (2003, p. 107). In relation to the elements described by Saunders et al. (2003, p. 81), this project also endeavours to utilise multiple methods. These are to be: the literature review, modelling, and the expert review.

While some might stress the importance of these methods falling under the same paradigm, Mingers (2001, p. 5) argues that paradigms are 'purely an heuristic device'; they should not restrict the ability to explore methodology combinations. As suggested by Mingers (2001, p. 5), limiting methodology in favour of any one paradigm is to be avoided.

### 1.6.1. Research Paradigm

This research project uses design science as its research paradigm. Peffers, Tuunanen, Rothenberger and Chatterjee (2007, p. 49) describe design science research as a rigorous process to create, evaluate, communicate and contribute artefacts. They list what qualifies as an artefact, as constructs, as models, as methods, as instantiations, as social innovations, and as technical, social, or informational resource properties. In addition to this, they specify that the artefact should be intended for organisational problems.

These are referred to as ‘practice rules’ by Peffers et al. (2007, p. 49). Hevner, March, Park, and Ram (2004, p. 83), however, proposed seven guidelines for design science research. These are listed below:

1. Design as an artefact;
2. Problem relevance;
3. Design evaluation;
4. Research contributions;
5. Research rigour;
6. Design as a research process; and
7. Communication of research.

These guidelines are in line with the requirements listed by Peffers et al. (2007, p. 49) to qualify as design science. Their descriptions also tie in with the need for specifics in artefact classification and business problem-solving.

This research follows a formalised approach, in which it thoroughly studies the resources, creates an artefact, evaluates it, and publishes it in the form of this dissertation, as well as in a paper published in the 2013 South African World Wide Web (ZAWWW) conference proceedings (See Appendix A). The artefact qualified since it addresses the business need for greater security in software design and implementation.

Peffers et al. (2007, p. 54) also present a strict process to adhere to, when following the design science research paradigm. These are: problem identification and motivation; the objectives of a solution; design and development; demonstration; evaluation; and communication. This process is shown in Figure 1.4.

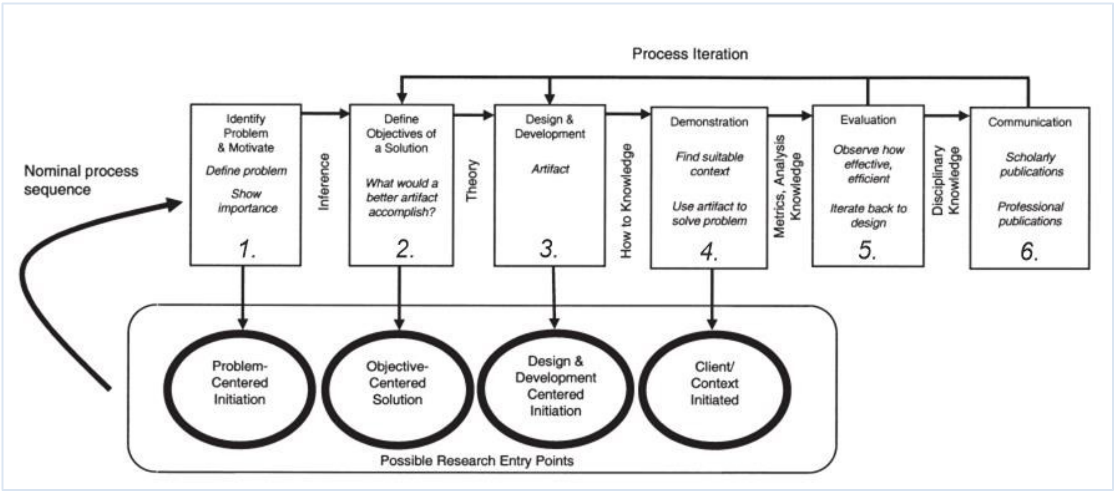

Process Iteration
Nominal process sequence
Identify Problem & Motivate
Define problem
Show importance
1.
Inference
Define Objectives of a Solution
What would a better artifact accomplish?
2.
Theory
Design & Development
Artifact
3.
How to Knowledge
Demonstration
Find suitable context
Use artifact to solve problem
4.
Metrics, Analysis Knowledge
Evaluation
Observe how effective, efficient
Iterate back to design
5.
Disciplinary Knowledge
Communication
Scholarly publications
Professional publications
6.
Problem-Centered Initiation
Objective-Centered Solution
Design & Development Centered Initiation
Client/ Context Initiated
Possible Research Entry Points

Figure 1.4: Design Science Process Model (adapted from Peffers et al., 2007, p. 54)

In order to conform to the Design Science Research Methodology Process, as suggested by Peffers et al. (2007, p. 54), this research adheres to the required stages indicated in Figure 1.4 in the following manner:

1. To **identify and motivate the problem.** A generalised literature review is conducted. A further literature review on secure software development ascertains that the identified problem exists, is important, is relevant, and is in some way treatable.
2. To **define the objectives**, the potential handling of the problem is formulated through a further literature review on secure software development and software design patterns.
3. Tested through the writing of a conference paper on a web-specific take on the topic, steps towards the **modelling of the artefact** are made. A further literature review and models contribute additionally to the artefact.
4. The **designed artefact is demonstrated** through examples. Three examples showing specific implementation of the generalised principles and concepts of the SecMVC model are validated along with the model in an expert review.
5. The **artefact is evaluated** through the expert review. This step determines how easily the artefact is utilised, the quality of the artefact, and how effectively the artefact addresses security concerns.
6. The **artefact is presented** as both the output of the ZAWWW publication (Appendix A) and of this research study, with this research presenting further enhancements. The dissertation and any papers published are to be communicated to both academia and industry.

### 1.6.2. Research Methodology

The research methodology utilised in this research project is as follows: an in-depth literature review, modelling, and an expert review. These are in accordance with the required steps in the process defined earlier regarding design science. The way these are used is based on the methods described by Martin S. Olivier (2009).

***Literature Review***

A wide variety of literature on the identified focus areas, namely, secure software development and software design patterns, is surveyed and utilised in subsequent assertions. This is in addition to the broader areas of software security and software development. The literature review serves to form the basis and foundation for other methods. This technique was first utilised prior to the final formulation of the problem; and it continues to be used throughout the duration of the research project.

***Modelling***

The artefact output by this research project is in the form of one or more models. These models integrate secure software design principles and concepts into the design process through the software design pattern depictions. Notably, the models forthcoming from this research are not design patterns in their own right; but, they are rather extensions of the original. In this case, it could be said that a pattern is far more specific than a model. In the case of a design pattern this means a fully re-usable guideline to the solution of a design problem.

These depictions capture and summarise the security aspects in a form that generalises their usage as software design patterns do. However, no attempt is made to qualify the output as a software design pattern. While still similar to, although more graphical than good coding practice, the secure software design principles and concepts should be easily perceived and understood by developers who are aware of the pattern delivering them.

The final output features the SecMVC model, which comprises the elements depicted in isolation, as well as with constituent software design patterns: Observer, Strategy, and Composite patterns.

***Expert Review***

Expert review is less known; although through its variations, a well-argued research method for evaluation purposes can be conceived. It is used frequently within the School of Information and Communication Technology (ICT) at the Nelson Mandela Metropolitan University (NMMU). Similar methods include: expert surveys, Delphi studies, expert interviews, elite interviews, expert panels and focus groups.

These differ in the following manner, as is considered by this research study:

- **Focus groups**: Using a random sample of individuals to discuss the subject matter in person, within the same room as one other (Bhattacherjee, 2012, p. 78);
- **Expert panels**: Using a preselected sample of relevant experts to discuss the subject matter in person, within the same room as one another (Slocum, Steyaert, & Lisoir, 2005, p. 121);
- **Elite interviews**: Using a preselected sample of elite experts to discuss the subject matter in person, without contact between the experts (Bhattacherjee, 2012, pp. 40, 69);
- **Expert interviews**: Using a preselected sample of relevant experts to discuss the subject matter in person, without contact between the experts (Littig, 2008, pp. 2–3);
- **Delphi studies**: Using a preselected sample of relevant experts to complete a survey on the subject matter repeatedly, with knowledge from past studies (Bourgeois, Pugmire, Stevenson, Swanson, & Swanson, 1948, pp. 1–2);

  **Expert surveys**: Using a preselected sample of relevant experts to complete a survey on the subject matter, without contact between the experts (Zhang & Budgen, 2012a, p. 2); and
- **Expert reviews**: Using a preselected sample of relevant experts to complete a review on the subject matter, without any need for contact between the experts (Allen, Jones, Dolby, Lynn, & Walport, 2009, p. 1).

Expert review is very similar to an expert interview; it can also be viewed as a type of elite interview, if the expert is considered to be elite. Elite experts are more authoritative in their respective fields. The primary difference between an expert interview and an expert review is the level of interaction with the expert. In addition to this, source material is provided in an expert review, to evaluate the subject matter. While this can also occur in an interview, it is not necessarily implied.

The focus of an expert review is to obtain consensus on the subject matter, for example, the validity of a solution. In the expert review, the validity of a solution can be indicated through its utility, quality and effectiveness. This is because of the use of these three research value indicators to assess a design science artefact (vom Brocke & Buddendick, 2006, p. 594).

Due to the higher level of qualitative analysis that is typical of an expert or elite interview or review, there is less need for a large sample of experts. Notably, constraints in finding experts, who can meet the required level of expertise in each evaluated discipline, results in lower sample sizes. Expert review, as understood by this research study, is discussed along with the aforementioned research methods in further detail in Chapter 5.

## 1.7. Chapter Layout

Figure 1.5 indicates the chapter outline for this research project. The figure represents the final structure, which has been through multiple iterations, in favour of a better-suited layout.

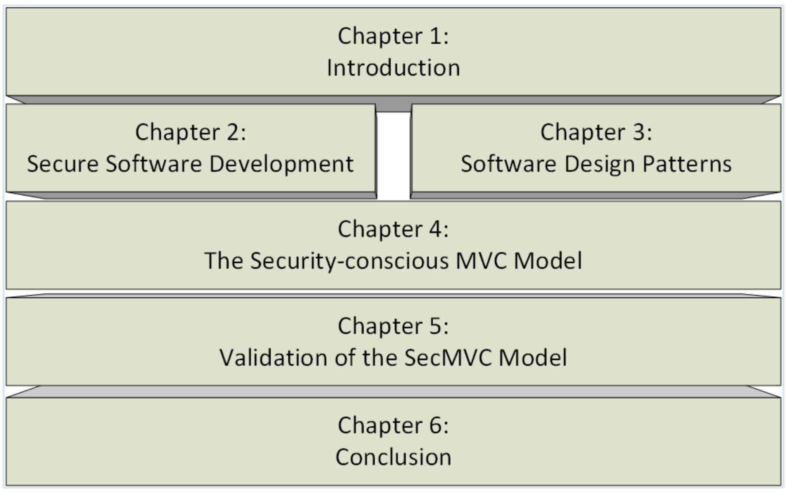


Figure 1.5: Chapter Layout

These chapters follow a drill-down approach, beginning with the specific focus areas and then bringing them together in the SecMVC model. The 'Secure Software Development' chapter approaches security concerns from the overall developmental context; and it refines these into secure software design principles and concepts. The 'Software Design Patterns' chapter discusses software design patterns in general; and it then proceeds to explore prevalent software design patterns, which make up the MVC pattern. After this, the MVC pattern itself and its alternative variations are addressed.

The solution chapter, ‘The Security-conscious MVC Model’, discusses the concepts surrounding secure software design with specific attention to the MVC pattern’s design. The secure software design principles and concepts, which are identified as necessary, are conveyed through a graphical representation, as well as by the supporting text. The SecMVC model is the output for this research.

Chapter 4 also addresses the implementation of the SecMVC model’s general secure software design principles and concepts in specific examples. It is then determined in Chapter 5 how effectively the SecMVC model serves its purpose as a tool for conveying secure software design principles and concepts, and whether this could typically result in a more secure design.

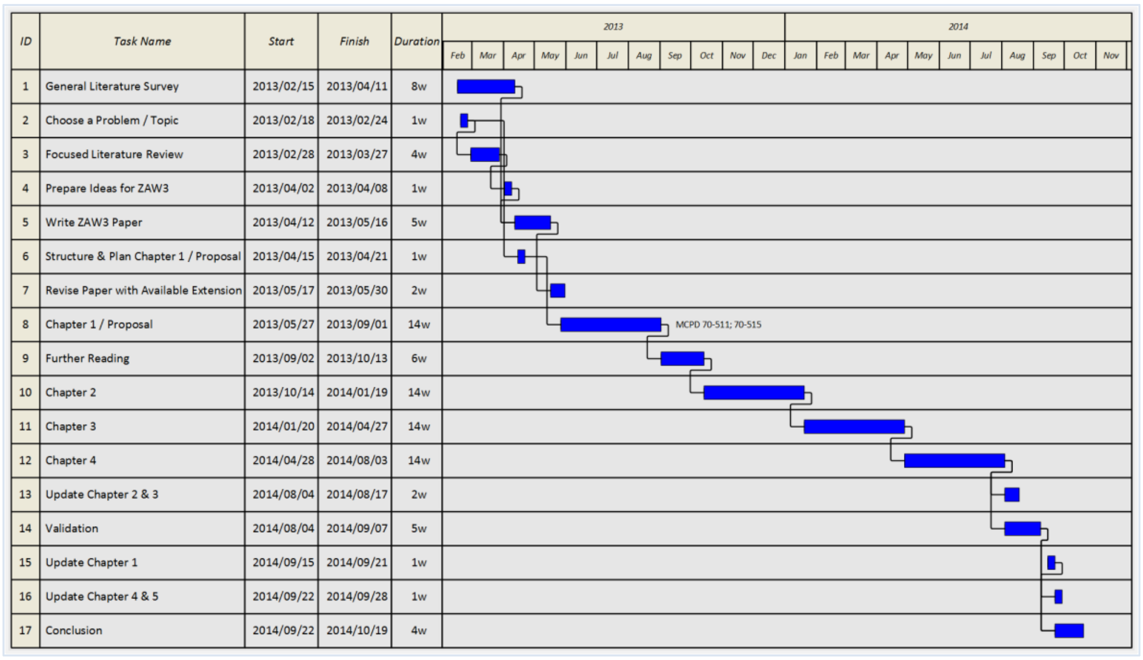
The SecMVC model’s creation is addressed in an iterative manner, bettering the candidate solution over a number of versions for any flaws or oversights found in its analysis. After reaching a favourable output, which is evaluated through expert review, the project concludes with a final chapter. This last chapter summarises the contribution in terms of the SecMVC model, what has been learnt, how the problem and subsequent objectives have been addressed; and it then comes to a close by reflecting, and calling for further research.

## 1.8. Research Plan

The research conducted in this project began in February 2013, with a concentrated and relevant literature review beginning shortly after the conception of the topic. The study allowed for ideas surrounding the identified problem to be constructed for the ZAWWW conference.

Beginning in parallel with some preparation for this introduction chapter and the proposal, the conference paper helped to fully realise the direction of this research; and it, furthermore, constitutes a fair contribution to it. Following the paper’s submission, the proposal was completed. A revised edition was completed prior to submission for evaluation in August 2013. Finally, suggested changes were completed and the proposal was finalised in October 2013.

Figure 1.6 depicts the time spent over the first few months, as well as how the remaining time was spent, after a number of revisions. This was dependent on the structure of the research study remaining unchanged; and thus, it was modified – where appropriate.



| ID | Task Name | Start | Finish | Duration |
|---|---|---|---|---|
| 1 | General Literature Survey | 2013/02/15 | 2013/04/11 | 8w |
| 2 | Choose a Problem / Topic | 2013/02/18 | 2013/02/24 | 1w |
| 3 | Focused Literature Review | 2013/02/28 | 2013/03/27 | 4w |
| 4 | Prepare Ideas for ZAW3 | 2013/04/02 | 2013/04/08 | 1w |
| 5 | Write ZAW3 Paper | 2013/04/12 | 2013/05/16 | 5w |
| 6 | Structure & Plan Chapter 1 / Proposal | 2013/04/15 | 2013/04/21 | 1w |
| 7 | Revise Paper with Available Extension | 2013/05/17 | 2013/05/30 | 2w |
| 8 | Chapter 1 / Proposal | 2013/05/27 | 2013/09/01 | 14w |
| 9 | Further Reading | 2013/09/02 | 2013/10/13 | 6w |
| 10 | Chapter 2 | 2013/10/14 | 2014/01/19 | 14w |
| 11 | Chapter 3 | 2014/01/20 | 2014/04/27 | 14w |
| 12 | Chapter 4 | 2014/04/28 | 2014/08/03 | 14w |
| 13 | Update Chapter 2 & 3 | 2014/08/04 | 2014/08/17 | 2w |
| 14 | Validation | 2014/08/04 | 2014/09/07 | 5w |
| 15 | Update Chapter 1 | 2014/09/15 | 2014/09/21 | 1w |
| 16 | Update Chapter 4 & 5 | 2014/09/22 | 2014/09/28 | 1w |
| 17 | Conclusion | 2014/09/22 | 2014/10/19 | 4w |

Figure 1.6: Research Plan

The chapter layout was also reworked, which constituted changes to the schedule to accommodate it. These changes were relayed in advance to the necessary parties.

## 1.9. Conclusion

This chapter has provided an overview of the research project entitled: 'SecMVC: A Model for Secure Software Design based on the Model-View-Controller pattern'. The research areas, the focus and the topic have been clearly described. A research problem was identified; and this was addressed through the primary and supporting objectives.

This chapter noted that software design patterns are capable of addressing each stage of software development, with insight from attack patterns and countermeasures, although little is currently available on the manner of specification. They are widely used in both industry and education, and are adequate media for the introduction of secure software design principles and concepts.

This research argues for the need of consistent usage of software security at the root of design and development. It also acknowledges the need for consideration at every stage of development. Stakeholders often do not consider the benefits of enhanced security at all stages; and thus, they only address it in the later stages – if at all. It was identified that among these stages, the design and implementation are where experience and skill tend to fall short. This skill shortage and lack of appreciation for security benefits led to the formulation of the problem.

With education and training, software developers, can introduce and maintain better security habits. However, the costs involved in separate and dedicated security courses give motivation for a supplement. ‘Secure Software Development’ is discussed in the following chapter; it addresses the security concerns a supplement would need to take into account, in order to adequately address the focus area.

# Chapter 2: Secure Software Development

## 2.1. Introduction

With the current interconnected state of the world, the security of every system is important. More often than not, however, security is treated independently to software. This overlooks a vital security principle; security should be grounded in the same early design and development stages, as an application itself (Breu, Burger, Hafner, & Popp, 2004, p. 1). By instead favouring immediate returns over long-term benefits, the value of concrete security is commonly undermined.

A well-argued point in information security is its need for human education, training and awareness. Quite often, it is the human aspect, which is the weakest link. While this factor cannot be stressed enough, the technical factors should not be ignored. It does not help if users are well trained to make use of applications securely – if the application itself is not designed with security in mind (Hight, 2005, p. 1). If the underlying code is not inherently secured, then the application's overall ability to resist security attacks falls flat.

The introduction of the .Net Framework, Python, Java and other Runtime Environments, however, one has been able to reduce some security risks. Much of what these managed frameworks reduce is the need for repetitive tasks, such as checks and exceptions (Howard & LeBlanc, 2009, pp. 540–556). However, this does not guarantee secure coding.

A single overlooked weakness can be devastating. Without incorporating security at every point in an application's lifecycle, the potential for oversight is substantial (Breu et al., 2004, p. 1). Without ensuring that it is applied at the roots, the coating of security does not effectively integrate with the entire application.

This brings us back to the original statement. There is a need for human training, education and awareness; however, this applies to developers as well. Developers need to be aware of how easily an insecure application can be exploited, and how heavily such an exploit affects everything around it. Developers, therefore, need proper training in being able to secure their software (Stoneburner, Hayden, & Feringa, 2001, p. 17). This is sometimes catered for, although programmers frequently gain their qualifications without any practical knowledge of secure software development.

The purpose of this chapter is to provide an overview of the high-level security aspects, which need to be considered by software developers, drilling down to progressively more detailed concepts. This collectively, should be able to provide insight into the relation between each aspect, and the resulting output of this research. Because of the nature of the problem this research seeks to address, it is noted that even older works in the field are still largely relevant, as the problem itself has been around for a considerable amount of time.

Figure 2.1 presents an overview of the relation between the notions to be addressed in this chapter, and the resulting output of secure software design concepts under a number of generalised secure software design principles. Over the course of this chapter, the ideas corresponding to this figure are discussed.

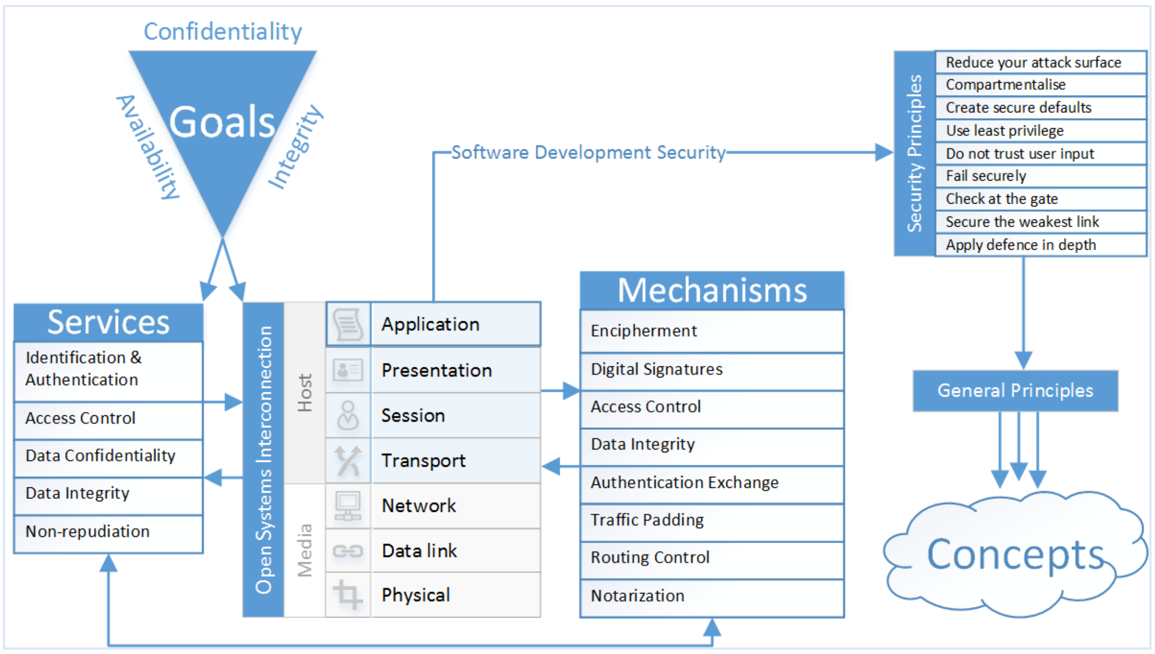


Figure 2.1: Overview of Concept Relations

The sections to follow include the Open Systems Interconnection (OSI) model's approach to security goals, services and mechanisms in relation to the OSI layers. Stemming from this, particularly from the application layer, is the connection to the software development lifecycle and its related participants. Security principles, as they pertain to software developers specifically, are identified, revised and then supported by associated secure software design principles and concepts, which are then discussed in further detail.

## 2.2. Software Security

Security is like a well-trained pugilist in defence. Adapted from Khan and Mustafa (2008, p. 1622), 'secure software' provides solutions, which block the vast majority of attacks, withstand most of what it cannot block, and recovers quickly – with minimal damage.

This notion of security not being an impenetrable wall is one also acknowledged by Bishop and Engle (2006, p. 15). A non-exhaustive list of threat examples, which comprise medium-to-high levels of likelihood, frequency and impact are shown below (ISO/IEC 27005, 2008, pp. 42–45).

- Theft of information;
- Use of a system in an unauthorised manner;
- Use of a system by an unauthorised user;
- False user identity;
- Malicious software;
- User error;
- Repudiation; and
- Software error.

In this, however, there is no silver bullet. A best effort must be made to cover all the bases. Breu et al. (2004, p. 1) suggest that this effort be handled in every stage of the development life cycle. This tactic is again given mention by Jones and Rastogi (2004, p. 1).

In particular, emphasis is made on security being a specific requirement. However, it has been identified that in industry, security is often seen as a diversion from productivity (Jürjens, 2002, p. 2; Khan & Mustafa, 2008, p. 1622). Companies are all too eager to cut development time and costs; and quite often good security is ignored, when it is not explicitly requested.

Instead of designing code securely, structurally from the inside out, programmers try to rely on instinct. This aspiration to successfully implement security as a habit, is one that is commonly left unsatisfied. All too often, developers are not given the training to use such intuition (Khan & Mustafa, 2008, p. 1622).

Furthermore, in a slideshow presentation given by Black Hat's Drew Miller (2003) at a Windows security conference, it was shown how the introduction of managed code in the .Net Framework changes the landscape for application security. Notably, the risks of buffer overflows, code and role access security, as well as the Trojan-horse style virus are found to have been mitigated to a certain degree. Further examples of these include memory corruption and overflows in general, as mentioned in Mano Paul's "The Ten Best Practices for Secure Software Development" whitepaper from the book: Official (ISC)$^2$ Guide to the CSSLP (2011).

To elaborate on this, many exploits are more difficult to produce, when code is in a lowered state of trust. With managed code, buffer overflows would have to be exploiting a weakness of the managed framework, rather than a weakness of the application. It is when unsafe and unchecked constructs are used, that the issue becomes a noticeable threat. According to Miller (2003, p. 5), in properly managed code the introduction of self-resizing variables renders overflows less of a concern.

With the aforementioned lack of training, as well as with the popularity of managed coding languages already mitigating a few of the risks found traditionally, developers are less likely to pay specific attention to security. The following section begins with information security goals.

## 2.3. Information Security Goals

In information security, controls are implemented to protect the confidentiality, integrity and availability (CIA) requirements of a system and its information (National Institute of Standards and Technology, 2011, p. 1). In this context, each aspect can be described as follows (Cherdantseva & Hilton, 2013, p. 3):

- Unauthorised information release (confidentiality);
- Unauthorised information modification (integrity); and
- Unauthorised denial of use (availability).

The CIA triad are referred to by a variety of descriptions, including security attributes, properties, criteria, and goals – to name but a few. This study uses the term 'goals' to describe the CIA triad by way of preference and familiarity, as opposed to the other terms. The year 2013 introduced the Reference Model of Information Assurance & Security (RMIAS). This model uses security 'goals' to describe five additional concepts (Cherdantseva & Hilton, 2013, p. 4). These are: privacy, non-repudiation, auditability, accountability, as well as authenticity and trustworthiness.

Despite the use of these extended concepts, the term information security 'goals' is still commonly used when referring to the original triad.

In the business world, implementing information security goals means creating an information security programme, which leads to the identification of risks, and the mitigation of those risks through technical controls and policies. These need to be re-evaluated continually. However, in the context of a software solution, a different approach can be taken.

As previously mentioned, software development does not always cater for good security. When it does, however, according to Polydys, Ryan and Ryan (2006, p. 56), it should address security at every stage of development. This is essential in maintaining an image of trustworthiness, conformance, and predictability. Without security 'baked in', it is 'difficult or impossible' to provide a secure solution.

When considering the information security goals, we can map these to trustworthiness, conformance and predictability, respectively:

- With confidentiality, a client needs to be able to trust that their data is secure;
- With integrity, the software needs to follow strict information integrity policies, to ensure that it is not modified without the required permissions and authorisation; and
- With availability, functionality should work whenever needed – without misbehaving or disappearing.

These goals are addressed through the information security services found in the Open Systems Interconnection (OSI) Reference Model, as discussed in the following section.

## 2.4. The Open Systems Interconnection Reference Model

The OSI model describes the communication between abstracted 'layers' of a system in communication networks. These layers can represent progressively higher-level forms of data, as one goes up each layer, as shown in Figure 2.2.

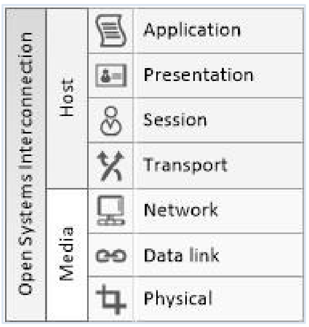


Figure 2.2: The OSI Model (adapted from Miller 2003)

The bottom-most layer describes the physical communication media, signal and binary transmissions. The data link layer handles physical addressing; while above it, the network layer handles logical and port addressing. These make up the media section of the OSI model.

The transport layer handles connections as a whole, while the communicative session layer is commonly handled by web sites and services. The presentation and application layers are the most addressed in code, and need to consider security aspects from not only security mechanisms, but also secure programming. These handle actual data objects and functionality, respectively. This is the host section of the OSI model.

As noted by Rachelle L. Miller (2003, p. 7), the OSI model is not designed to enforce a structure. This study's main focus is on the development related aspects derived from the security services of this model. These security services are discussed in Section 2.4.1, together with security mechanisms in Section 2.4.2.

### 2.4.1. Security Services

In securing Information Systems (IS), there exists a well-used framework, which addresses how software should serve an organisation's security, as well as the means to do so. The security services of the OSI Reference Model in ISO/IEC 7498-2 (1989) are often provided as a basis for the analysis of software security. These are:

1. Identification and Authentication
2. Access Control
3. Data Confidentiality
4. Data Integrity
5. Non-repudiation

These services may be elaborated on in the following way. Firstly, identification and authentication refer to both data origin (that the peer is the source of the data), as well as the peer entity authentication (that an entity is who it claims to be).

Secondly, access control manages what resources are allowed to be allocated. Resources refer to not only the ability to read and write to devices, but also to execute a process or service. Hight (2005, p. 2) mentions that a system's access control is largely affected by how well users play their part in security.

Thirdly, in data confidentiality, to maintain the protection of the data from unauthorised disclosure, specific lower level services are provided, including traffic flow and the entire connection or Service Data Unit (SDU) (International Telecommunication Union, 1991, p. 11). If traffic flow is left open to inspection, conclusions may be derived from it. SDU refers to a single instance of sent data, which has not yet been encapsulated. Within either an SDU or the entire connection, there lie selective field confidentiality – providing protection to only those specific fields that require it.

Fourthly, data integrity refers to the assurance that the data will remain unchanged. This can take place in the same full connection, connectionless and selective field options, as data confidentiality. Data integrity may be checked for a full connection – with or without – the ability to recover from any detected disparities. It may check for a 'connectionless' single SDU. One may elect to only check selective fields in either option.

Finally, there is non-repudiation. This refers to the inability to deny that a transaction took place (Alkussayer & Allen, 2010, p. 98). The service records transactions, so that there is evidence of everything which transpires. This can take place in either or both of the recipient and sender receiving transactional evidence.

This section has discussed the security services used to maintain confidentiality, integrity, and availability on each layer of the OSI reference model. Keeping these security services in mind is important for each layer, although consideration alone is not enough – there must also be support for these services in design and implementation. This is where security mechanisms come in.

### 2.4.2. Security Mechanisms

As described by ISO/IEC 7498-2 (1989) and X.800 (International Telecommunication Union, 1991, pp. 10–12), the following eight mechanisms may be utilised to better deliver the information security services:

1. Encipherment
2. Digital Signatures
3. Access Control
4. Data Integrity
5. Authentication Exchange
6. Traffic Padding
7. Routing Control
8. Notarisation

Encipherment, or encryption, refers to the rendering of all data as illegible. When reversible, encipherment may make use of either symmetric or asymmetric encryption. Irreversible encipherment, however, need not make use of either if desired; this type of encryption can make use of hash functions to irreversibly transform information. In symmetric encryption, a single secret key may encrypt the data, as well as decrypt that same data. Asymmetric encryption, on the other hand, requires two keys: one to encrypt, and one to decrypt. The use of reversible encryption requires strong key management (Swanson, 1996, p. 51).

Digital signatures are identifiers attached to messages to uniquely differentiate between peers. This includes both the signing and verification of signatures. Encryption may also be used to facilitate signatures (Swanson, 1996, p. 57). Like asymmetric encryption, this makes use of both private and public keys. Verification makes use of the public key to deduce whether the message has been manipulated with the private key. Because the private key is only held by the sender, verification of the public key proves who sent the message.

Access control mechanisms refer specifically to the actual allowing and disallowing of access to resources. The relation can be understood by comparing access control to a private establishment's bouncer. If you do not have access rights, you are not granted access. This metaphor can go further in describing access control lists, pre-authenticated passwords, status of the entity, time/route of access and duration – without much elaboration.

Data integrity mechanisms check for equality between data – before and after retrieval. This can be at block or stream level. Data integrity can be ensured through the use of checkvalues or block checks, which may be encrypted – to ensure that no modification has been made. If a modification is identified, this may include automatic recovery, error-correction or retransmission.

The authentication process is handled by authentication exchange mechanisms. This requires typical password usage, but can also make use of encryption and credentials as authentication indicators (Swanson, 1996, pp. 43–45). Authentication may also involve other mechanisms such as with signatures and notaries. Of particular interest in this regard is that of unilateral (two-way) and mutual (three-way) authentication through handshakes. This can be used for encryption to protect against replaying an encrypted transmission.

Combined with encipherment, or a similar confidentiality service, traffic padding mechanisms are used to obscure the patterns of network traffic. This prevents external analysis from detecting important weaknesses in the network. According to Jiang, Vaidya and Zhao (2003, pp. 1–2), traffic packets are spoofed to disrupt any analysis; and this is referred to as a *cover mode*. In doing this, potential attackers can come to false conclusions.

As indicated by Yi, Naldurg and Kravets (2002, p. 1), routing controls may be implemented to bypass less trusted nodes in a network, when secure information is concerned. The ability to dynamically, or through prearranged metrics, determine the most secure route through feedback is the basis of the routing control mechanism (International Telecommunication Union, 1991, p. 12).

As described by Nakahara (2000, p. 1), '[a notarisation] safely stores the evidence of events and actions during trading for a long term[;] and subsequently [it] certifies the fact by presenting formatted evidence (notary token)'. The X.800 states much the same, adding that this evidence includes integrity, origin, time and destination. The notary can be viewed as a third party that is trusted by both persons, handling all transferred information. This communication can also make use of other non-repudiation mechanisms, as well as encipherment.

The focus of this research is on the upper layers of the OSI reference model, specifically presentation and application. This is the area focused on and most influenced by software development architects, analysts, engineers and programmers. While security mechanisms can be utilised by each of these roles, software development security can go deeper than mechanisms. The next section revisits the Software Development Life Cycle (SDLC), and through it, provides further context for this research.

## 2.5. Security in the SDLC

Software development includes multiple stages in most development cycles. Common to these are the analysis, design, and implementation phases. Some examples of an SDLC are the agile, evolutionary, incremental, object-oriented, prototyping, spiral, and waterfall life cycles (Futcher, 2011, p. 149). Regardless of the methodology chosen, however, security needs to be considered throughout.

Within these life cycles, different development roles share responsibility for certain stages. An example of the Software Development Life Cycle, previously discussed in the introduction chapter, is featured along with common development roles and their collective responsibilities in Figure 2.3.

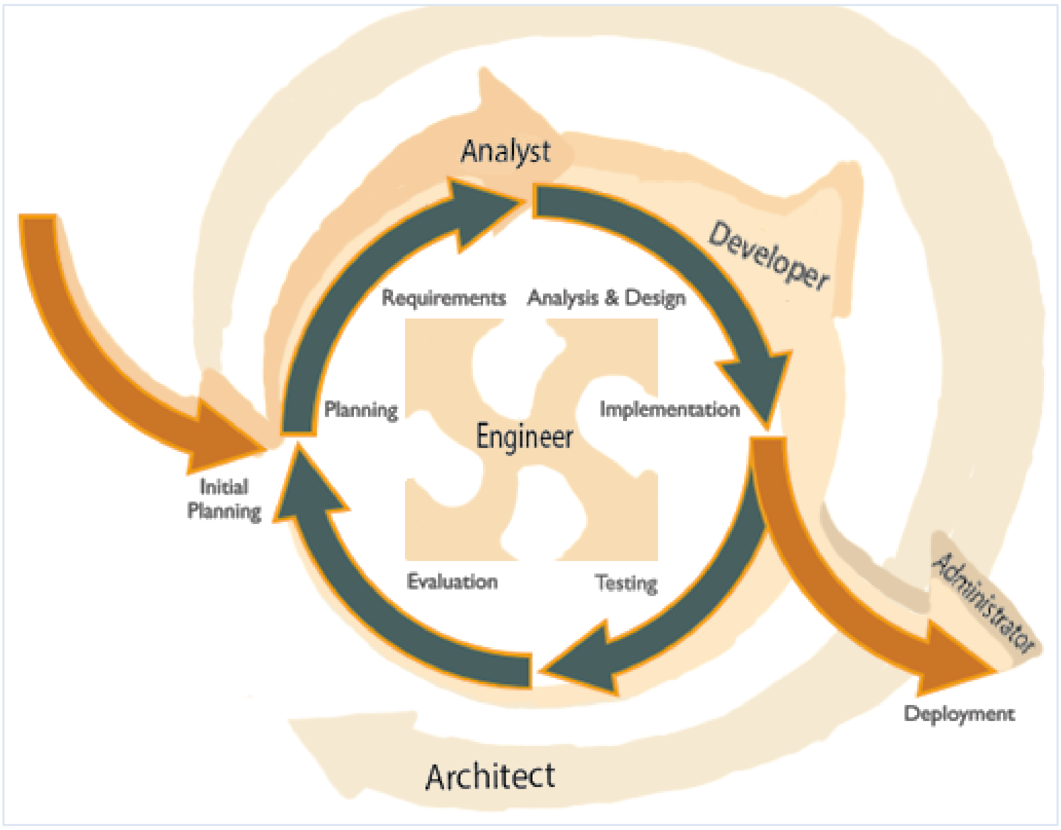


Figure 2.3: Roles in SDLC (adapted from Paul, 2008a)

This study focuses on the software context, in contrast to the system as a whole. The distinction between these lies in a system being more holistic, where comparatively, software is merely a part of a system (Futcher, 2011, p. 129).

Due to changes in industry, there is some overlap in nomenclature between engineers and developers (Anguelo, 2009, p. 1). Essentially, engineers employ computer science in their solutions. Their approach to development can be described as more rigorous and technical, with less reliance on an Integrated Development Environment (IDE). In the context of an SDLC, engineers must take all phases into account at all times. Developer roles, on the other hand, are typically broad, flexible; and as such, they may be assigned along each stage of an SDLC.

Analyst roles typically focus on earlier stages; while on the other side of the cycle, administrator roles typically deal with deployment and the setup of the target system. The most responsible role is that of the architect, who oversees the entire process, considering hardware, cost, schedules, and training (Anguelo, 2009, p. 1). Typically, a developer's scope of responsibility is determined by the employer; however, developers are usually most responsible for design and implementation.

This research focuses on aspects found in the development environment specifically, which constitutes a considerable part of both of these phases.

As mentioned in the introduction chapter, this study aims to reduce the scope to a reasonable size, focusing mainly on design aspects of the Software Development Life Cycle. The scope lies in the field of secure software development, indicated by security professionals to be a significantly important requirement (Ayoub, 2011, p. 25). Furthermore, the second highest training need identified by the same study is in application and system development security, with the highest security concern (81%) directed to design activities (Ayoub, 2011, p. 20). The design phase, where design activities take place, is also a main component of this study's focus.

By directing the focus of this research – specifically on secure software development in the design phase – the aspects pertaining to requirements analysis, the setup of the target system, its configuration, and the configuration of any network hardware are less emphasised. However, as also mentioned, each stage of development should account for security – these aspects should not be ignored and as such their intricacies should be examined by the persons responsible. Not doing so could sabotage any attempts a developer makes towards securing the software.

Furthermore, this research uses .NET as its chosen target framework due to its relevance in the School of Information and Communication Technology (ICT) at the Nelson Mandela Metropolitan University (NMMU). This department has a strong focus on both Microsoft and .NET specifically, being the medium in which software development is taught. It is also arguably one of the most current and well-known managed frameworks; and it is the medium in which the researcher is most proficient.

The next section specifically addresses security in the .NET development environment.

## 2.6. Security Considerations in the .NET Framework

Software can be exposed to various threats, such as the renowned buffer overflow. These threats exploit vulnerabilities in software. However, the .NET framework restricts a good number of these by default, which protects against many online attacks (Howard & LeBlanc, 2009, pp. 540–556; Pirnau, 2013, p. 2). This embedded security makes the developer's life easier. However, this does not mean that a programmer should ignore security considerations, which are handled by their choice of runtime, as its contributions are invalidated if the coder does not use them properly (Howard & LeBlanc, 2009, p. 535).

The .NET framework introduces automatic bounds, type, and pointer manipulation checking, among other measures, to make managed code resilient to buffer overflows (Ferrara, Logozzo, & Fahndrich, 2008, p. 1; Meier, Mackman, Vasireddy, Dunner, Escamilla, & Murukan, 2003, p. 131). This does not, however, prevent issues in the unmanaged Application Programming Interface (API) or Component Object Model (COM) calls.

Calling unmanaged or unsafe libraries should be limited to when absolutely necessary. When used ensure that only validated input is ever passed in. Received output should also be validated. These calls should ideally be in a separate 'wrapper assembly' as well, which allows the unsafe and or highly privileged code to run with isolated and distinct permission requirements (Meier et al., 2003, p. 168; Pirnau, 2013, p. 4). When writing unmanaged code itself, make sure to compile with the /GS switch, as this contributes to the prevention of buffer overflows.

There are further arguments in favour of separating safe managed code from unsafe and or unmanaged code. To take a step further than a mere wrapper, this code can be placed in a separate process altogether, and be protected by kernel level security (Klinkoff, Kirda, Kruegel, & Vigna, 2007, pp. 419–424). The .NET framework enables multiple Application Domains within one process, which can achieve isolation (Pirnau, 2013, p. 6). When using a separate process, the Remote Procedure Call (RPC) can be used (Klinkoff et al., 2007, p. 423). The RPC, however, can only receive by-value parameters.

Attackers can circumvent security in various low level ways. Executable functions can be modified if memory pages are not protected; the Native Windows Library (NTDLL), located just above the kernel space, can be used dynamically, or through in-line assembly code as well (Klinkoff et al., 2007, p. 425). There is even a registry key, which if not protected by the Code Access Security (CAS) policy can be used to force-load a library into any application's start-up procedure. This alludes to the fact that much of what is developed is subject to flaws or oversights, potentially encountered through less attention to detail.

In the development environment it is important that a certain level of control and conformity be maintained, in order to ensure reliable solution quality. In general, this is achieved through standards, guidelines and best practices. These are essentially rules, separated by strictness. Standards require thorough adherence; best practices are situational, although still strict; but guidelines are less so.

The use of these may provide assistance for achieving quality, extensibility, a common language, as well as a consistent format (Futcher, 2011, pp. 131–132). In some cases, enforcing development rules can increase complexity and production time, potentially without significant returns. Furthermore, they can be difficult to implement, or too open to interpretation.

It is, therefore, important that rules, which are set out for developers to adhere to are clear, concise, specific and preferably simple. It is, however, not always the case that standards, best practices and guidelines have these qualities. This is why it is also important that the right rules are used for the situation. Additionally, top-level and management support for the extra costs – in light of the better quality – would allow developers to achieve this quality despite productivity concerns.

In order to prevent attackers from exploiting vulnerabilities, an application must be developed with attention to its level of security. The following section covers the rules whereby a developer may enhance the security of a solution. This is accomplished through an assortment of secure software design concepts, organised under the mantle of four grouped or general secure software design principles.

## 2.7. General Secure Software Design Principles

In general, mechanisms are used to provide higher levels of security in software, communications, and networks. However, these are not all that must be considered when designing and implementing applications. Meier et al. (2003, p. 11) define nine security principles for application developers to adhere to, when designing solutions, as well as each component within them. Based on these principles, this research proposes a revised summary of these depicted as four more general secure software design principles. These are shown grouped, as follows in Table 2.1.

Table 2.1: Summary of Principles

| 9 Principles by Meier et al. (2003) | General Principles |
|---|---|
| Reduce your attack surface | *Least privilege* |
| Compartmentalise | |
| Create secure defaults | |
| Use least privilege | |
| Do not trust user input | *All input is evil* |
| Fail securely | *Output securely* |
| Check at the gate | *Layer your defence* |
| Secure the weakest link | |
| Apply defence in depth | |

In their definition of 'reduce your attack surface', Meiers et al. (2003, p. 11) summarise "…disabling or removing unused services, protocols, and functionality." This correlates very closely with *least privilege*, which merely encourages the reduction of what is available to what is necessary.

Furthermore, their description of 'compartmentalisation' features the question: 'What resources can [an attacker] access?', which essentially aims to get the reader thinking about what can be retrieved, from which section of an application, encouraging the limitation of scope. Two of the three examples of compartmentalisation are *least privilege*, and the third, 'firewalls'. This a tool which can be (and should be) used to achieve it.

There is also 'create secure defaults', which encourages the concern that defaults should be as restrictive as possible. Their description encourages denying and disabling of access by default. This, too, seems to match the principle of *least privilege* very closely. However, their description also includes aspects one might expect in other principles entirely. 'When an error occurs' has the makings of an example one would expect to see under the 'fail securely' principle (Meier et al., 2003, p. 11).

With the aforementioned similarities between 'defaults', 'compartmentalisation' and 'attack surface' principles and the *least privilege* principle, this research proposes that they be grouped under one name to simplify the representation of knowledge into a smaller and, therefore easier to remember, format. The suggested term for this grouping is *Least Privilege*.

A very important principle is not to trust user input. Inputs are an attacker's main point of entry, when trying to exploit or search an application for vulnerabilities. This principle covers a large area, as this is not just limited to the front end user interface. Non-exhaustively, other points of access include query strings, web service calls, using the application's libraries, or ones which reference them with higher levels of access. As noted by Howard and LeBlanc (2009, p. 341), *all input is evil*.

Treating all input as though you know it will be malicious, goes further than mere mistrust. It also stresses the significance of this principle.

The next principle pertains less to exploitation and more to information gathering. 'Fail securely' covers hiding sensitive data after a failure in general, but focuses specifically on error messages (Meier et al., 2003, p. 11). By default, desktop applications show fully detailed exceptions, which while great for development, constitute a serious security flaw, if left that way. Web applications allow custom errors; however, these should not expose any more information than they need to.

This same principle can also apply to logging, especially since exceptions can be logged. Information that is logged should also be recorded securely. If discovered and unprotected, the information within could be used to find weaknesses – or the log could be modified to hide activity. Since not all logging is of failures, although all logs and exceptions are outputs, the principle could be broadened by naming it *Output Securely*.

Three further related principles are 'check at the gate', 'secure the weakest link' and 'apply defence in depth'. The first suggests that authentication and authorisation should occur for each layer, as soon as one can do so, so that no access is provided to an unauthorised (though potentially authenticated) user. The second refers to ensuring that each layer between network, host and application (whichever is weakest) is addressed, and then the process repeated. Adapted from Carlos Lyons: A vulnerability in one would allow for exploitation of the others (Meier et al., 2003, p. 6).

The third, 'apply defence in depth', refers to gatekeepers at each layer – keeping in mind that these should be able to account for each other being compromised.

Because each of these concepts is addressed with the multi-layered nature of in-production software in mind, unlike the more isolated principles, they could be addressed together. This research study suggests that these be grouped under the title *Layer Your Defence*. Although this grouping works towards better simplicity and cohesion, it does not hide detail – by fully replacing the principles.

In the following section, each of these grouped secure software design principles is used as a basis to elaborate on related secure software design concepts. The first of the revised four general security principles is *least privilege*.

### 2.7.1. Least Privilege

Access rights, privileges and trust are very important aspects of software security. A substantial amount of the enhanced security from managed frameworks, specifically the .NET framework, is based on trust (Howard & LeBlanc, 2009, p. 536). *Least privilege* is the most all-encompassing generalised principle that this study describes. The secure software design concepts that it covers are summarised in Table 2.2.

Table 2.2: Summary of Least Privilege Concepts

| Least Privilege |
|---|
| 1. Permit only what rights, access and functionality is necessary – for as little time as is necessary. |
| 2. Decide what trust is necessary using evidence of origin, identity and intent. |
| 3. Restrict and record attempts to achieve access. |
| 4. Isolate the components with varying trust requirements. |
| 5. Isolate assemblies, for web applications use separate application pools. |
| 6. Disable system services, protocols and ports which are not required for applications to function. |
| 7. Provide only what file system, registry, event and application log access is required. |
| 8. Provide only what read, write, create, modify or delete permissions are required. |
| 9. Limit what sensitive information is maintained in the file system, registry, or logs. |
| 10. Limit the readability of the necessary sensitive records through obfuscation and encryption. |
| 11. Regularly relocate logs to a remote inaccessible location for analysis and review. |
| 12. Do not leave code open to reverse engineering. |
| 13. Strong name your assembly for deliberate restriction. |
| 14. Use Code Access Security Policy for general restriction. |
| 15. Use Code Access Security in code for detailed restriction. |
| 16. Use Code Access Security at class or method level declaratively if it does not need to be dynamic. |
| 17. Avoid partial stack walks as these are susceptible to luring attacks. |
| 18. Avoid the use of 'deny' and 'permit only' as these may be bypassed. |
| 19. Avoid 'demand' unless required, if used isolating it and following the Demand/Assert pattern. |
| 20. Avoid security sensitive calls in threading which is susceptible to race conditions. |
| 21. Avoid inheriting unneeded security context in threads, for instance through impersonation. |
| 22. Wrap functionality with special security permissions, such as unmanaged code, in isolated assemblies. |
| 23. Ensure that whatever calls a critical function has the authorisation, credentials and rights to do so. |

The principle of *least privilege*, for one, is not only to provide an entity with as few rights as possible (Meier et al., 2003, p. 11); but it also requires that access be permitted for the shortest duration possible (Howard & LeBlanc, 2009, p. 536). This goes further in allowing deliberate restriction and demanding of access when an application is strong-named.

The caller must be able to provide evidence that it is who, or what, it says it is (Howard & LeBlanc, 2009, pp. 109–118). Even so, its access may only be granted with certain levels of trust. Every request should be recorded; attempts should be restricted; sessions should timeout; and this should happen in a manner, which does not permit abuse.

The idea of limiting privileges applies especially to web development, because web functionality is primarily based on partial (medium) trust. Now, the recommended procedure is to keep a web application isolated, which is best achieved through having a separate application pool (Microsoft, 2012).

This applies especially to system services, which should each be disabled on a host, unless they are necessary (Meier et al., 2003, pp. 8–9). The application should restrict the usage of protocols and ports to a small subset of what is needed for effective use. This includes access to the registry, event log, file system, and any application logs as well, where only as little access as strictly necessary need be given.

Prevent potential exploitation of the ability to write; attackers may attempt to flood the log into overwriting old entries to enable repudiation. Logs should be backed-up and moved to a remote inaccessible location – for continual analysis and review.

Registry access is typically split between Local Machine and Current User, where Local Machine may be accessed by any process, unless Access Control Lists are used (Meier et al., 2003, pp. 166–167). In writing, encrypt sensitive information – if not entirely omitting it. This encryption can be achieved through the Data Protection API.

There are also additional security considerations when threading is involved. Because threading is susceptible to race conditions, it is important that no security sensitive calls are made, based on conditions that are influenced asynchronously (Meier et al., 2003, p. 171). In general, functions are safer when synchronised. This also applies to any static constructors and dispose methods, which if asynchronous, are also at risk. Furthermore, threads may or may not require the same security context as their parent. Usually, this is based on the parent thread; however, if impersonation is used, then the context is not inherited.

Rights to access should not be assumed. In all points of access, there must be stringent procedures to verify that whatever calls a server-side business logic function has the authorisation, credentials and right to do so.

Code Access Security is a very prominent aspect in relation to .NET permissions based protection. It provides application wide permission filtering in CAS Policy, as well as in code permissions, at class or method-level declaratively, or imperatively (Pirnau, 2013, p. 4).

The advantage of CAS is that it is fairly atomic. Using 'evidence' gathered from origin uniform resource identifiers (URI), zones, signatures and keys, the 'code group' is determined and used to match a 'security policy' (Chu, 2008, pp. 73–74).

The different permissions, which can be granted, or withheld, are numerous (Pirnau, 2013, p. 4). From granular data, registry and file access to Domain Name Service (DNS), sockets and message queues. Of particular interest is the SecurityPermission, which includes flags for unmanaged code, reflection, serialisation, and remoting, as well as various others (Meier et al., 2003, p. 194). The classes used are generally suffixed with 'Permission'.

At the call level, this features 'stack walks', dependent on what kind of 'demand' is used (Pirnau, 2013, p. 4). The standard demand requires a full stack walk, which is the safest action. There is also a link demand – which only checks the immediate caller. This makes it vulnerable to a 'luring attack', where a trusted caller is used in-between (Chu, 2008, p. 75). In CAS, code can 'assert', 'deny' or 'permit only' certain permissions (Meier et al., 2003, p. 203). An 'assert' effectively ignores the stack walk, making its use very dangerous; and 'deny' and 'permit only' can generally be bypassed (Chu, 2008, p. 75).

There are still uses for 'assert', although these should be isolated, and should follow the Demand/Assert style of implementation.

This involves first demanding permission, before ‘assert’ can be called, and then reverting the ‘assert’ as soon as possible – specifically in a ‘finally’ block to ensure that it always happens. The proceeding section covers ‘do not trust user input’; in its generalised principle form, *all input is evil*.

### 2.7.2. All Input Is Evil

The top four weaknesses, as defined by Bob Martin et al. (Martin, Brown, Paller, Kirby, & Christey, 2011) are SQL (Structured Query Language) injection, operating system command injection, buffer overflows and cross-site scripting, respectively. These all correlate to malicious input. A summary of this principle’s secure software design concepts is shown in Table 2.3.

**Table 2.3: Summary of All Input Is Evil Concepts**

| All Input Is Evil |
|---|
| 1. Suspect all input of being potentially malicious. |
| 2. Avoid open-ended input, instead providing a choice of defaults or a dynamically populated list. |
| 3. Permit only what characters could potentially be needed, rather than only filtering out invalid ones. |
| 4. Do not provide unlimited or unnecessarily large length bounds. |
| 5. Sanitise and validate input before processing it. |
| 6. Sanitise any input which might be tampered with, including hidden fields and cookies. |
| 7. Ensure that input is filtered, escaped, quoted or encoded where appropriate. |
| 8. Make use of any additional precautions offered by server and browser features. |
| 9. Both requests and responses should be encoded and sanitised. |
| 10. Do not rely on the integrity of client-side validation or script. |
| 11. Canonicalise filenames using Path.GetFullPath(). |
| 12. Interact with databases and other external storage in a predefined and strict manner. |
| 13. Ensure that transmitted data is not compromised by utilising encryption and encrypted checkvalues. |
| 14. Be wary of input through non-standard channels. |
| 15. Functions which accept generalised parameters, such as delegates, should account for maliciousness. |
| 16. Delegates should not be exposed unless the assembly is strong named without partially trusted callers. |
| 17. Serialised data should not contain sensitive information, and should not use partially trusted callers. |
| 18. Dynamic loading should avoid input from untrusted sources, unless using exceedingly strict validation. |

These vulnerabilities can, however, be reduced (Martin et al., 2011). First and foremost, suspect all input as being malicious; favour drop-down lists over combo boxes; and reduce any and all freedom for expression where possible, rather whitelisting than blacklisting.

Avoid the use of file names for input, as these can be interpreted in various ways (Meier et al., 2003, p. 29). If these must be used, then sanitise and validate them before use, instead utilising absolute paths, if possible. The Web configuration includes requestEncoding and responseEncoding attributes, which can limit input representation.

Environment Variables can also be modified; as such do not rely on them. Input for filenames specifically should be canonicalised on top of other validation, which can be done using Path.GetFullPath() (Meier et al., 2003, p. 164).

This includes escaping, encoding and quoting characters properly, using validation, and cleansing input from all sources, even hidden fields, cookies, and URIs. These tactics are commonly utilised by browser and website server software. This includes *HttpOnly* session cookies for Internet Explorer and FireFox, as well as the *filter* option for Apache Struts (Martin et al., 2011).

In the interests of further protection from injection, make sure that all calls to the database are either stored procedures, previously prepared statements, or assigned only by parameter. Ensure that retrieved data is not modified during transmission. This is done by creating checkvalues, which uniquely identify the data. Note that checkvalues should be encrypted, though, since a checkvalue can sometimes be recalculated.

Any user interaction needs to be closely checked against malicious input. This is best handled by limiting the scope for unexpected submissions. In validating and again ensuring that received information conforms to strict encoding, applications may protect against these threats. The same applies to the client side code. An application should not assume that all script is legitimate.

Inputs, which can be of concern in .NET in particular are delegates (Meier et al., 2003, p. 169). When a delegate or event is exposed by an assembly, virtually any function can be registered to it. In order to best protect from this threat, either do not expose delegates, or use strong naming without the AllowPartiallyTrustedCallersAttribute.

The same principle applies to reflection (Meier et al., 2003, pp. 172–173). Dynamic loading with reflection should avoid using input from untrusted sources. If used, it should validate any input and ensure that it in no way could negatively influence the code, type or assembly.

Serialisation is another aspect, which could be vulnerable (Meier et al., 2003, pp. 170–171). Serialised data should not contain sensitive information, and should be validated. However, if there is a need to serialise a class with sensitive items, the [NonSerialized] attribute could be added to the item.

Alternatively, the class can instead implement the ISerializable interface in a way which omits sensitive information. Again, serialised objects exposed by an assembly should ideally be accompanied by strong naming – without, the AllowPartiallyTrustedCallers-Attribute declaration. The following section addresses 'fail securely', but includes logging in its scope, in addition to exception handling – for the generalised principle.

### 2.7.3. Output Securely

Exception handling is an integral part of application development. If it is not managed, it provides an easy way for attackers to gather information, or to leave the application in an otherwise insecure state. Table 2.4 provides a summary of the secure software design concepts detailed in this section.

Table 2.4: Summary of Output Securely Concepts

| Output Securely |
|---|
| 1. Omit sensitive information and insecure details from all outputs of an application. |
| 2. Where sensitive information must be stored it may instead be cross-referenced, or encrypted. |
| 3. Access to each output should only read or write what it has to, if it has to, where it has to. |
| 4. Centralise all logging, providing only useful secured information. Do not log excessively. |
| 5. Log at multiple levels of detail for both easier analysis and better security for less verbose records. |
| 6. Logs should be regularly transferred to further secure locations, such as off site. |
| 7. Centralise, manage and handle exceptions, logging usable, but secure information. |
| 8. Notify the user of the failure, and any helpful information for the user, without implementation details. |
| 9. Do not allow exceptions to relay debug information to the user. Turn on custom errors in ASP.NET. |
| 10. Do not use exceptions in place of logical checks, or for expected errors – do not ‘catch and release’. |
| 11. When using Visual Basic .NET filters, take into account that these may execute prior to the finally block. |

Failures must be handled in a structured manner, without logging or revealing sensitive information. Furthermore, where possible, these exceptions should not be passed on to the client, instead of being handled on the server (Meier et al., 2003, p. 122).

When an error requires user notification, this must be custom tailored to what a user should see. For ASP.NET applications, do not leave the custom errors mode assigned to “Off” (Christey, Kenderdine, Mazella, & Martin, 2013, p. 30). It is beneficial to centralise both logging and exception management. The Common Weakness Enumeration (CWE) suggests that all security-related successes and failures be logged at multiple levels of detail (Christey et al., 2013, p. 1144).

However, it is noted that excessive logging and exception checking can be counterproductive, obscuring possible anomalies, or affecting system performance. Exceptions should not be used in place of programmatic checks, as this can reduce performance or impede cohesiveness (Christey et al., 2013, p. 656).

Exceptions should be handled structurally, making use of the built in try, catch and finally blocks (Meier et al., 2003, p. 162). Exceptions should not have empty catch blocks (‘catch and release’), or simply throw the exception instance (‘catch and throw’), as these are bad practice and could obstruct future developers. In Visual Basic .NET, filters are also provided, although these should be used with care. Filters higher on the stack can execute before a finally block, briefly avoiding any state change performed there.

While users need minimal exception information, a greater level of detail can be logged, so long as it does not include sensitive data. Important or highly relevant sensitive data can be encrypted, obfuscated and referenced for later use; although it is important that logs containing such data should be stored at less accessible locations, such as off site.

Access to the File System, Registry, Event Log, and any other logs, should follow *least privilege* and only read or write if it has to, to only where it has to, and only what it has to (Meier et al., 2003, pp. 164–166). The FileIOPermission, EventLogPermission and RegistryPermission declarative attributes or imperative objects enable a granular control over what access the code has.

Any information that is written to these areas needs to be stripped of any insecure details, which might help an attacker to gain undesired knowledge. Omit sensitive information (Meier et al., 2003, pp. 164–166); this applies to all outputs of an application. The next section covers 'check at the gate', 'secure the weakest link', and 'apply defence in depth'.

### 2.7.4. Layer Your Defence

The fifth most overlooked vulnerability, according to the CWE top 25 (Martin et al., 2011, pp. 3–4) is the 'missing authentication for [a] critical function'. Following closely, are 'missing authorisation' and 'hard-coded credentials' at sixth and seventh place, respectively. The secure software design concepts covered by this principle are shown in Table 2.5.

Table 2.5: Summary of Layer Your Defence Concepts

| Layer Your Defence |
|---|
| 1. All critical functions should have their callers authenticated and authorised. |
| 2. Determine what level of access should be required for each area in an application. |
| 3. Deny by default. Firewalls help achieve this. |
| 4. Routers should use restrictive footprinting responses. |
| 5. Make use of Secure Sockets Layer, Internet Protocol Security and a segmented network. |
| 6. Incoming external packets with internal Internet Protocol addresses should be denied. |
| 7. Outgoing packets with invalid internal Internet Protocol addresses should be denied. |
| 8. Make use of Intrusion Detection Systems to prevent Denial of Service attacks. |
| 9. Make use of the available Windows registry settings to adjust timeouts, backlogs and queues. |
| 10. Credentials, keys and authenticating functions should not be 'hard-coded' into an application. |
| 11. Encrypt sensitive data. |
| 12. Only use reversible ciphers for information which needs to be recovered. |
| 13. Authentication measures should make use of well-known tried and tested framework solutions. |
| 14. Authentication must take place at each entry point, as well as on the server as opposed to on the client. |
| 15. Ensure that any interaction with the host operating system is secured and isolated. |
| 16. Sign the application with a strong name – this allows checking for integrity and pointer manipulation. |
| 17. Seek full trust so that a strong name is possible, partial trust is no longer necessary for web applications. |

The first refers to only guarding the 'front door' (Martin et al., 2011). As with access control, assess what areas require what role authentication for access; otherwise, deny by default (Meier et al., 2003, p. 11). When designing authentication measures, stick to using known solutions provided by the framework, and authenticate on the server rather than the client.

For automatically denying access, turn to application firewalls. Firewalls can be very advanced, potentially capable of analysing communications and intelligently restricting access. They, however, are typically limited to application, protocol and port access (Meier et al., 2003, pp. 3–7). Question how the application interacts with the host operating system. Each function, which does so, constitutes a potential vulnerability.

Similarly, the router should be limited to restrictive footprinting responses only (Meier et al., 2003, pp. 19–20). Secure Sockets Layer (SSL) and Internet Protocol Security (IPSec), combined with a segmented network and strong physical security, can prevent sniffing. To prevent spoofing, deny incoming external packets, which possess an internal Internet Protocol (IP), as well as outgoing packets, which possess invalid internal IPs. There are Intrusion Detections Systems (IDS), which are capable of preventing Denial of Service (DoS) attacks. However, Windows does possess registry settings to manipulate Transmission Control Protocol (TCP) connection timeouts, backlogs, and queues as well.

Although less severe than missing authentication, storing hard-coded access is forbidden; it provides another back door for the determined hacker. Where it is identified to be sensitive, data should be encrypted with tried and tested algorithms. In the case of passwords, however, there is no need to have a reversible cipher. Store the hash of the password and then hash whatever entry requests comparison; since their values would be the same. This and all other business logic should be kept confidential as well.

The CWE top 25 lists 'missing encryption of sensitive data', as the eighth most common weakness (Martin et al., 2011, pp. 3–4). At nineteenth position, is the use of a 'broken or risky' algorithm. Similarly, in Howard and LeBlanc's Writing Secure Code (2009, pp. 259–269), mention is given to the unintentional use of a predictable random generator for encryption.

In .NET for one, the System.Security.Cryptography namespace should be used, rather than any random function (Meier et al., 2003, p. 174). On the other hand, while it is typically encouraged to use the Data Protection API for key management, there is evidence that this may no longer be the most secure option (Burzstein & Picod, 2010, p. 7).

One recommendation that is still very effective is the signing of an application with a strong name. By doing this, the Command Line Runtime (CLR) can ensure that the executable maintains integrity, allows additional security checks, and does not have external pointer manipulation (Meier et al., 2003, pp. 130–131). Strong naming also allows run-time security decisions, which are determined by the code that calls them. Although using this requires an application to have full trust, partial trust is no longer the go-to strategy for a web application to be isolated (Microsoft, 2012).

In general, the use of structured frameworks, which protect against traditional threats, is recommended. In addition, every aspect adding to the difficulty to exploit a weakness is helpful.

## 2.8. Conclusion

Information is an important asset to organisations, government, and individuals alike. Its protection is, therefore, critically essential. Information Security protects information through ensuring its confidentiality, integrity and availability. This is accomplished through services and mechanisms on various levels in the OSI model. The application level is primarily handled by software developers, who often are not encouraged to consider security; and therefore, it is not always approached securely. To resolve this problem, developers should be practically conditioned to consider secure software design principles and concepts when designing and programming solutions.

This chapter has examined security from the top level to finer, low-level detail in the software development security domain. Technical aspects surrounding implementation have been covered broadly, and a focus has been made on the grouping of secure software design principles and their underlying concepts.

A revised and simplified list of secure software design principles has been compiled to aid in structuring secure software design concepts, and for conveying them more efficiently. This research has defined four secure software design principles, as opposed to nine. Their descriptions are concise, and still not overly long. Therefore, remembering them, and their underlying concepts should be easier. The revised principles include:

- *Least Privilege* (includes 'defaults', 'compartmentalise' and 'attack surface');
- *All Input Is Evil* (greater than mistrust);
- *Output Securely* (specifically exceptions and logs); and
- *Layer Your Defence* (includes 'check at the gate', 'weakest link' and 'in depth').

In the following section, approaches to software design are described in the form of software design patterns. These patterns hold time-tested solution schemas to common problems, although they also convey underlying secure software design principles. These principles and their associated concepts, where relevant, are also covered.

# Chapter 3: Software Design Patterns

## 3.1. Introduction

In software development, software design patterns are used to provide a guideline for the resolution of commonly occurring problems (Ali & Elish, 2013, p. 1). Some problems are opportunities – where a schema of prior knowledge, developed from experience, can be applied – to result in a core functional and yet unique solution (Zhang & Budgen, 2012b, p. 3).

The experience needed to recognise these opportunities comes from becoming familiar with the pattern. When experts discover new patterns, they may choose to document them, allowing others to sooner gain the same understanding of the pattern (Ali & Elish, 2013, p. 1). Although they have been around for a while, initially as mere illustrations and descriptions, design patterns are still gaining popularity (Khwaja & Alshayeb, 2013, p. 1).

Zhang and Budgen (2012b, p. 3) note that the viability of design patterns relies on two assumptions: Firstly, that a knowledge schema can be represented in a way whereby others can acquire enough understanding to be able to identify and utilise implementation opportunities; and secondly, that doing so would lead to better, and more sustainable designs.

Potential advantages investigated by Zhang and Budgen (2012b, p. 3) include increased productivity, programming ability, use of best practice, and better communication through a shared vocabulary – where potentially complex ideas can be conveyed in a few words. Ali and Elish (2013, p. 1) also identified these. However, Khwaja and Alshayeb (2013, p. 1) add reusability and overall software quality to the possible benefits. Ali and Elish (2013, p. 1) describe software quality as supporting: 'maintainability, reliability and [lowered] fault-proneness'.

Whether a pattern adequately conveys the information needed to replicate its implementation appropriately could depend on two broad factors (Zhang & Budgen, 2012b, p. 3): Firstly, whether it is defined clearly, concisely, and consistently compared to other patterns; and secondly, whether the user is capable of interpreting and relating to it.

Once the user has transferred this information to their own knowledge schema, they can learn where it best applies and when not to use it – based on a building experience. Zhang and Budgen (2012b, p. 1) note that the ability to use a pattern effectively is more quickly transferred to experts. Since experts have spent more time gaining knowledge in this domain, they are generally better equipped to learn quickly when presented with knowledge in a similar format. On the other hand, improper or unneeded usage of a pattern can lead to more negative effects than positive (Zhang & Budgen, 2012b, p. 17).

The purpose of this chapter is to introduce software design patterns; firstly, by how they are formally represented and maintained; and then individually with further detail. In these introductions, the patterns are explained and compared with one another. This leads up to the main focus for this study in software design patterns; the Model-View-Controller (MVC) pattern.

This chapter aims to provide a background to the MVC pattern and its components, as well as its alternate versions. While each of its variations holds the benefit of providing component patterns for the simple impartation of modelled ideas, the MVC pattern is the original. As such, it is the one which others derive from. It is more general, more well-known; and therefore, it is the choice of this study.

The next section describes the means whereby patterns are shared, introducing the 'Gang of Four' and their organisation of software design patterns. It also introduces the idea of a design pattern specification language (DPSL), which defines the means whereby a pattern is shared with the development and academic community.

## 3.2. Specification

Early design patterns in software development began prior to the well-known 'Gang of Four's (Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides) twenty-three software design patterns in 1994 (Gamma, Helm, Johnson, & Vlissides, 1994, p. 22; Khwaja & Alshayeb, 2013, p. 1). There was initially Cunningham and Beck's Smalltalk-specific design pattern language, as well as Coplien's catalogue of C++ 'idioms'. The 'Gang of Four's patterns, however, were made to be platform agnostic – aside from their reliance on object orientation. These twenty-three software design patterns and their type, scope, and relation to the MVC pattern, together with secure software design concepts, are represented in Figure 3.1.

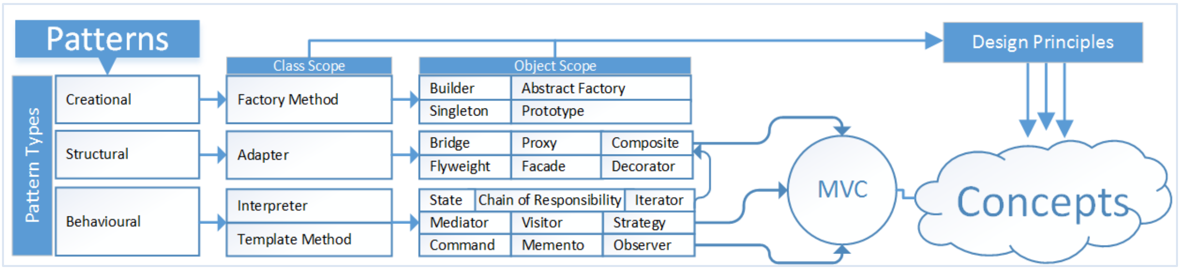


Figure 3.1: Overview of Software Design Pattern Relation Concepts

A design pattern specification language (DPSL) is a structured method of documenting a pattern in a specific format. There are various types of DPSLs, including definition/description, detection, verification/validation and illustrations (Khwaja & Alshayeb, 2013, p. 2). In extension to this, a DPSL may be structured to account for a specific 'basis' or 'notation' of the patterns from which it has been tailored to. For example, it may be based on a mathematical formalisation or a structured modelling language; or it could focus on a purely textual or graphical notation.

A DPSL commonly makes use of Unified Mark-up Language (UML), which illustrates a number of design characteristics in a structured and formalised way (Khwaja & Alshayeb, 2013, p. 3). Notably, patterns are often described through class diagrams. These are illustrations, which denote the composition of virtual blueprints for objects and their interactions with one another. In addition to this, a DPSL may include highly graphical support, which also does not conform to UML. There are times when a pattern may be more efficiently conveyed with a non-standard illustration, instead of a class diagram.

Class diagrams are to be the first method of demonstration in use for this section, which focuses on areas around the MVC pattern. Each of these contained patterns in the Compound MVC pattern are featured in the original twenty-three 'Gang of Four' patterns. As such, each is a well-known pattern. As it happens, they are also relatively widely used, especially in the case of the Observer pattern (Zhang & Budgen, 2012a, p. 7).

Using concepts pertaining to object orientation and the nature of classes and objects, such as composition, this next section discusses both the Composite pattern, the Iterator and the Compound.

## 3.3. The Composite Pattern

The Composite pattern is a structural pattern (Ali & Elish, 2013, p. 5). That is, its usage primarily influences the design – and to a lesser extent – the functionality. The idea behind the pattern revolves around the fact that an object is capable of containing other objects (Freeman, Robson, Bates, & Sierra, 2004, p. 356). In the case of the Composite pattern, the object which contains, and the object which is contained, are both derived from the same class – Component.

Each composed object is capable of further holding another inside it – similar to the Russian Nesting Doll concept (the Matryoshka Principle). An understanding of recursion is beneficial for truly grasping the pattern; and thus, many consider this pattern to be crucial (Zhang & Budgen, 2012a, p. 12).

Eventually, there are no further objects contained; and the object does not need to account for any contained versions. Only its own implementation is of concern. In the Composite pattern, this non-container is referred to as a 'leaf' (Freeman et al., 2004, p. 358). This logic stems from the idea that the composite is like a tree, either branching towards a single leaf, or to another branch (composite). The component can be either of these, as shown by the class diagram in Figure 3.2.

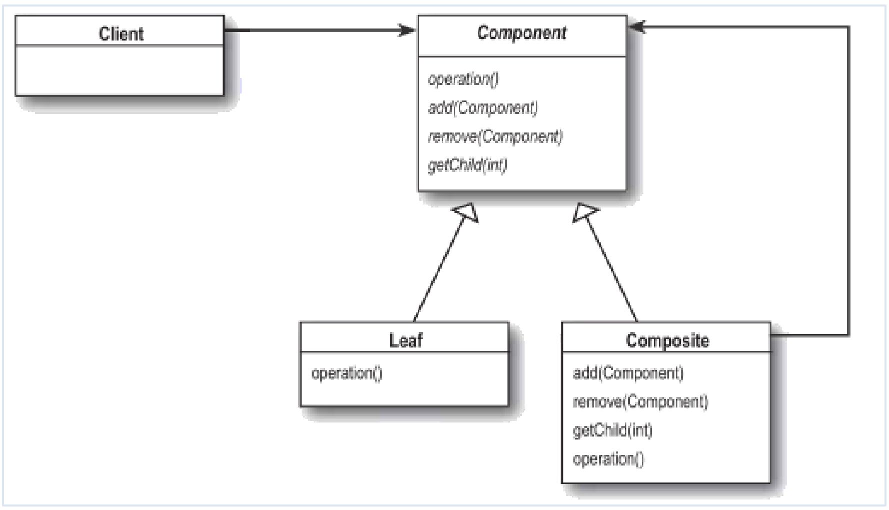


Figure 3.2: The Composite Pattern (adapted from Freeman et al. 2004)

On the other hand, the composite objects (the ones containing others) need not worry about the leaves' various implementations. A composite object must relay calls to its contained children; and it may do so – in addition to implementing its own logic – if need be. Typically, it should also be able to add and remove either leaves or other composites – which between these does not matter – since both are components. These are held internally and privately, although they can be retrieved individually, if implemented through a function (getChild(int) in Figure 3.2).

The implementation of whatever functions are needed by individual leaves is ensured by the containing composite object. While, if only one object were ever contained, it would merely be a means of relaying the call; since the Composite pattern allows for a composite to contain multiple components. This requires the use of an iterating function, or loop, to relay the call to each child component.

Each programming language typically has its own implementation for this, a number of which, contain the 'foreach' loop. This, instead of using an explicit number of loops, only iterates for the list's number of items. Alternatively, for example in a linked list, an enumerator can sequentially retrieve the next item in the list – until no item is referenced as being next. Either of these may be used to iterate through the items contained in the composite for any functions, essentially being instantiations of the Iterator pattern (Freeman et al., 2004, p. 363).

While not complex enough to require a separate section, the Iterator pattern is nonetheless featured within the Composite pattern. It is a behavioural pattern (Ali & Elish, 2013, p. 5). Behavioural patterns are more concerned with function than form. The last of the three types of patterns are creational patterns, which focus on augmenting object instantiation to better suit the situation. The Compound MVC pattern does not typically feature creational patterns; although it could do so, if this were required and suitable.

As mentioned, there are various implementations of the Iterator already in the underlying managed frameworks – almost rendering the pattern self-explanatory. The Iterator provides a means whereby it is possible to access each item in a collection sequentially, although it does so without revealing the underlying implementation (Freeman et al., 2004, p. 336).

Patterns are, therefore, as seen in the Composite pattern, capable of containing not only multiple concepts, but also other entire patterns. A pattern, which does this, is known as a Compound pattern (Freeman et al., 2004, p. 500). The primary pattern this research focuses on, the MVC pattern, is one such Compound pattern. The pattern this section has just covered, the Composite, is one of three main patterns, which the MVC pattern contains. Another of these is the Strategy pattern.

## 3.4. The Strategy Pattern

A behavioural pattern, the Strategy pattern, is another that can be explained through object interactions (Ali & Elish, 2013, p. 5). Not only can an object contain another, as the Composite has shown, it can also do so without being dependent on precisely what object that is. When a class defines an object as one of its components, it may contain any object which inherits, implements, or actually *is* the defined object.

With this logic, a class, which should hold a generalised object, can still hold a more specific object, which is essentially still part of the same hierarchy. Whether one or the other is used, it can still hold the same state and perform the same functions; so, the containing class is able to use these interchangeably. The Strategy pattern focuses on modularity, in the same way as an application can receive different extensions, add-ons, and swap libraries – as long as they conform to an expected standard (Freeman et al., 2004, p. 24).

In the Strategy pattern, the standard to conform to is in the form of an interface (Freeman et al., 2004, p. 13). Classes, which implement the Strategy pattern, in essence, contain defined instances of the specific interfaces. This can be seen in Figure 3.3. The developers then use them without worrying about what object is actually contained – so long as it implements the interface.

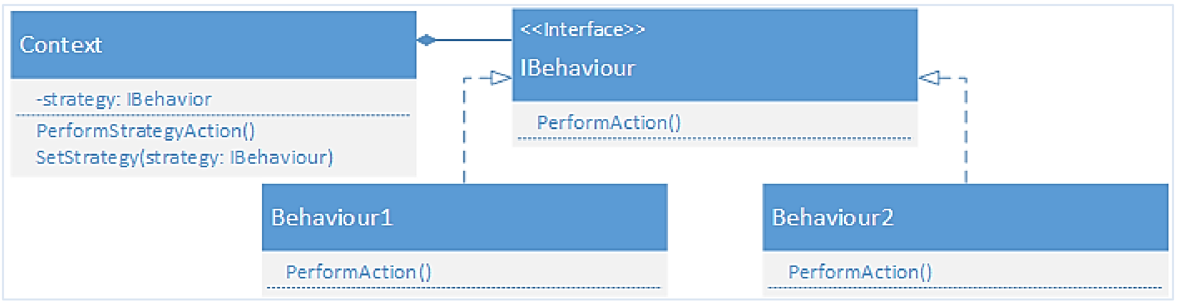


Figure 3.3: The Strategy Pattern (adapted from Freeman et al. 2004)

Freeman et al. (2004, p. 32) show that software development best practices, in the form of object-oriented principles, can be conveyed through design patterns. In the previous section, the Iterator pattern was mentioned. Freeman et al. (2004, p. 339) show through the Iterator that classes should limit responsibilities to avoid overly frequent and uncontrollable changes.

In the case of the Strategy pattern, there are three principles: Encapsulate changing variables; rather compose than inherit; and consider the interface before the implementation (Freeman et al., 2004, p. 32). Freeman et al. (2004, p. 32) show that not all inheritable features are well suited to each child. That is, aspects which vary between the inheriting children of a class should be kept separate. Strategy is one way of achieving this, while still gaining the benefits of polymorphism. It also shows how 'has-a' can be used instead of 'is-a', which is a lot less restrictive.

Finally, by considering the interface first, the Strategy pattern may be easily extended separately, without needing any implementation details of the client. It need only focus on its own function; and this makes it both flexible and reusable.

This section has described the Strategy pattern, showing how it encourages reuse. The next section's focus, and the final pattern in the Compound MVC pattern, promotes loose coupling as well (Freeman et al., 2004, p. 74). The Observer is the next pattern this chapter covers – before delving into the MVC pattern itself.

## 3.5. The Observer Pattern

The Observer pattern is a behavioural pattern, much like both the Strategy and the Iterator pattern; it supports maintainability; and it can diminish fault proneness (Ali & Elish, 2013, pp. 5–6). The Observer pattern, shown in Figure 3.4, shares a substantial commonality with what object-oriented languages generally describe as 'events'. When a key is pressed, or a mouse moves, hovers, or clicks, then an event is triggered. Different functions are registered with an event handler to respond to these triggers, such as in an 'OnClick' subroutine.

In the Observer pattern, these functions are, instead, whole objects which implement a consistent function, described by an interface. The 'subject', that is, the object which first interprets these triggers, holds a collection of objects, which subscribe to this observing interface. When implementing the 'subject' interface, it exposes an ability to add or remove these at runtime; and when it detects a trigger, it notifies all of its 'observers'.

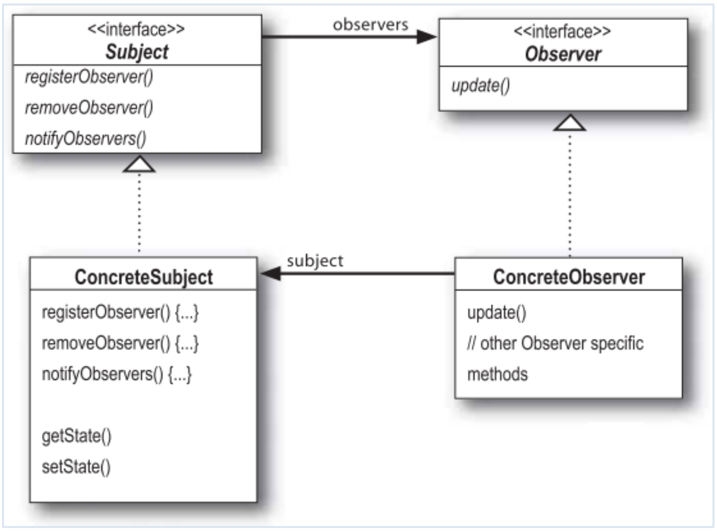


Figure 3.4: The Observer Pattern (adapted from Freeman et al. 2004)

The typical usage of the Observer pattern begins with a prospective observer object registering itself with the subject object; this could be through passing itself as a parameter to the 'register' function (Freeman et al., 2004, p. 57). The subject then adds it to an internal collection of Observer interface objects, as this guarantees that the update function exists. When the subject determines that its components need to be notified, it calls the notify function. This method could then iterate, potentially using an Iterator pattern, calling thereby each observer object's update function.

The way in which the observer objects handle updates is one of mainly two styles (Freeman et al., 2004, p. 63). There is the 'push' style, in which the subject sends state to the observers as a parameter to the update function – and for this, the Observer interface must specify this parameter in the update method declaration. Alternatively, there is the 'pull' style; and this requires the subject to expose data publicly so that the subjects can fetch state selectively.

It was stated in the previous section that the Observer pattern promotes loose coupling. Loosely coupled objects have minimal knowledge of each other's internal structure and implementation. The Observer pattern only relies on implementation of the observing interface. The subject and its observers can work independently from one another, without changes affecting either side (Freeman et al., 2004, p. 53).

Much like the Strategy, the Observer pattern also encourages flexibility and reuse. It has been identified as one of the most useful patterns (Zhang & Budgen, 2012a, p. 7). The following section introduces the MVC pattern. The MVC pattern is a Compound pattern, which makes it ideal for conveying information in stages of its components. This following section discusses the MVC pattern itself and the other patterns in the Compound MVC pattern.

## 3.6. The Model-View-Controller Pattern

In software development, a concept known as N-tier architecture (most commonly 3-tier) is used to separate application logic from business logic, as well as data modelling, and user experience (Kachurka, 2013, p. 1). Similar to this concept, the MVC pattern focuses on separating application design into common developer roles: back-end components, User Interface (UI) design, and interfacing with functionality between these (Kachurka, 2013, p. 2; Smith, 2009, p. 2). The structure of the MVC pattern works to separate the responsibilities of components as well; and as such, it is especially suited to web applications (Syromiatnikov & Weyns, 2014, p. 10).

The View shows the windows, buttons, and other controls to the user; the Controller interprets clicks, and responds to other commands; and the Model does the business logic and object retrieval – then relaying the changes to the View again (Freeman et al., 2004, pp. 536–540). The View can request state information from the Model; and the Controller can ask the View to update its display. This relationship is shown in Figure 3.5.

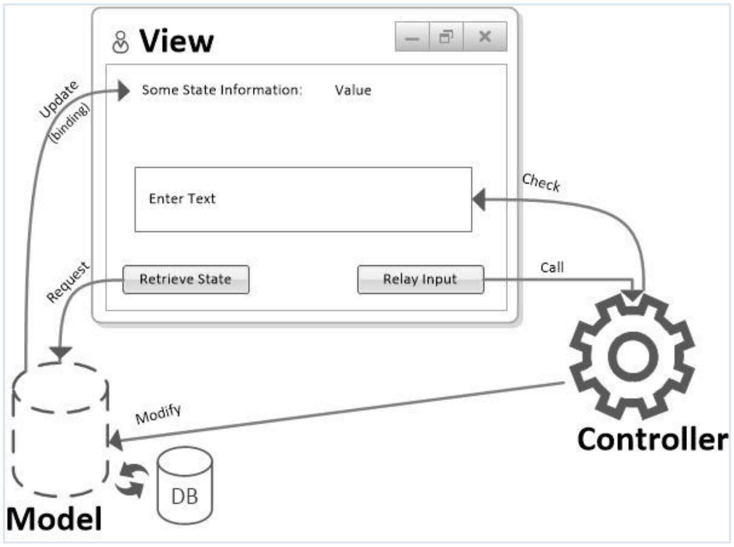


Figure 3.5: The Model-View-Controller (adapted from Freeman et al. 2004)

As mentioned earlier, the MVC pattern features concepts found in other patterns (Freeman et al., 2004, p. 541). In the following sections, these concepts are briefly discussed, as they pertain to the MVC pattern.

### 3.6.1. The Observing View

The View is an observer of the Model. It registers to be updated when the state changes. This registration could be shown, for example, in Figure 3.5 on the button entitled 'Retrieve State'. It could be that when clicked, the View calls the Model's register method and passes itself in as a parameter. However, in reality, this would happen prior to the View being loaded.

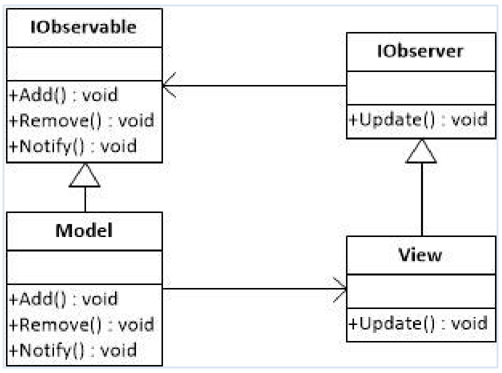


Figure 3.6: The Observer Pattern (adapted from Freeman et al. 2004)

Using Figure 3.6, this can be understood in the following manner. The View can access the Model to add itself as an observer. When something changes, Notify() calls for all observers, in this case the View, to update their state.

### 3.6.2. The Strategic View

The View is also able to make use of a selection of Controllers, which implement its required functions. The ability to swop out these Controllers works in the same way as the Strategy pattern, shown with context in Figure 3.7.

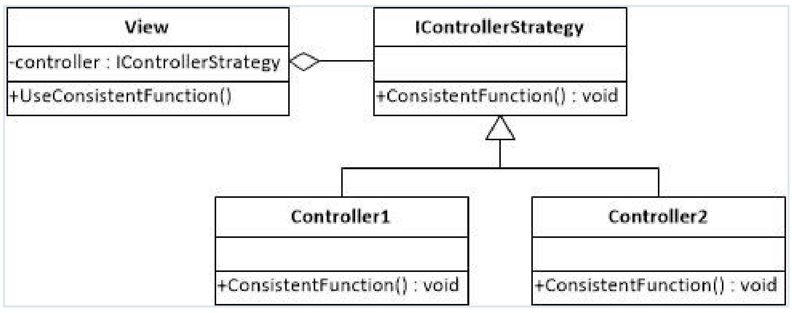

View
-controller : IControllerStrategy
+UseConsistentFunction()
IControllerStrategy
+ConsistentFunction() : void
Controller1
+ConsistentFunction() : void
Controller2
+ConsistentFunction() : void

Figure 3.7: The Strategy Pattern (adapted from Freeman et al. 2004)

This keeps the structure of components consistent, so that they can be used interchangeably (Freeman et al., 2004, p. 24). The View is effectively able to use any Controller – without the developer having to worry about specific implementation.

### 3.6.3. The Composite View

Finally the View is also able to compose of further Views. A control can comprise of a set of other controls – and each of their functions are handled – as if only one control existed. The ability to contain multiple objects, and to act interchangeably as one object or many, is shown in the Composite pattern depicted in relation to the MVC pattern in Figure 3.8.

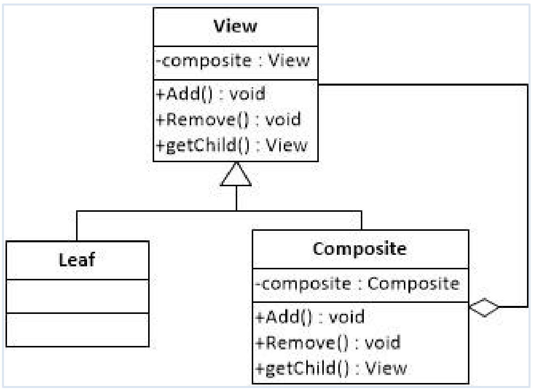


Figure 3.8: The Composite Pattern (adapted from Freeman et al. 2004)

The following section discusses patterns which are derived from and are therefore similar to the MVC pattern.

## 3.7. The Model-View-Presenter Pattern

The MVC pattern is an age-old pattern, which has been revisited a number of times, branching into an MVC family of patterns – each catering to its own environment and priorities (Holmström, 2011, p. 18; Kachurka, 2013, p. 2; Smith, 2009, p. 2). Firstly, the Model-View-Presenter (MVP) modifies the concept of a Controller to that of a Presenter.

The Presenter handles interaction between the Model and the View, updating the View and reacting to user interaction. It also supplies an additional three subsections: 'commands' between Presenter and Model; 'selections' between Presenter and Model; and 'interactors' between Presenter and View (Kachurka, 2013, p. 2).

The MVP could split into two variations; the Passive View, where all View logic must be separately coded; and the Supervising Controller (also known as Supervising Presenter), where the View is bound to the Model via data-binding; and the Controller handles user interaction (Syromiatnikov & Weyns, 2014, p. 8). The disadvantage of the Supervising Controller is the coupling between View and Model – which is less favourable for testing. MVP patterns suffer from reduced modularity, as there is no strict enforcement of 'separation of concerns' (Syromiatnikov & Weyns, 2014, p. 10).

Alternatively, there is the Mediating Controller (or Model-View-Adapter), where there is no communication between View and Model (Kachurka, 2013, p. 2). ‘Adapter’ more adequately describes the way in which both the Presentation Model (a variation less dependent on Controllers) and the Model-View-ViewModel work – providing adapters to the Model, instead of direct access.

## 3.8. The Model-View-ViewModel Pattern

Abstracting the View of the MVC pattern, the Presentation Model was the basis for the Model-View-ViewModel (MVVM). The MVVM directly binds View and ViewModel. The ViewModel encapsulates all View behaviour, and this allows for potentially no Controller. As such, instead of saying that the MVVM is a specialised version of the MVC pattern, it is more correct to say that it is a specialisation of the Presentation Model towards Windows Presentation Foundation (WPF) (Smith, 2009, p. 2).

The MVVM pattern embraces the separation of concerns principle, as its modular View-to-ViewModel nature allows Views and Models to be wholly independent of one another (Syromiatnikov & Weyns, 2014, p. 10). However, due to its extensive use of synchronisation, it can lend itself to hindered performance.

The MVVM is less frequently used within web applications, as it is tailored for WPF (Syromiatnikov & Weyns, 2014, p. 9). Similarly, MVP is tailored to forms. However, the standard MVC pattern is widely used in web development, especially for ASP.NET.

## 3.9. Conclusion

This chapter discusses what software design patterns are; whether they are relevant and effective; as well as how they might be recorded and implemented. Their origin is briefly examined; and arguments for potential advantages are highlighted. Whether a pattern makes a solution ‘better’, depends on whether the solution would benefit rather than suffer from trade-offs.

In general, solutions hold value in maintainability, extensibility, flexibility and reusability. In smaller applications with no desire to evolve, it might not be worth consideration; although in larger solutions, these aspects save vast amounts of time, money, and other resources. In fact, patterns used unnecessarily could reduce the effectiveness of a solution. This is especially true if that solution is not geared towards the benefits that a pattern might provide, such as in smaller applications.

For larger solutions, if the use of a design pattern does not contribute to these aspects, then it has not been implemented correctly. However, on the other hand, after weeding out incorrect implementations, developers should be able to learn from their mistakes, and further their own knowledge schema. As expected, more experienced developers should know when a pattern could benefit an application – and when not to use them.

A variety of patterns pertaining to the MVC pattern are discussed, showing how they work and what category they fit into, as well as how they are conveyed through class diagrams. On the other hand, the MVC pattern uses a more visually oriented diagram. This, and its contained patterns, are discussed, as well as two alternative variations of the MVC pattern: the Model-View-Presenter and the Model-View-ViewModel.

The next chapter describes how the security-conscious MVC (SecMVC) model is composed; and how it has evolved from incorporating derived security mechanism concepts to conveying both principles and concepts in the secure software design context.

# Chapter 4: The Security-conscious MVC Model

## 4.1. Introduction

In the previous chapters, this research study discussed secure software development in general. This led into secure software design principles and concepts in Chapter 2, as well as software design patterns in Chapter 3. Additionally, it argued the significance of the Model-View-Controller (MVC) pattern as a potential medium for conveying these principles and concepts. As security is not inherent in the MVC pattern itself, this chapter seeks to integrate secure software design principles and concepts into it.

The purpose of this chapter is to introduce and discuss the security-conscious MVC (SecMVC) model; this is an amalgamation of the MVC pattern and secure software design principles, as well as their underlying concepts. Before introducing the final revision of the SecMVC model, however, this chapter explains the reasoning behind its composition; and it demonstrates how the SecMVC model was achieved through its various iterations. The next section describes the general idea with regard to the model's foundation.

## 4.2. Foundation of the SecMVC Model

The proposed SecMVC model is based first and foremost on the visual depiction of the MVC pattern, which shows the Model, View, and Controller in a triangular relational diagram. This is shown in Chapter 3, Figure 3.5, as well as in Figure 4.1 for convenience.

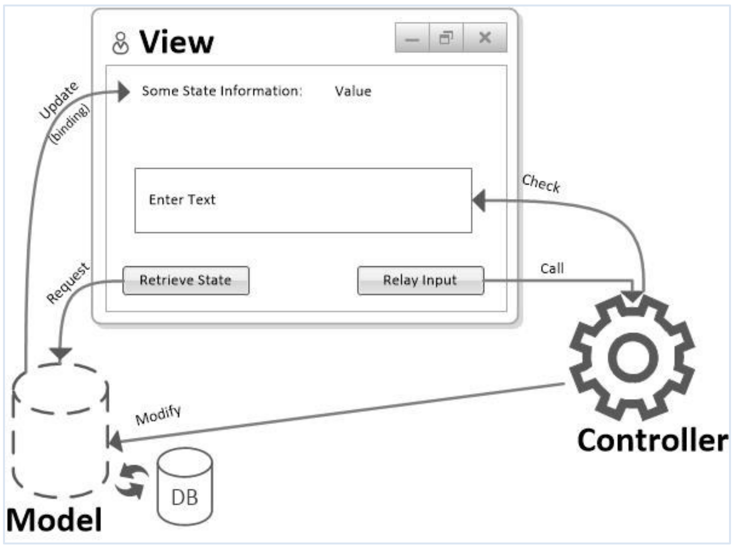


Figure 4.1: The Model-View-Controller (adapted from Freeman et al. 2004)

When presented as an image for visual aid, the Model is typically illustrated as a cylinder, much like a flowchart database graphic; while the View is often shown as the window of an application. The more abstract Controller, on the other hand, can be depicted as anything from a simple shape to a group of cogs.

The SecMVC model is the product of the MVC pattern, and a variety of identified secure software design principles and concepts. These two focus areas are delineated from the broader fields of secure software development and software design patterns. The way in which the SecMVC model relates to these is presented in Figure 4.2.

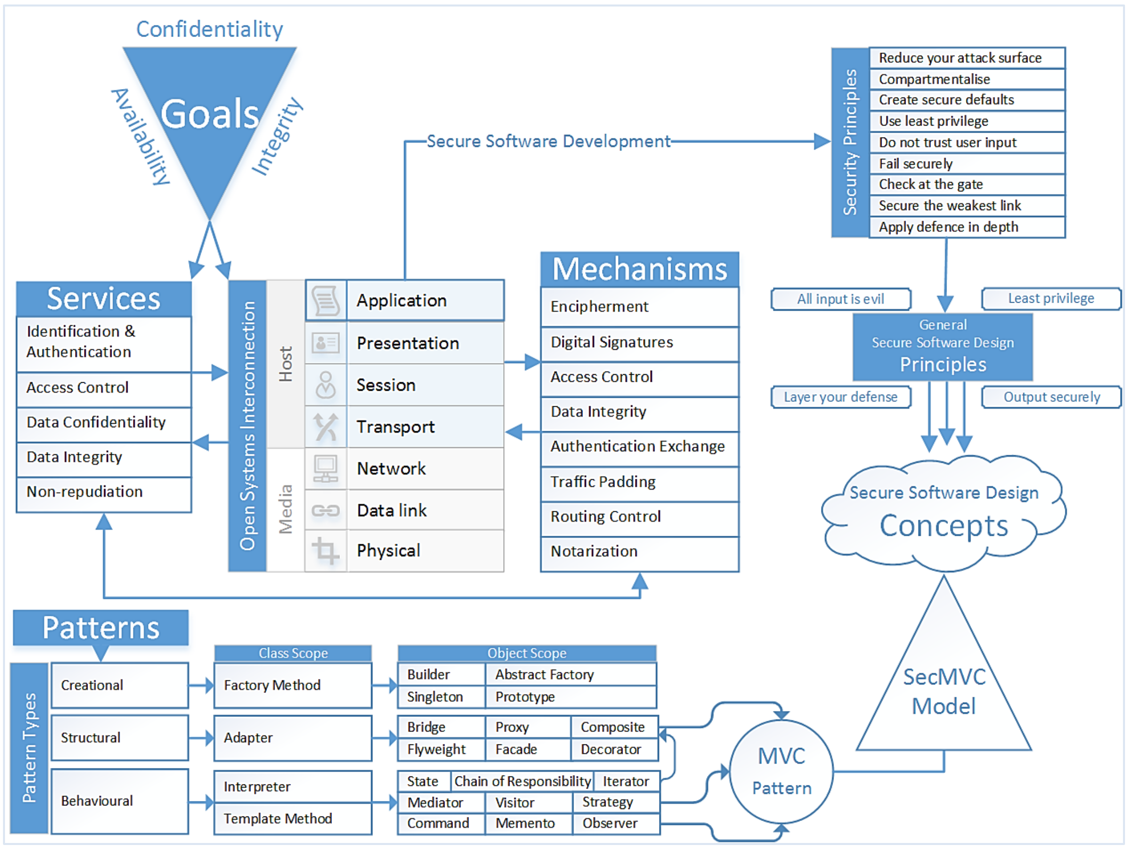


Figure 4.2: SecMVC Model in this Research Study

The SecMVC model does not, however, end with only one revision; and as such, it went through numerous iterations. The primary difference between the initial and current versions is the use of Open Systems Interconnection (OSI) security mechanisms, as opposed to secure software design principles to group secure software design concepts. The comparison is shown in Table 4.1.

Table 4.1: SecMVC Model Components

| **OSI Security Mechanism Based Concepts** | |
|---|---|
| Access Control | Access Control; Trust; Guard all entrances |
| Authentication Exchange | Authentication; Authorisation |
| Encipherment | Hash stored passwords; Encrypt sensitive data |
| Notarisation, integrity, signage | Logging; Integrity |
| 'All input is evil' | Limit input; Refuse bad data; Whitelist instead of blacklist; Verify/Validate cycle; Encoding |
| **Secure Software Design Principle Based Concepts** | |
| *Least Privilege* | Limit; Isolate; Deny; Wrap; Timeouts; Logging |
| *All Input Is Evil* | Suspect; Limit; Reject; Scrutinise; Encrypt; Sanitise; Verify/Validate cycle |
| *Output Securely* | Centralise; Encrypt/Cross-reference/Omit; Notify; Timeouts; Logging; |
| *Layer Your Defence* | Authentication; Authorisation; Trust; IPS; SSL; CAS |

While there is some overlap between the mentioned summarised secure software design concepts, they each exist within specific secure software design principles. These concepts, and the revisions through which they are presented, are described in the next section.

## 4.3. Revisions of the SecMVC Model

Initially, the SecMVC model was designed to incorporate secure software design concepts, using the security mechanisms derived from the OSI reference model's security services (ISO/IEC 7498-2, 1989). As such, the SecMVC model was first approached with these in mind. Specifically, secure software design concepts were introduced to the model: Firstly with access control in mind, then authentication exchange, encipherment, notarisation, data integrity, and digital signage, respectively. Accompanying these were a host of input related secure software design concepts under the title: *All input is evil*, which later became the secure software design principle of the same name.

From access control, the notion of authorisation, authentication, access rights, privileges and trust are of particular importance. Even in managed frameworks, a substantial amount of the enhanced security, specifically in the .NET framework, is based on trust. *Least privilege*, a general security principle, encourages providing an entity with as few rights as possible, as well as for the shortest duration possible. Figure 4.3 shows the beginnings of the SecMVC model with access control concepts featured.

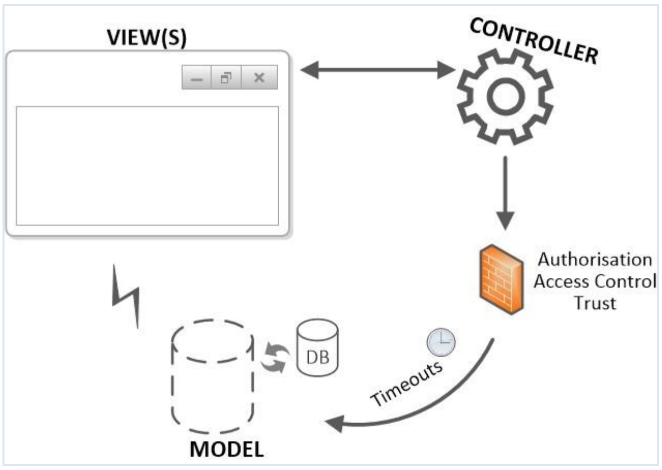


Figure 4.3: SecMVC Model with Access Control Concepts

In addition to authorisation, as featured from access control, there is authentication. Both of these are commonly overlooked defensive measures. The authentication exchange related risks include only guarding the 'front door', hard-coded credentials, and the tendency to implement custom security measures, when the provided known solutions are most effective. In order to cater for these authentication exchange concepts, the model was further incremented, as shown in Figure 4.4.

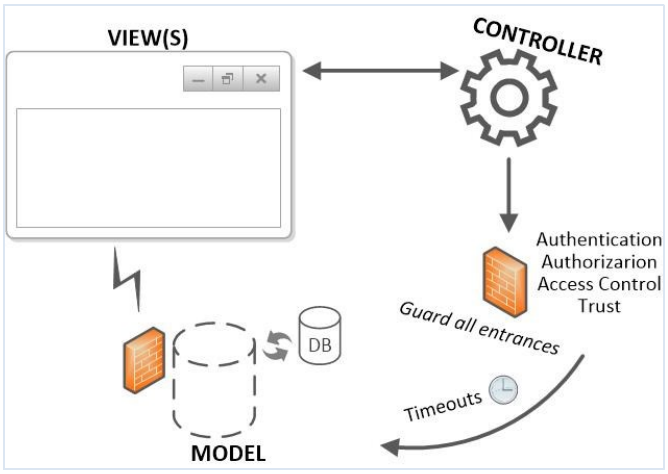


Figure 4.4: SecMVC Model with Authentication Exchange Concepts

The security mechanism 'encipherment' introduces the concepts of encryption, as well as the risk of using an ineffective cryptography algorithm. The data held within the Model should be kept secure, using one-way ciphers, such as hashes, where values do not need to be decrypted, as for authentication. This, together with all other business logic, should be kept confidential. These encipherment concepts are reflected, along with previous secure software design concepts, in Figure 4.5.

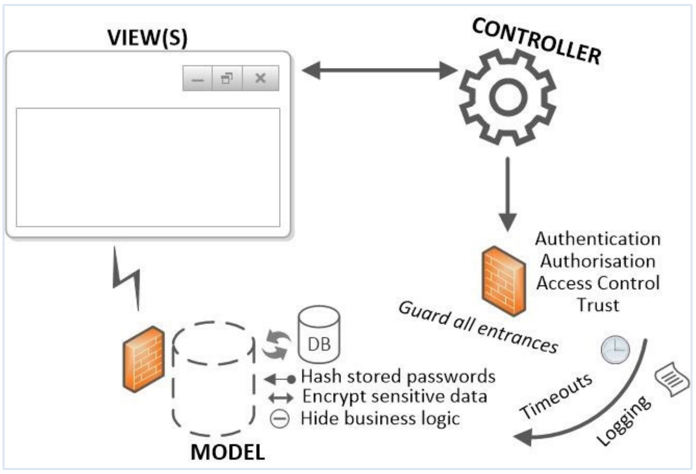


Figure 4.5: SecMVC Model with Encipherment Concepts

With regard to the use of a notary to store evidence of a transaction, as well as digital signatures as a form of non-repudiation, identifying elements of these are addressed in verbose logging. The recording of actions taken, and the use of signage to verify claims, is vital, in ascertaining ownership of those actions. Specifically, access to the Model from all points of entry should be monitored. Conversely, unnecessary details might slow and overload writing operations and storage.

Integrity, as a key security service, is enforced by comparing checksums or hashes, before and after transfer. Ensuring data integrity in the Model, as well as in physical storage, comes down to encryption. However, in terms of poor software security, the integrity mechanism can also relate to injections and overflows. These are first addressed through the user interface. As identified in the Common Weakness Enumeration (CWE) by Bob Martin et al. (2011, pp. 3–14), the top four vulnerabilities are input related. This leads to the notion that *all input is evil*.

From escaping, encoding and properly quoting characters to the use of validation and verification of input, the interaction between a user and the system should be developed under the assumption that the user is malicious. This mind-set extends to all aspects surrounding input, including closed responses, such as radio buttons, as opposed to open text fields, limited input length, and restricted character whitelists.

All tools available, which could restrict dangerous input, should be used, comprising all available points of entry – and not just the user interface. This, therefore, extends to data access as well, such as in the use of prepared statements and stored procedures, as opposed to dynamically constructed queries. The use of these input related precautions is included with the previously mentioned security aspects in the first version of the SecMVC model. This is shown in Figure 4.6.

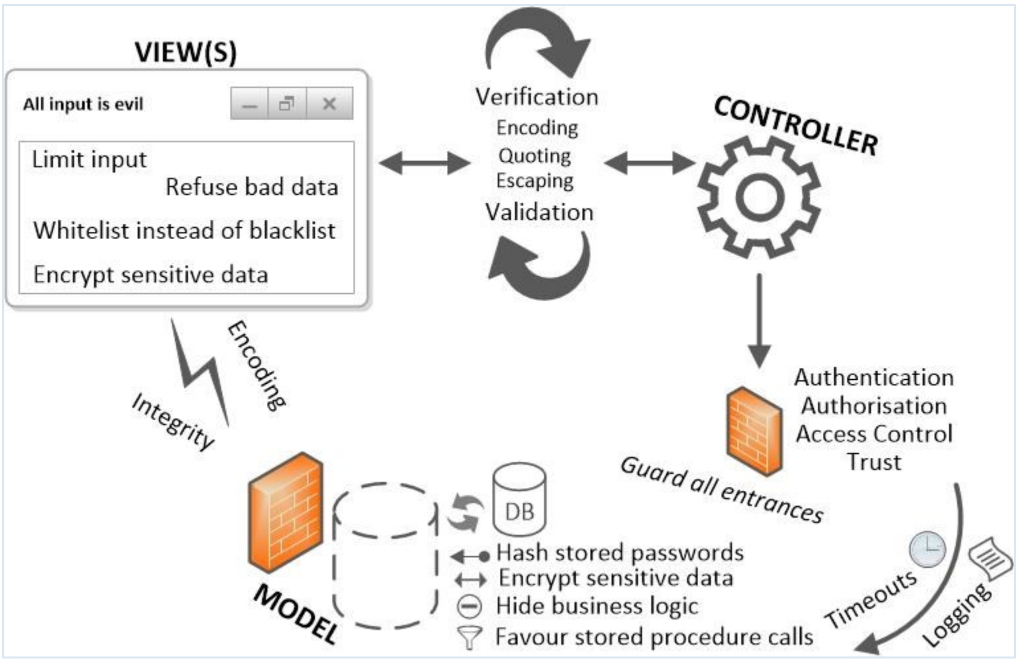


Figure 4.6: SecMVC Model Version 1

The first version of the security-conscious MVC model served, as the output for a peer reviewed paper in the 2013 South African World Wide Web (ZAWWW) Conference. This paper is presented in Appendix A. This initial revision served as a basis for an improved model, which was approached with generalised secure software design principles in mind. The second revision, SecMVC model Version 2, is shown in Figure 4.7.

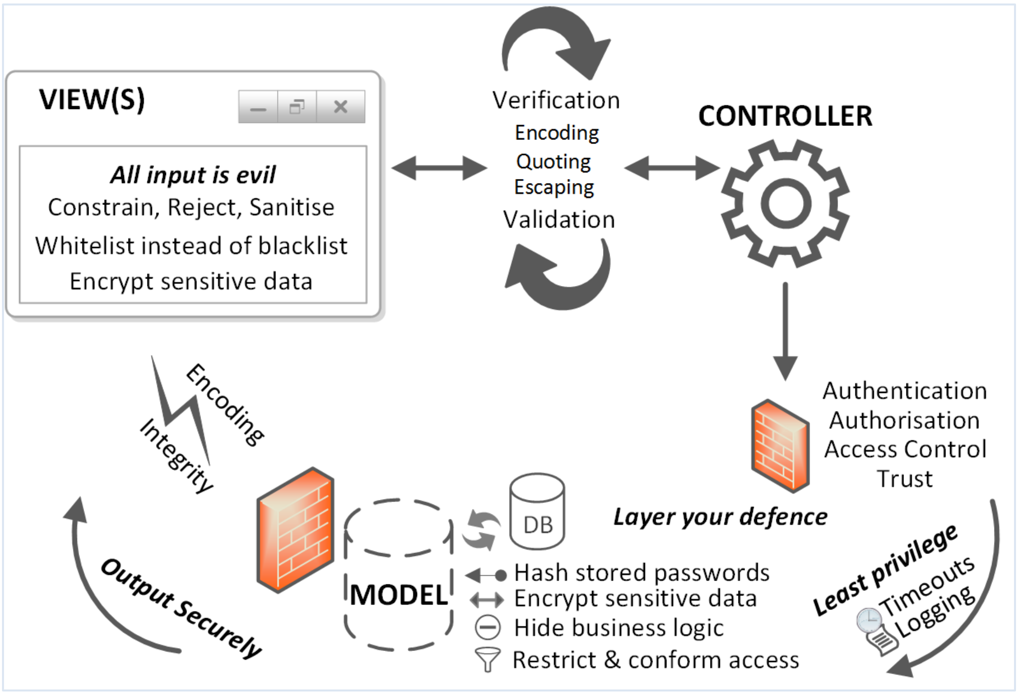


Figure 4.7: SecMVC Model Version 2

When compared with the original approach, which considered security concerns through security mechanisms, the revised approach includes secure software design principles, which each represent a number of more detailed secure software design concepts. These four secure software design principles: *All input is evil;*, *least privilege; layer your defence;* and *output securely*, were derived from the original nine security principles of Meier et al. (2003, p. 11) in Chapter 2, Section 2.7. Many of these concepts are shared with those identified from the mechanism perspective.

The ease in relating these two approaches, that is, security mechanisms and secure software design principles, reinforces the relevance of their shared resulting concepts. However, the revision of the SecMVC model did not adequately illustrate the importance of the general secure software design principles, as opposed to merely the relating concepts.

In order to better differentiate between secure software design principles, secure software design concepts and the MVC pattern itself, the third revision of the SecMVC model uses colour consistently. This third SecMVC version is shown in Figure 4.8. The elements are better spaced; and the model uses better symmetry, to help show the relative nature of the pattern. These elements surround the notion of layered defence.

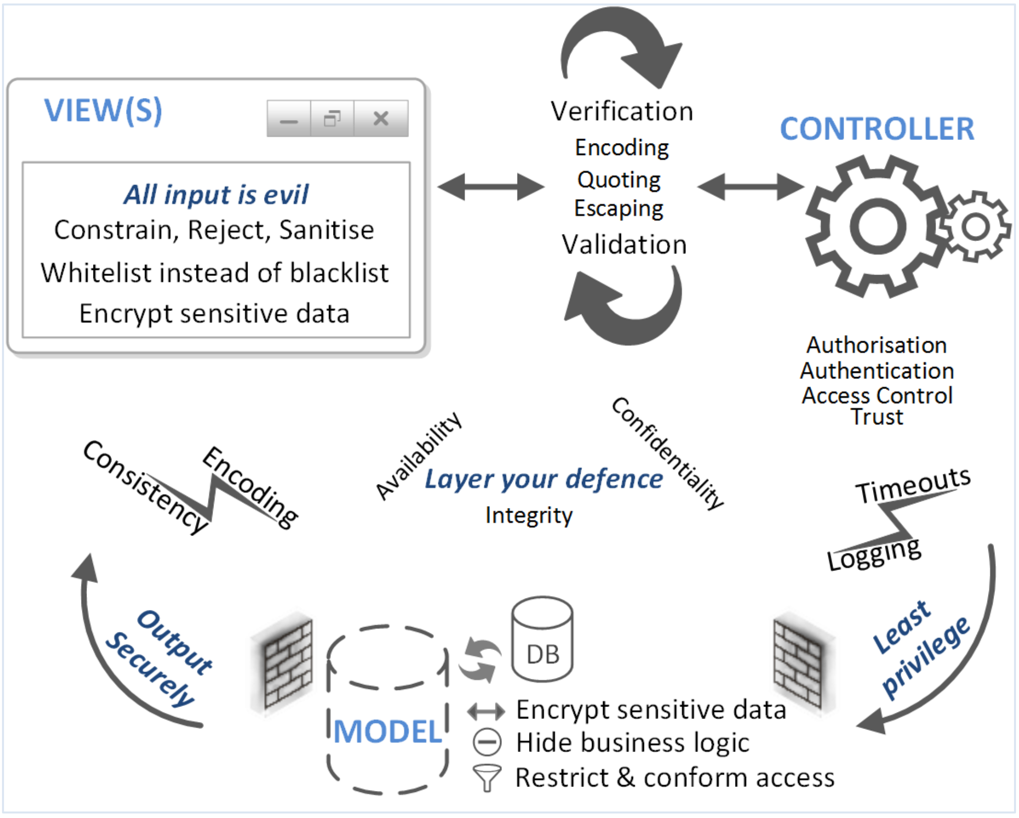


**Figure 4.8: SecMVC Model Version 3**

With the purpose of explicitly including more prominent secure software design concepts, which were identified in the literature, the fourth version of the SecMVC model is shown in Figure 4.9. Maintaining the colour scheme, the fourth revision reassigns the *layer your defence* principle's depiction to the authorisation, authentication and trust elements, which fall more into line with the principle. Added to these, are Code Access Security (CAS), Secure Sockets Layer (SSL), and Internet Protocol Security (IPS).

Encryption and sanitisation are moved to between the View and Controller; while the View's secure software design concepts are constrained to the simplified terms 'suspect', 'limit', and 'reject', as they pertain to input. Added to the View are cautions to the use of delegates and dynamic loading, as comprising less obvious types of input. Timeouts and logging are shown between the View and the Model – for further consistency.

The Controller element in the fourth revision is shown to include the simplified secure software design concepts: 'isolate', 'wrap', and 'deny'. Like the View, it also mentions 'limit'. These are shown with regard to assembly access.

The Model element is also modified in the fourth revision. With reference to data access, the Model shows the simplified secure software design concepts: 'notify', 'centralise', and the need for: encryption, omission, or cross-referencing of sensitive information.

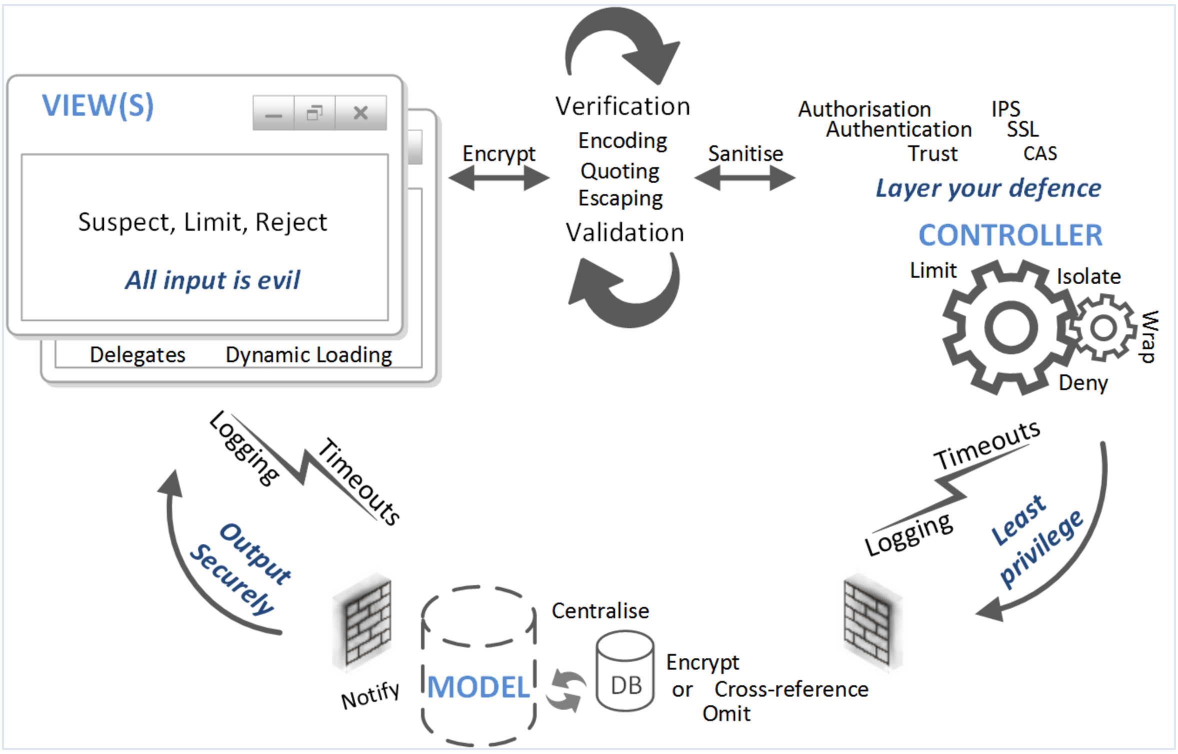


Figure 4.9: SecMVC Model Version 4

The SecMVC model Version 4 attempts to convey the most prominent of secure software design concepts under the four generalised secure software design principles. For the *least privilege* secure software design concepts listed in Chapter 2, Table 2.2, the term 'limit' is shown in both the Controller and View(s) of the model. This points to limiting rights, access, functionality, permissions, time, and the ability to provide evidence, as well as the anonymity of that evidence. It also includes the limitation of retention and readability for sensitive information.

Further reference to access restriction is shown in terms above the Controller, including CAS. For this, the 'isolate' and 'wrap' terms serve as hints towards the need to isolate dangerous functions, requiring special security permissions, such as with unmanaged code, as well as those which utilise the 'Demand/Assert' pattern.

From Chapter 2, Table 2.3, showing *all input is evil* secure software design concepts, the notion to 'suspect' all forms of input is shown in the View(s) of the SecMVC model. Again, limitation is prominent, favouring whitelists and closed-ended inputs, as well as restricting length and rejecting questionable input. Encryption and sanitisation are shown in the SecMVC model surrounding the verification and validation process, while examples of technology-specific assistance are shown above the Controller with SSL and IPS.

Canonicalisation is hinted to by output 'encoding', as well as with 'escaping'. Furthermore, non-standard input is given caution to through a behind-the-scenes View, which specifically warns of delegates and dynamic loading, such as serialisation.

In *output securely* secure software design concepts, shown in Chapter 2, Table 2.4, 'encryption', 'cross-reference' and 'omission' are shown as options for sensitive information by the Model part of the SecMVC model. *Least privilege* again ties into this, as shown to the right of the Model. 'Centralisation' is shown to reinforce the notion of managed logging and exception handling; while 'logging' is shown to occur on either side of the Model's interactions. Notable occurrences should 'notify' users with adequately helpful information, and limit sensitive information to those with the necessary permission to view it.

For Chapter 2, Table 2.5, *layer your defence* secure software design concepts, 'authentication' and 'authorisation' are shown beside their containing principle, along with assisting technologies. Here, 'trust' is shown to govern access; and it demonstrates a preference for strong-naming; while the notion to 'deny' by default is shown alongside the Controller. Again, restrictions are shown in the term 'limit'; while further logging, timeouts and encryption occur between and within the Model.

## 4.4. SecMVC Model in Practice

In order to discuss the potential effectiveness of the SecMVC model in practice, some meaningful examples have been used. These examples include three of the most notable security risks, according to the Open Web Application Security Project (OWASP), and the Common Weakness Enumeration (CWE).

The most recent effort to date by the OWASP to rank prevalent security risks is the OWASP Top 10 of 2013 (Williams & Wichers, 2013, p. 6), which maintained 'injection', 'broken authentication and session management', and 'cross-site scripting' as the top three security risks. This is compared to the CWE (Martin et al., 2011, pp. 3–4), which ranked 'injection' weaknesses at first and second, the classic 'buffer overflow' at third, 'cross-site scripting' fourth, and 'missing authentication' at fifth. The two rankings reinforce each other's conclusions.

In the delineation for this research, it is specified that a managed framework would be assumed, when assessing security aspects. The chosen managed framework on which this research relies is Microsoft .NET. Therefore, classic 'buffer overflow' attacks are mitigated by internal checks. This sharpens the focus in terms of top weaknesses, or risks, to the following:

1. Structured Query Language (SQL), Operating System (OS), and Lightweight Directory Access Protocol (LDAP) Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)

A developer, who takes the SecMVC model into consideration, with an understanding of its elements and their relation to the underlying secure software design concepts and principles, should be less likely to create a solution with these weaknesses. The way in which such a developer would take note of these is shown in the following cases.

### 4.4.1. SQL, OS, and LDAP Injection

The OWASP Top 10 (Williams & Wichers, 2013, p. 7) has the following derived suggestions regarding injection:

- Consider all possible users and administrators as potential input for untrusted data;
- Almost any source of input can be an injection, including unconventional or internal inputs;
- Injection flaws come from interpreting untrusted data – separate this from interpreters;
- Parameterise the input, and escape any special characters carefully; and
- Whitelist input rather than blacklist, providing closed-ended options.

Noticeably, these suggestions coincide with the secure software design concepts outlined in the SecMVC model's four general secure software design principles. The model encourages considering all input as potentially malicious, even from unexpected sources, such as internally through delegates and dynamic loading. This is seen in the View(s) section of the model.

The Controller element encourages isolation, requiring trust before manipulating input. Between these, the verification and validation process surrounds the need for adequate encoding, quoting and escaping. This canonicalisation exists for file inputs as well, which are also a noted CWE risk (Martin et al., 2011, pp. 3–4).

The CWE provides similar suggestions for SQL Injection in particular (Martin et al., 2011, pp. 7–8). It suggests the use of 'prepared statements, parameterised queries, or stored procedures' along with the use of 'strong typing'. It advocates the separation of data and code, much like OWASP does between untrusted data and interpreters. Whitelisting, along with limited blacklisting, is deemed useful, while 'encoding, escaping, and quoting' is described as the most effective solution. It also suggests lowest privilege, which is another of the SecMVC model's secure software design principles, as well as the use of isolation of privileges per specific functions.

The CWE's suggestions for OS Injection include 'sandboxing', isolating processes from one another and the operating system (Martin et al., 2011, p. 9). Managed code, such as that in the .NET framework, is also noted to provide protection. Again, the use of escaping is shown to mitigate some of the risk involved.

### 4.4.2. Broken Authentication and Session Management

In the CWE's suggestions for Missing Authentication for Critical Function, specific mention is given to only guarding the 'front door' (Martin et al., 2011, p. 14). It suggests using server-side checks, even for checks already done on the client-side. Again, the isolation of different areas of access within an application are recommended, and the centralisation of authentication procedures is encouraged. The use of tried and tested technology is shown to be important, with note given to the dangers of custom implementation.

OWASP provides a 'cheat sheet' with regard to session management (Siles, 2014). In this, advice is given regarding factors which developers can consider and influence. A non-exhaustive list, which summarises this advice is shown in the following list:

- Session ID names should be generic, not describing their technology as per default;
- Session IDs should be long enough to prevent brute force attacks;
- Session IDs must be unpredictably random, revealing no meaningful information;
- Session IDs must be strict, only originating from server generation, and are untrusted input;
- Session IDs must be renewed when the level of privilege changes, and after authentication;
- Sessions should have an idle and absolute timeout, and may use a renewal timeout;
- Sessions should allow users to manually terminate their sessions;
- Cookies are the preferred session mechanism, as they allow advanced token properties;
- The accepted session mechanisms should be limited to those chosen by design;
- Avoid custom implementations. Note that default settings might also not be sufficient; and
- Both the session mechanism and the web session should make use of secure technologies.

Considering these suggestions, a developer adhering to the SecMVC model would already avoid ‘only guarding the front door’. Mindful of all forms of input as being potentially malicious, the developer would not expect client-side checked input to be trusted data. The developer would ‘isolate’ different areas of access, and still centralise the implementation of error handling, logging, and authentication. Known solutions using secure technologies would further promote the security of the application.

This developer would apply the secure software design principles of *least privilege* and *all input is evil*, as well as their secure software design concepts to session management. With limited representation of sensitive information taken into account, as well as isolation and a need for trust, the construction of both session IDs and session ID names would closely follow the suggestions outlined in the OWASP ‘cheat sheet’.

As the developer would prefer a higher degree of control over the session mechanism, if possible, session cookies could also be used. When privilege levels change, including post authentication, the developer should renew the cookie. This also provides a means against XSS attacks. In the context of the .NET framework, ASP.NET function calls could be used. This includes the “Session.Abandon()” and “Response.Cookies.Add()” methods.

In these session cookies, the developer would use ‘Secure’ and ‘HTTPOnly’ attributes to prevent Man-in-the-Middle and XSS attacks, respectively. ‘Secure’ ensures that cookies are only transported over secure channels, and ‘HTTPOnly’ prevents client-side scripting from retrieving the cookie. This is due to the need for SSL usage, as well as a strong mistrust for client-side script in terms of the session cookie, which is a sensitive form of information.

The developer should also be restrictive on the ‘Domain’ and ‘Path’ attributes of the cookie, using the default domain as the origin server (not setting the attribute). This adheres to the notion to ‘limit’ where the cookie may be sent, as well as to ‘isolate’ session IDs from other entities on the host.

In terms of 'timeouts', the 'Expire' or 'Max-Age' attributes would not be set, so as to avoid persistent cookies, when security is a higher concern than convenience. However, session idle and absolute timeouts would also be set, which would prevent still open browser windows from continuing the session over dangerous periods of time. Should an attacker hijack the session ID, a renewal timeout implementation could additionally limit the attacker's time to cause damage. The OWASP 'cheat sheet' recommends 2 to 5 minutes idle timeout for secure applications.

### 4.4.3. Cross-Site Scripting

The most prominent form of prevention against cross-site scripting, according to the OWASP Top 10 (Williams & Wichers, 2013, pp. 9–10), is the 'separation of untrusted data from active browser content'. That is, once again, the isolation of untrusted data from interpretation. Another repeated suggestion is the consideration of all forms of input for their potential to cause damage. Further encouragement is given to escaping, based on Hypertext Markup Language (HTML) context. The same applies to whitelist input validation, although it is again noted that the practice does not guarantee protection. Additional considerations are listed below:

- All untrusted input must be escaped, validated and verified as safe before processing;
- If using Asynchronous JavaScript and XML (AJAX), then safe client-side scripting Application Programming Interfaces (API) are recommended;
- If unsafe APIs are in use, strict validation and encoding are recommended; and
- While automated tools can assist detection, favour code review, and penetration testing.

OWASP also provides a 'cheat sheet' for the prevention of cross-site scripting (Williams, Manico, & Mattatall, 2014). In this, various known XSS attack vectors, specifications and manual testing have been used to provide rules for the prevention of XSS attacks. These rules are summarised in the following list:

- In any web page, untrusted input must be placed in specific, isolated and consistent slots;
- The escaped syntax must be used for non-alphanumeric characters in any of these slots;
- HTML escape untrusted data in content, specifically for: &, <, >, ", ', and /;
- HTML escape untrusted data in JavaScript Object Notation (JSON) values;
- Attribute escape untrusted data in common HTML attributes;
- Script escape untrusted data values in script if needed, untrusted data here is dangerous;
- CSS escape and strictly validate untrusted data before using it in style properties;
- URL escape untrusted data in query strings;
- Consider the use of a Content Security Policy; and
- As with session management, use specialised sanitisation libraries and HTTPOnly in cookies.

In the .NET context, Microsoft provides the AntiXSS library (Dorrans, 2014). In ASP.NET specifically, there also exists the "ValidateRequest()" method, which assists in sanitising requests for XSS (Williams et al., 2014). As the developer would be suspicious of all forms of input, these features or similar ones would be used to verify the safety of the data.

One of the ways in which the developer would limit the risks of XSS is through proper organisation of untrusted data, avoiding the use of such data in the areas depicted in Figure 4.10. Notably, contained script elements, such as those in 'onClick' attributes and the like, should avoid the use of untrusted data as well.

```
<script>HERE</script>  --clientscript
<!--HERE-->            --comments
<div HERE="e.g."/>     --attribute name
<HERE href="e.g."/>    --tag name
<style>HERE</style>    --stylesheets
```

Figure 4.10: Where Not To Put Untrusted Data (adapted from Williams et al., 2014)

As the SecMVC model advocates the use of encoding, quoting, and escaping, the developer would take steps to canonicalise all forms of input. Different elements have different encoding requirements, however. The different elements which can hold untrusted data (as such, being at risk of XSS attacks) and their escaping strategies are shown in Table 4.2.

Table 4.2: Escape Strategies (adapted from Williams et al., 2014)

| HTML Escaping Conversions | & | < | > | " | ' | / |
|---|---|---|---|---|---|---|
| | &amp; | &lt; | &gt; | &quot; | &#x27; | &#x2F; |
| Attribute Escaping | All non-alphanumeric characters with HTML &#xHH; format | | | | | |
| URL Escaping | Standard percent encoding for parameter values in %HH format | | | | | |
| Script Escaping | All non-alphanumeric characters with Unicode \uXXXX format | | | | | |
| CSS Escaping | All non-alphanumeric characters with \XX or \XXXXXX format | | | | | |

The SecMVC model discussed in this chapter, as well as its potential implementation, have been explored. The model is discussed in terms of its validation in Chapter 6. Through this, the model's usefulness, quality and effectiveness are ascertained. The following section concludes this chapter.

## 4.5. Conclusion

This chapter presents and discusses the SecMVC model, the research output and design science artefact for this research study. This is where the software design pattern and secure software development areas meet, in order to provide a model for conveying secure software design principles and concepts.

The foundations and revisions of the SecMVC model are discussed in this chapter, showing how the model started with security mechanisms to organise concepts, and then moved to secure software design principles and concepts. The explanation of the model, its composition, and its elements are also described.

After the SecMVC model’s introduction and discussion, further discussion follows. This presents prominent security risk examples, and the ways in which the SecMVC could be used to turn its generalised secure software design principles and concepts into specific implementations. In Chapter 5, Section 5.8, the SecMVC is extended further, providing greater capabilities for this.

The next chapter provides a validation for the SecMVC model. As suggested in feedback to be discussed in the chapter, the fourth version of the model (Figure 4.9) is amended by way of three sub-models, in order to provide greater detail. These sub-models contribute to the SecMVC model’s resultant value as a research output.

# Chapter 5: Validation of the SecMVC Model

## 5.1. Introduction

The previous chapter introduced the proposed SecMVC model as the output artefact for this research study. Due to the use of design science, which requires validation of the artefact, the SecMVC model is validated to determine whether it is fit to use as a research output. The purpose of this chapter is to discuss the method of validation, namely, the expert review, and the process followed to complete the expert review. This is done in Sections 5.2 and 5.3, respectively.

Section 5.4 introduces the potential expert reviewers; while Sections 5.5 and 5.6 discuss the interpretation of the expert review feedback received by the deadline. Section 5.7 presents a consolidated analysis of the expert reviews; and Section 5.8 presents amendments to the SecMVC model, based on the feedback provided.

## 5.2. The Expert Review Method

In the Introduction Chapter, the expert review method, as understood by this research study, was briefly introduced, as it relates to other similar research methods. This section discusses these methods in detail, in order to explain this research study's usage of the expert review method. The research methods mentioned include:

- Focus groups;
- Delphi studies;
- Elite interviews;
- Expert interviews;
- Expert panels; and
- Expert surveys.

According to Bhattacherjee (2012, p. 40), **focus group** research is a discussion between subjects of a small sample size (6-10 respondents), which is moderated by a trained individual, who facilitates the joint participation. Bhattacherjee (2012, p. 78) considers focus groups to be a kind of group interview survey. The other types include: 'face-to-face' and 'telephone' interviews. Focus groups are a qualitative research method. However, other methods make use of quantitative processes as well.

The Delphi method, also known as a **Delphi study**, or a Delphi panel, uses both qualitative and quantitative processes, in order to retrieve insights from selected experts (Bourgeois, Pugmire, Stevenson, N. Swanson, & B. Swanson, 1948, pp. 1–2). This occurs repeatedly over an extended time frame, with a small sample per panel (10-18 respondents), usually to determine future trends. This method does not require face-to-face interaction; and it can be administered through correspondence. The premise is that experts do not need to be representative of the population. Being experts, their insights yield more accurate results. This is known as expert sampling.

Bhattacherjee (2012, p. 69) describes expert sampling as the non-random selection of subjects who possess expertise on the examined phenomenon. This is helpful because expert insights are considered to be more credible. As an example, Bhattacherjee hypothesises corporate accountants familiar with the Sarbanes-Oxley Act being used as experts, to understand the impacts of the Act. This suggests that the subject need only be informed, in order to be considered an expert.

Sometimes, the term 'elite' is used interchangeably with the term 'expert' (Posthumus & von Solms, 2009, p. 119). Despite the interchangeable usage, an elite sample and an expert sample can differ significantly (Littig, 2008, p. 13). The elite are those who possess knowledge, influence and are prominent in their field (Littig, 2008, p. 3; Posthumus & von Solms, 2009, p. 119). Experts, however, only need to be knowledgeable in the examined field (Bhattacherjee, 2012, p. 69). While an elite sample may be a kind of expert sample, an expert would not necessarily be considered elite. The same can be said for elite and expert interviews.

An **elite interview** is a qualitative method, where an interview survey is conducted and recorded (Bhattacherjee, 2012, pp. 40, 69; Posthumus & von Solms, 2009, p. 119). This process takes place between the interviewer and one of a small sample of experts at a time. As it is an elite interview, these are experts who are considered to comprise the elite. Some variations of this method provide for the usage of background content by the elite expert prior to the interview, as well as for the interview to occur remotely through correspondence (Posthumus & von Solms, 2009, p. 121). Furthermore, this method may be used to acquire clarification on the research value of an artefact (Posthumus & von Solms, 2009, p. 120).

An **expert interview** is a less specific variation of an elite interview, instead merely using those who are familiar with the subject matter, such as potential users (Littig, 2008, p. 13; vom Brocke & Buddendick, 2006, p. 595). Notably, acquiring insight from potential users is useful for the evaluation of a design science artefact (vom Brocke & Buddendick, 2006, p. 594). Both expert and elite interviews, as opposed to the focus groups, focus on individual responses. On the other hand, expert panels are typically performed in a forum with regular meetings (Slocum, Steyaert, & Lisoir, 2005, pp. 125–126).

An **expert panel** makes use of a preselected sample of experts in a variety of relevant fields. These experts meet and discuss written reports on the subject matter, eventually reaching an agreed upon conclusion (Slocum et al., 2005, p. 121). Similar to the Delphi study, which can also sometimes be called a Delphi panel, this method can be completed remotely (Khodyakov, Hempel, Rubenstein, Shekelle, Foy, Salem-Schatz, O'Neill, Danz, & Dalal, 2011, p. 1). Another method similar to expert review, the expert survey, can typically be completed remotely as well.

An **expert survey** is a broader research method than the previously discussed methods, referring to a survey, which uses a preselected sample of experts in particular. This research method is quantitative. Zhang and Budgen (2012a, p. 2) utilise this research method in their survey of 'experienced user perceptions about software design patterns', referring to their respondents as experts – due to their familiarity with the subject matter.

The discussed research methods, along with this research study's understanding of an expert review, are summarised with their similarities and differences in Table 5.1. In this table, the typical sample size, the level of interaction between the participants, the selectiveness of the sample, and the extent to which the findings may be generalised, are given low to high ratings, respectively. The type of research method is indicated in the 'Measure' column. Finally, the elements in each column, which match those of the expert review, are highlighted. The similarity between these research methods and the expert review is thereafter discussed.

Table 5.1: Summary of Research Method Similarities

| | **Sample** | **Interaction** | **Selectiveness** | **Generalisation** | **Measure** |
|---|---|---|---|---|---|
| Focus Group | Low | High | Low | Medium | Qualitative |
| Delphi Study | Medium | Medium | Medium | Medium | Both |
| Elite Interview | Low | Low | High | Low | Qualitative |
| Expert Interview | Low | Low | Medium | Low | Qualitative |
| Expert Panel | Medium | Medium | Medium | Medium | Qualitative |
| Expert Survey | High | Low | Medium | High | Quantitative |
| Expert Review | Low | Low | Medium | Low | Both |

An expert review can be understood as a specialised form of peer review which specifies the evaluation criteria, in order to assess the subject matter (Allen, Jones, Dolby, Lynn, & Walport, 2009, pp. 1–2). In this sense, it becomes an evaluative research method. In their paper, Allen et al. (2009, p. 2) note that about two reviewers may be assigned to assess a paper's output. Depending on the size of the subject matter to be evaluated, more experts may be used in order to divide the workload. For Allen et al. (2009, p. 2), this meant a total of 16 expert reviewers for 716 papers. These figures correlate to a 'Low' typical sample size, similar to a focus group, as well as elite and expert interviews.

In terms of interaction between participants, Allen et al. (2009, p. 2) note that experts were required to review the papers independently, making use of scales between 1 and 4. This indicates a 'Low' level of interaction between the experts, similar to interviews and surveys. Notably, this is an example of quantitative measurement. Typically, reviews also allow additional comment and elaboration, which is qualitative. Therefore, akin to a Delphi study, an expert review may use both kinds of measurements.

An expert review is no exception to the rule with regard to expert sampling. Expertise must be relevant, in order to provide an accurate assessment (Allen et al., 2009, pp. 2, 6). As such, short of becoming an 'elite review' – as understood by the term 'elite', an expert review would entail 'Medium' selectiveness, much like Delphi studies, and expert interviews, panels, or surveys.

Due to a typically small sample size, an expert review should not be used to generalise the findings to a large population. Despite this, larger sample sizes may be used if they are available to overcome this limitation (Allen et al., 2009, p. 1). However, larger sample sizes are more difficult to code when qualitative. Depending on the chosen format of the expert review, with a large sample size, it may be more correct to call it an expert survey, foregoing qualitative measurements. On the other hand, small sample sizes, much like Delphi studies, can incorporate both qualitative and quantitative measurements.

The validation method chosen for the proposed SecMVC model is that of an expert review. An expert review, as understood by this research study, can also utilise the type of remote correspondence procedure found in variations of an elite interview. It may also share the usage of background content – to better inform the expert. As it is a review rather than an interview, the emphasis is placed on background content availability with regard to an evaluated subject matter.

This research method was chosen because of the importance of better understanding in qualitative analysis, along with the support of quantitative measurement, where a relevant representative population, in this instance, would be difficult to acquire. Of the other potential qualitative methods available, the researcher is of the opinion that an expert review is the most suited to this research study's requirements. Experts were chosen with specific attention to the validity of their insights; and they are provided with sufficient detail to make informed judgements about the SecMVC model, thereby satisfying the evaluation requirements of the design science process.

The process associated with the chosen research method is discussed in the following section.

## 5.3. The Expert Review

Experts were chosen, based on their field of expertise and qualification level within that field. Experts within the School of Information and Communication Technology (ICT) in the Nelson Mandela Metropolitan University (NMMU) were not considered – so as to avoid any conflict of interest. After confirming participation, the experts were each sent an email outlining the expert review process, along with any relevant documents. An example of this email is shown in Appendix B1.

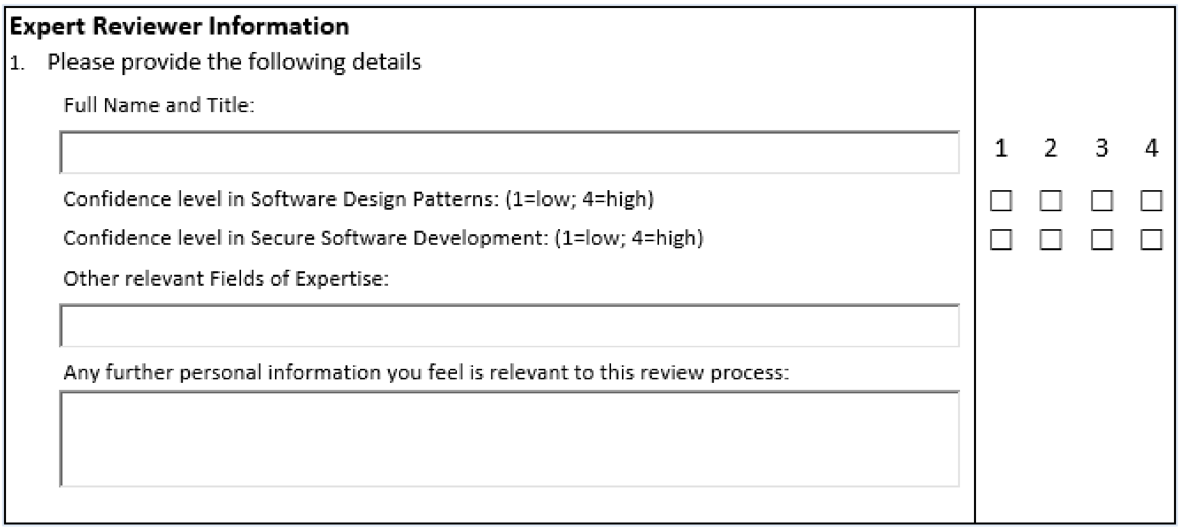
Accompanying the email was a background document, shown in Appendix B2. This provided context for the SecMVC model, as well as additional content through the expert review's appendices. The appendices used included an excerpt of the motivation for the summary of general secure software design principles, these principles and their underlying secure software design concepts, and examples of usage. These appendices are presented in Appendix B3, Appendix B4, and Appendix B5, respectively.

Both the appendices and the background for the expert review shared a list of references in a separate document. This is shown in Appendix B6. Finally an expert review questionnaire document was also attached, as presented in Appendix B7.

The expert review, in the form of a questionnaire document, provided details on how to complete the questionnaire, the questionnaire itself, and a final statement thanking the expert for their contribution. The entire expert review was also taken through a pilot or pre-launch test, prior to its completion.

The expert review was initially drafted, and then evaluated by peers in a colloquium meeting, to determine where improvements could be made to acquire useful evaluative feedback. This resulted in changes to the supporting content and the questionnaire. A Likert scale was used, instead of mostly binary questions, and these questions were modified, in order to be more consistent and definitive in the evaluation of the model.

The original expert review questionnaire is presented in Appendix B8; and a list of evaluating peers is provided in Appendix B9. In the final version, the questionnaire was split into various sections covering expert reviewer information, utility, quality, and efficacy of the SecMVC model, as well as the overall impression. The expert reviewer information section is depicted in Figure 5.1.


**Expert Reviewer Information**

1. Please provide the following details

Full Name and Title:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Confidence level in Software Design Patterns: (1=low; 4=high) | ☐ | ☐ | ☐ | ☐ |
| Confidence level in Secure Software Development: (1=low; 4=high) | ☐ | ☐ | ☐ | ☐ |

Other relevant Fields of Expertise:

Any further personal information you feel is relevant to this review process:

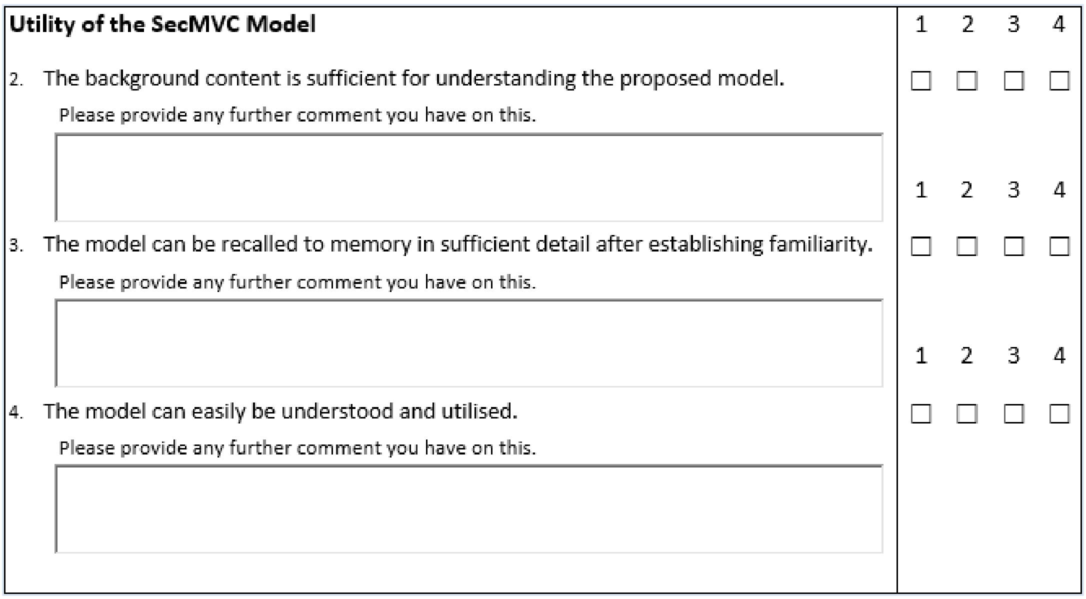
Figure 5.1: Expert Reviewer Information

Notably, names were removed, in order to provide anonymity. Experts were asked to indicate a level of confidence, from low to high, in each of the two primary fields relating to the SecMVC model. Further details pertaining to the experts were asked, as closed-ended questions, or only as being relevant to the SecMVC model and the review process.

The questionnaire used 'Yes' or 'No' questions, and a four point Likert scale, to determine the extent to which the expert agreed or disagreed with the statements pertaining to the SecMVC model. Opportunity to provide further comment, or to elaborate on the answers was provided, wherever it could contribute to greater clarity.

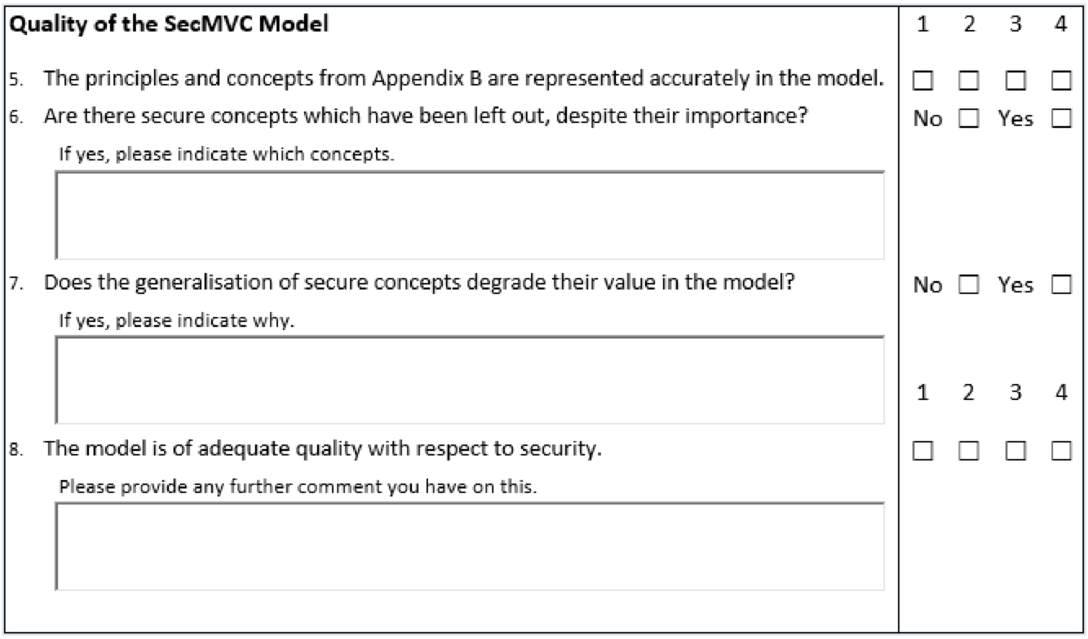
The use of utility, quality, and efficacy as components for assessing research value is a technique used by design science, as well as in other forms of evaluation (vom Brocke & Buddendick, 2006, p. 594; Wang & Wu, 2009, p. 5).

In the utility, quality, and efficacy sections, one or more finalising statement or question was used, in order to establish a definitive stance. These sections are shown in Figure 5.2, Figure 5.3, and Figure 5.4, respectively.



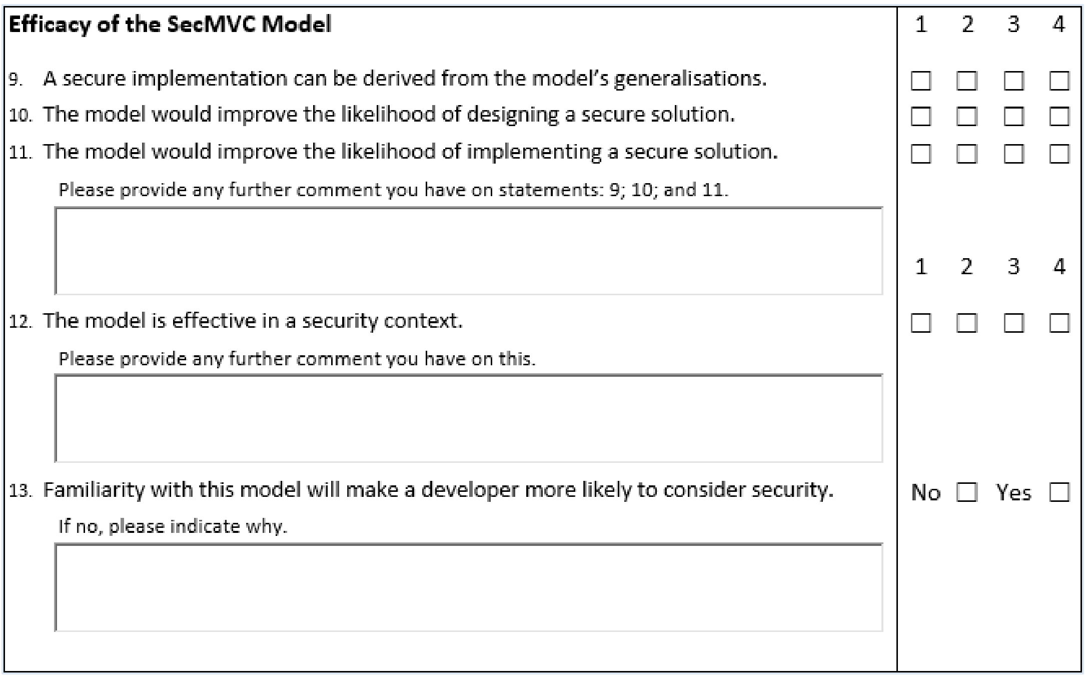
| **Utility of the SecMVC Model** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 2. The background content is sufficient for understanding the proposed model. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |
| | 1 | 2 | 3 | 4 |
| 3. The model can be recalled to memory in sufficient detail after establishing familiarity. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |
| | 1 | 2 | 3 | 4 |
| 4. The model can easily be understood and utilised. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |

Figure 5.2: Utility of the SecMVC Model



| **Quality of the SecMVC Model** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 5. The principles and concepts from Appendix B are represented accurately in the model. | ☐ | ☐ | ☐ | ☐ |
| 6. Are there secure concepts which have been left out, despite their importance? | No | ☐ | Yes | ☐ |
| If yes, please indicate which concepts. | | | | |
| 7. Does the generalisation of secure concepts degrade their value in the model? | No | ☐ | Yes | ☐ |
| If yes, please indicate why. | | | | |
| | 1 | 2 | 3 | 4 |
| 8. The model is of adequate quality with respect to security. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |

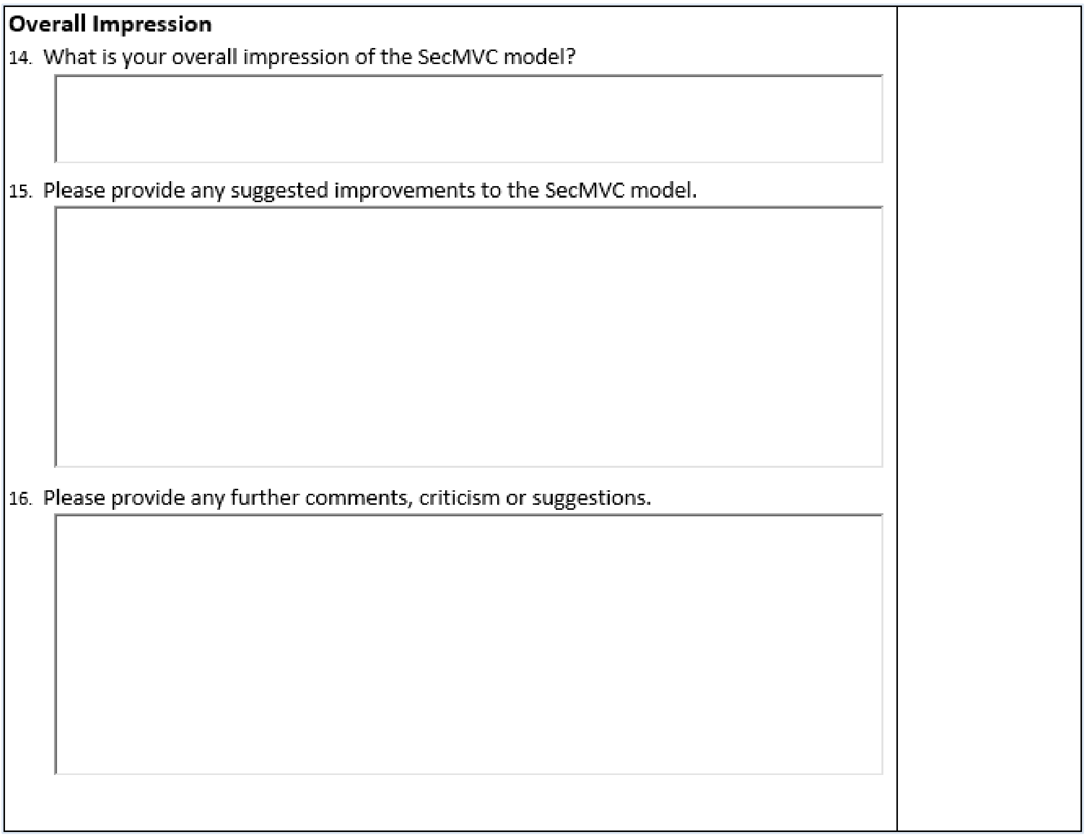
Figure 5.3: Quality of the SecMVC Model



| **Efficacy of the SecMVC Model** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 9. A secure implementation can be derived from the model's generalisations. | ☐ | ☐ | ☐ | ☐ |
| 10. The model would improve the likelihood of designing a secure solution. | ☐ | ☐ | ☐ | ☐ |
| 11. The model would improve the likelihood of implementing a secure solution. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on statements: 9; 10; and 11. | | | | |
| | 1 | 2 | 3 | 4 |
| 12. The model is effective in a security context. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |
| 13. Familiarity with this model will make a developer more likely to consider security. | No | ☐ | Yes | ☐ |
| If no, please indicate why. | | | | |

Figure 5.4: Efficacy of the SecMVC Model

Finally, the overall impression of the SecMVC model was established with the opportunity to elaborate on suggested improvements, as well as comments, criticism and suggestions. This section of the questionnaire is shown in Figure 5.5.



**Overall Impression**

14. What is your overall impression of the SecMVC model?

15. Please provide any suggested improvements to the SecMVC model.

16. Please provide any further comments, criticism or suggestions.

Figure 5.5: Overall Impression

The underlying format of the expert review is discussed in this section, both through iterations and the resulting questionnaire sections shown. In the following sections, this chapter discusses the potential reviewers, followed by the feedback provided by the respondents; and it then provides an overview of this feedback. Suggested improvements to the SecMVC model are discussed and or implemented thereafter.

## 5.4. The Expert Reviewers

The experts chosen for evaluation included a sample of six potential reviewers. These experts were contacted via email to determine whether they would be willing to evaluate the model.

Two of the experts were penetration testers, who reported to a single person with whom the researcher had contact. However, these experts did not indicate an interest in the study. Another penetration tester was contacted directly, but was also not able to complete the review in time. A successful PhD candidate in the domain of security patterns agreed to assist with the review, although only towards the deadline did this individual become unavailable – due to other responsibilities.

These reviewers would have been considered experts – due to the broad software security knowledge typically possessed by a penetration tester. Furthermore, a successful PhD candidate with a background in patterns and security would hold both the relevant and extensive expertise sought.

The two reviewers, who returned a completed evaluation timeously, provided very helpful information. The first of these is a professor in the Computer Science field with specific experience in software design patterns, among additional expertise in patterns and guidelines, as well as other computing and software development fields. This respondent is considered to be an expert – due to the strong background in software design patterns.

The second reviewer is a security, privacy and pattern oriented expert in the software development field. The insights from an expert in the broader focus areas are of considerable value. The following section presents their feedback. The third review, which was returned after the deadline, is only discussed at a high level, along with the first two expert reviews in Section 5.7.

Appendix C1 presents the feedback received from the first expert, who responded further with an evaluation in the expert review questionnaire. As the reviews are anonymous in the presentation of this study, this expert is referred to as Expert 1.

## 5.5. Interpretation of the First Expert Review

Expert 1 indicated high confidence in software design patterns, but low confidence in secure software development. Despite this, he indicated that he is well equipped to evaluate secure aspects of the model by means of thorough analysis. Although he indicated that he is not an Information Security expert, it is understood that security in general was implied.

### 5.5.1. Utility of SecMVC According to Expert 1

Expert 1 made reference to the background document in comment on Question 2. Here, he indicated the need for further insight in software design patterns than provided in the background document. In the context of the background document, which is merely a summary, this is correct. Chapter 3 does, however, already present the further insight suggested. Furthermore, the use of the MVC pattern as a medium for conveying secure software design principles and concepts, was described as contradictory to the purpose of a software design pattern. This is indeed correct, which is why the SecMVC model is not intended as a pattern, but rather as a model.

Expert 1 also mentioned that the MVC pattern is a "very commonly used pattern", which is indeed a key argument, as to why it was used. In addition, as a basis for this research, it is also ideal for web-based applications, and it contains other patterns, which can be used to further elaborate on the content. Expert 1 answered Question 2 with a '3' (agree), indicating that the background content was sufficient for understanding the SecMVC model.

Furthermore, Expert 1 indicated that the model could be remembered sufficiently after establishing familiarity, answering Question 3 with a '3' (agree). Further comment was provided, noting that recalling the "top level issues [was] quite easy". However, it is also noted in the comment for Question 3 that specific details, meaning the secure software design concepts themselves, were not so easy to recall.

In Question 4, which determines the reviewer's overall stance on the utility of the SecMVC model, Expert 1 answered with a '2' (disagree). This indicated that the reviewer believed that the SecMVC model cannot be both easily understood, as well as easily utilised. In the further comment area, he clarified that while the model and its supporting content could be easily understood, as indicated previously, it is not so useable without its supporting content.

This suggests that the SecMVC needed to undergo a further revision, in order to incorporate further detail from its supporting content – possibly in supporting sub-models, as indicated by Expert 1 in the same comment. Expert 1 also suggested the use of a pattern language, seemingly specific to secure software design. While this suggestion presents a worthwhile direction to investigate in the future, it is currently beyond the scope of this research study.

### 5.5.2. Quality of SecMVC According to Expert 1

Expert 1 noted in Question 5 that the representation of secure software design principles and concepts, according to Appendix B4, was accurate, answering with a '3' (agree). He also indicated that no important secure concepts had been left out in Question 6. However, it is again noted that due to the abstract nature of the model, information is not fully represented.

Question 7 poses the question: "Does the generalisation of secure concepts degrade their value in the model?" Expert 1 answered 'No'. This, despite Expert 1 indicating that these concepts were not fully represented. From this, it may be deduced that Expert 1 sees value in a generalised depiction – despite the lack of a full representation.

Question 8 determines an overall stance on the quality of the SecMVC model, with respect to security. Expert 1 answered, 'Yes', although he clarified that this was considered along with the supporting content.

### 5.5.3. Efficacy of SecMVC According to Expert 1

In Questions 9, 10, 11, and 12, Expert 1 answered with '3' (agree). In the comment section, he clarified that this was with the supporting content inclusion being taken into account.

This indicated that the reviewer believed that with the supporting content, despite the generalisation of secure software design principles and concepts, secure implementations could well be derived. These answers indicate that with supporting content considered, the expert believed the likelihood of designing and implementing secure solutions would be improved.

Question 12 was used to assess the overall stance on efficacy; and asks whether the model would be effective in a security context. Expert 1 responded with '3' (agree) for this question, with further clarification of the need for supporting content to be included with the model. Question 13 asks whether familiarity with the SecMVC model 'will make a developer more likely to consider security'. Expert 1 responded 'Yes' to this question.

#### 5.5.4. Overall Impression of SecMVC According to Expert 1

Expert 1 indicated a favourable overall impression of the SecMVC model, noting that it was a good start towards a comprehensive structured resource for addressing secure software design. Suggested improvements by this reviewer included the expansion of the model into a “software security pattern language”.

By this, it is understood that the reviewer believed that further ‘patterns’, as opposed to models, could be created by using a standardised language. Although this is a direction worth investigating in future work, it is not within the scope of this research study. The reviewer also kindly offered further assistance where required.

Appendix C2 presents the feedback received from the second expert. As the reviews are anonymous in the presentation of this study, this expert is referred to as Expert 2. The next section interprets the feedback from the second expert reviewer.

## 5.6. Interpretation of the Second Expert Review

Despite having expertise in the field of security protocols specifically, this reviewer chose to indicate a confidence level of ‘2’ (medium-low). This is probably due to the specific use of the term ‘secure software development’, as opposed to software security in general.

Furthermore, Expert 2 only indicated a confidence level of ‘2’ (medium-low) for software design patterns. This, too, was probably due to the high level of focus in the term ‘software design patterns’ as opposed to patterns in general. Therefore, it is taken into consideration that the confidence level might well be higher than indicated.

### 5.6.1. Utility of SecMVC According to Expert 2

In Question 2, which asks whether the provided background content is sufficient for understanding the SecMVC, the reviewer indicated ‘2’ (disagree). This means that not enough content was provided with the expert review for the reviewer to fully comprehend the model.

This is made clear by the further comment, where he noted that the purpose of the model was not sufficiently clear. While the model could be used in an educational context, its primary purpose is for use in practice by developers who already have an understanding of the MVC pattern.

Question 3 received a ‘3’ (agree) from Expert 2, which indicated that from his perspective, all relevant information could be remembered if familiarity were established. Question 4 determines the overall stance for utility with the answer of another ‘3’ (agree). However, further comment clarified that utilisation was, in his opinion, still limited.

Expert 2 stated that "…the utility is restricted to those cases where the [MVC pattern] paradigm is applicable", which indicates that the reviewer interpreted the SecMVC model as an implementation specific to the MVC pattern's paradigm.

This may be due to the belief that the model addresses those specific cases which use the MVC pattern. The model should, on the other hand, be useful in all software design situations, as its logic is not dependent on the MVC pattern. As it is intended as a general model for conveying secure software design principles and concepts, the goal is not to tie it to the MVC pattern, but to use the common understanding of the pattern to serve as a medium for representation. These principles and concepts are applicable to any software design which needs to consider security, especially web-based applications.

### 5.6.2. Quality of SecMVC According to Expert 2

On the 'Quality of the SecMVC Model', Expert 2 supplied favourable responses to each question. For the accurate representation of the information, a '3' (agree) was given.

To whether any important secure software design concepts had been omitted, Expert 2 answered 'No'. Expert 2 also answered 'No' to whether generalisation degraded the value of the information in the SecMVC model.

To Question 8, which determines the expert's overall stance on the quality of the SecMVC model with respect to security, Expert 2 answered with a '3' (agree). Further comment was provided, suggesting the need for further validation against sets of best practices, such as ISO 27001. Notably, ISO 27001 (ISO/IEC 27001, 2005), as well as ISO 13335 (ISO/IEC 13335-1, 2005), are information security best practices. While related to secure software development and design, these are higher level practices for management rather than for developers.

Despite this, ISO 27001 does make mention of several recommendations which tie into the suggested secure software design concepts (ISO/IEC 27001, 2005, pp. 6, 21–25, 28). These include:

- Management and detection of errors and security incidents;
- Logging and protection of log information;
- Access control and privilege;
- Authentication;
- Limitation of time, access, and utility;
- Validation, integrity, and encryption; as well as
- Privacy and protection of sensitive information.

Some aspects not catered for by the model which could be better represented include continual improvement and regular review (ISO/IEC 27001, 2005, pp. 9, 14). These recommendations were considered to be valid in the greater development context rather than in specific secure software design, although they should be taken into account as best practices in general.

### 5.6.3. Efficacy of SecMVC According to Expert 2

In the 'Efficacy of the SecMVC Model' section, Expert 2 once more supplied favourable responses. By indicating '3' (agree) for Question 9 to 11, Expert 2 showed that he was in agreement with the model's capability to provide specific implementations from its generalisations, as well as to enhance the likelihood of secure solution design and implementation. Further comment was provided for these, noting that while the concept was feasible, and the reviewer believed that it would work, knowing this would be easier with a practical evaluation.

Furthermore, by indicating a '3' (agree) for Question 12, Expert 2 showed that he agrees that the model is effective in the specific context it was designed to address. Question 13 was also answered with a positive response, indicating 'Yes' to a familiarity with the model contributing to a developer's likelihood for consideration of security.

### 5.6.4. Overall Impression of SecMVC According to Expert 2

In response to Question 14, the reviewer commented that the approach taken with regards to the SecMVC model was one that was interesting, and that it should be applied to further patterns. This could indeed be done in further research. However, due to previous suggestions, this could perhaps be implemented further in the component software design patterns of the Compound MVC pattern. Question 15 was answered with similar feedback to Question 14, suggesting the use of further patterns in varying circumstances.

The returned expert review feedback is presented and discussed in the previous Section 5.5 and this Section 5.6. The following section discusses these reviews, together with an overall evaluation of the SecMVC model, in addition to implementing some suggested improvements.

## 5.7. Consolidation of Reviews

This section considers the feedback provided by respondent expert reviewers. This is used to establish whether the SecMVC model is a research output, which is of value, taking into account, of course, that the suggested feasible improvements are fully implemented.

The consolidated respondent feedback is presented in a summary in Table 5.2. The questions are shown simplified in the 'Summarised Question' column. These only represent their corresponding full versions referenced by the '#' column, and should not be taken in isolation. The questions are split between confidence level, utility, quality, and efficacy.

Table 5.2: Consolidation of Reviews

| Section | # | Summarised Question | Expert 1 | Expert 2 | Expert 3 | Overall | Revised |
|---|---|---|---|---|---|---|---|
| **Confidence Level** | 1 | Software Design Patterns | 4 | 2 | 3 | 3 | N/A |
| | 1 | Secure Software Development | 1 | 2 | 4 | 3 | N/A |
| **Utility** of the SecMVC model | 2 | Sufficient background content | 3 | 2 | 3 | 3 | N/A |
| | 3 | Recollection after familiarity | 3 | 3 | 2 | 3 | 3 |
| | 4 | Easily understood and utilised | 2 | 3 | 1 | 2 | **3** |
| **Quality** of the SecMVC model | 5 | Appendix B representation | 3 | 3 | 3 | 3 | 3 |
| | 6 | Secure concepts left out | No | No | Yes | No | No |
| | 7 | Generalisation degradation | No | No | Yes | No | No |
| | 8 | Quality with respect to security | 3 | 3 | 3 | 3 | **3** |
| **Efficacy** of the SecMVC model | 9 | Secure implementation derived | 3 | 3 | 3 | 3 | 3 |
| | 10 | Secure designing more likely | 3 | 3 | 3 | 3 | 3 |
| | 11 | Secure implementing more likely | 3 | 3 | 3 | 3 | 3 |
| | 12 | Effective in security context | 3 | 3 | 3 | 3 | **3** |
| | 13 | Secure consideration more likely | Yes | Yes | Yes | Yes | Yes |

The 'Overall' column represents the collective interpretation, which takes into account the comment section responses, the perceived understanding, and the experts themselves. The value shown in each row of the 'Revised' column corresponds to the resultant outcome interpreted by this research study – considered after implementing the suggested amendments as discussed in Section 5.8. The value in bold font indicates the total levels of utility, quality and efficacy.

In the confidence level section, the values for software design pattern confidence is set at '3'. This is due to the high confidence level of Expert 1, as well as the perceived level of expertise in Expert 2. Expert 3 indicated a level '3' confidence level. While '2' is considered low for Expert 2, an average of '3' is still achieved.

In terms of secure software development, it is noted that both Expert 1 and Expert 2 provided lower confidence levels than implied by further comment, such as the use of "security protocols [and thorough] homework". In addition to this, Expert 2 is known to have experience in software security in general. Despite this, the average value still rounds to '3', taking into account Expert 3's high level of confidence. As experts in an expert review need not be considered 'elite', this research study considers these values to be acceptable.

The next section discusses the overall interpreted utility of the SecMVC model.

### 5.7.1. Utility of the SecMVC Model

Question 2, regarding sufficient background content, is less about evaluating the model and more about validating the utility of the expert review background document and appendices, as having provided sufficient information to evaluate the SecMVC model. Expert 1 noted that additional software design pattern background was needed. Despite this, he selected ‘3’; while Expert 2 indicated that this was not sufficient for understanding the “purpose of the model”. Expert 3 indicated a misinterpretation regarding the MVC pattern’s separation of concerns, as opposed to the SecMVC model’s sections used as an aid for understanding. Secure software design concepts are not separated by these sections.

As such, the background content provided by the expert review could have been more detailed. The lack of stress on the SecMVC model being a model, and not a software design pattern, has led to slightly less useful comments in particular. Despite this, it is believed that sufficient validation and valuable insight have, nonetheless, been provided. While the background content could have been better, it is not inadequate. Therefore, the average value is calculated as ‘3’.

With regard to Question 3, summarised ‘recollection after familiarity’, both Expert 1 and Expert 2 responded with a ‘3’. Expert 3 indicated that the SecMVC model was complex. While he noted that the flow was not obvious, there is no directional flow between secure software design concepts in the SecMVC model. Expert 1 clarified that specific details were not as easy to recall. For this purpose, the SecMVC model is appended with three sub models in Figure 5.6, Figure 5.7, and Figure 5.8. These sub-models are presented and described in Section 5.8. The average value for Question 3 remains ‘3’.

In Question 4, an overall stance on the utility of the SecMVC model was determined. In this, Expert 1 commented that the SecMVC model “needs all the detailed sub-models”, as “the [SecMVC] model by itself is too abstract and limited”. Sub-models have been implemented, as a result of this feedback and are presented in Section 5.8. Expert 2 clarified his answer, indicating that utilisation might be limited to MVC pattern applications. Expert 3 drew a similar conclusion, noting the limitation in .NET usage. Consequently, he was correct in noting that the concepts could be applied generically. However, this research study only endeavoured to review .NET usage; and consequently, it cannot claim any broader generalisation.

A shortfall of the background document, therefore, is that it did not render the SecMVC model's use as a generalised model for all security design situations, especially in the .NET context, sufficiently clear. It is interpreted that modifications, based on the suggestions provided by the experts, increases the average value for utility. Due to the amendments implemented in Section 5.8, the utility of the SecMVC model should achieve the value of '3'. This indicates that the design science artefact, the SecMVC model, can be interpreted to be of **sufficient utility**.

The following section discusses the overall perceived quality of the SecMVC model.

### 5.7.2. Quality of the SecMVC Model

Question 5 determined whether the experts believed that the secure software design principles and concepts presented in Appendix B of the expert review, and in Appendix B4 of this research study, were *accurately* represented. Despite Expert 1 indicating that the specific details were not *sufficiently* represented in the SecMVC model (prior to suggested amendments), their depiction was still considered *accurate*. Expert 1, Expert 2, and Expert 3 all indicated a value of '3' for this question.

Question 6, on the other hand, determined whether *secure concepts* had been left out, despite their importance. Both Expert 1 and Expert 2 answered 'No', maintaining the value for the overall interpretation. Expert 1, however, mentioned that *secure software design concepts* were not sufficiently represented (prior to suggested amendments). It is interpreted that Expert 1 understood *secure concepts* as potentially more than the *secure software design concepts* in Appendix B4.

From this, the 'important' *secure concepts* were, therefore, not left out of the lists of *secure software design concepts*, even if they were left out of the SecMVC model in isolation (prior to suggested amendments). Clarification was also given that Expert 1 understood the abstractive nature of the SecMVC model. This reinforces the decision to extend the model with sub-models, as opposed to merely updating the model with further detail.

Expert 3 instead indicated 'Yes' for Question 6, but did not note which concepts had been left out. These are, however, apparent in his comments to Question 8. Expert 3 pointed out that *layer your defence* may need re-working, and suggested that the SecMVC use more generic security concepts.

Question 7, similarly, sought to determine whether generalisation degraded the secure concept value in the SecMVC model. Expert 1 reinforced his earlier statement about abstraction. He indicated through the comment “not applicable”, along with an answer of ‘No’, that generalisation in this case did not take away from the quality of the SecMVC model. Expert 2 also responded ‘No’, while Expert 3 answered ‘Yes’, clarifying that while there was sufficient representation, “a normal developer may not fully grasp all concepts contained under the generalisation”. For the purpose of this study, ‘No’ remains the overall interpreted answer.

Question 8 determined an overall stance on the quality of the SecMVC model, specifically with respect to security. Expert 1, Expert 2, and Expert 3 all answered with ‘3’, with Expert 1 indicating the need for further detail. This issue is addressed in the amendments to the SecMVC model. Expert 2 noted the value of a comparison against ISO 27001, which is discussed in Section 5.6.2. These answers, along with the suggested amendments in Section 5.8, indicate that overall the SecMVC model, the design science artefact for this research, is of **sufficient quality**.

The following section discusses the overall efficacy of the SecMVC model.

### 5.7.3. Efficacy of the SecMVC Model

In Questions 9, 10, and 11, the SecMVC model’s ability to derive secure implementations, and increase the likelihood of designing and implementing secure solutions is determined. Expert 1, Expert 2, and Expert 3 all indicated an answer of ‘3’ for all three questions. As such, each overall value maintains the value of ‘3’. Expert 1 noted the need for the model to be taken together with the suggested further detail. Expert 2, however, indicated that although it was difficult to assess, he approved of “the general idea and [believes] it [would] work”. Expert 3 indicated a similar stance on this.

Question 12 determined the overall stance on the efficacy of the SecMVC model with specific respect to security, asking for expert opinion on the statement ‘the model is effective in a security context’. Expert 1, Expert 2, and Expert 3 all answered with ‘3’, with Expert 1 clarifying once more that this was with further detail inclusion considered. As this further detail is achieved through the sub-models presented in Section 5.8, the design science artefact of this research, the SecMVC model, is interpreted to be of **sufficient efficacy**.

Finally, Expert 1, Expert 2, and Expert 3 all indicated that ‘familiarity with [the SecMVC] model will make a developer more likely to consider security’. Expert 3 also indicated a favourable overall impression, suggesting further detail, review, and reinforcing the value of several aspects mentioned in Chapter 2, but not contained in the background document.

With the aforementioned overall values considered, the SecMVC model contributes to a candidate solution of *consideration for security* to the problem statement posed in Chapter 1, Section 1.3. This is that '***software security is not consistently addressed during design, which undermines core security concerns, and leads to the development of insecure software***'.

It is interpreted through the feedback from this expert review that the research output and design science artefact, the SecMVC model, is of **sufficient research value**. The following section implements and discusses the suggested amendments to this model by the participants of the expert review.

## 5.8. Amendments to the SecMVC Model

The SecMVC model is based on the MVC pattern. This software design pattern is a Compound pattern, which includes multiple patterns as components of the overall MVC pattern. As mentioned in Chapter 3, Section 3.6, the components include: the Composite pattern, the Strategy pattern, and the Observer pattern.

The suggested improvements to the SecMVC model by participants in the expert review discussed in this chapter specifically advocate that further patterns be used, and that further detail be included from the secure software design concepts in Appendix B4. The proposed implementation that addresses these suggestions is the extending of the SecMVC with the MVC pattern's component software design patterns as further sub-models.

The first of these is shown in Figure 5.6. This sub-model combines the *layer your defence* secure software design principle and its concepts with the Composite pattern. These include elements from the SecMVC model, replacing Code Access Security (CAS) with Intrusion Detection Systems (IDS). Greater emphasis is made on access, strong naming, adjustments and encryption. For example, the encryption elements include 'sensitive data', 'credentials', 'keys', 'authentication functions', and a lack of 'hard-coding'.

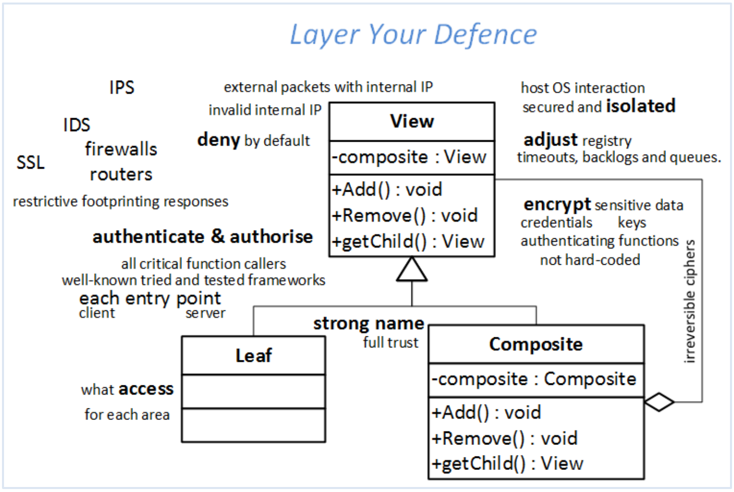


Figure 5.6: SecMVC Composite Sub-model

Additional information is provided for authentication and authorisation. In this, the use of 'well-known tried and tested frameworks' in 'each entry point' for 'all critical function callers' is advocated. Denial and isolation elements are also featured with their respective relating considerations.

Unlike the SecMVC, this sub-model does not hide the details of the Composite pattern itself. This is due to the fact that while the MVC pattern may be considered common and well-understood, the same might not necessarily be true of its component software design patterns. In addition, these sub-models are intended to present further detail than the SecMVC model, which is intended as an abstraction or generalisation.

Summarised secure software design concepts within the SecMVC model, specifically, or by relation, are further emphasised by way of a larger font size and bold font style. Further detail relating to these is placed adjacent to the concept, with even greater detail placed further apart. An example of this can be seen between the 'deny' concept, 'by default', 'invalid internal IP', and 'external packets with internal IP' in Figure 5.6.

The second sub-model is shown in Figure 5.7. This sub-model incorporates the *all input is evil* secure software design principle into the Strategy pattern. Like the SecMVC Composite sub-model, prominent summarised secure software design concepts are given a larger and a bold font. Concepts which are particularly relevant, but not emphasised in the SecMVC model are given larger font sizes without the bold font style, such as 'closed-ended' inputs, defaults, filters, and dynamic lists.

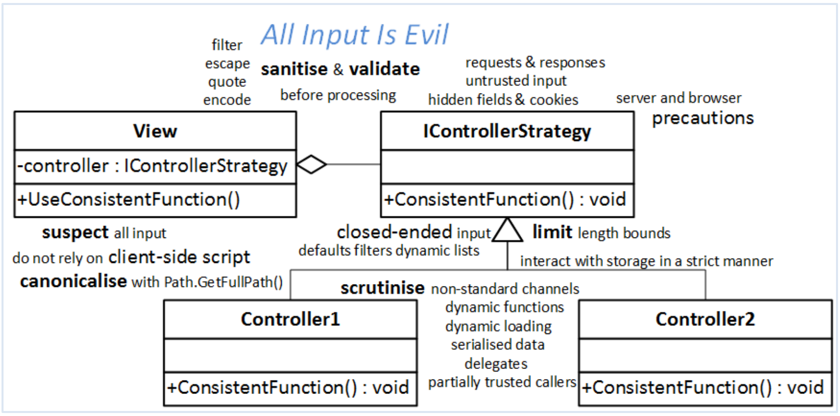


Figure 5.7: SecMVC Strategy Sub-model

The 'scrutinise' concept in particular, lists the other concepts which should be given special care with regard to security. The 'sanitise & validate' concept serves a similar purpose. Both should occur prior to processing, and the left-hand concepts (filter, escape, quote, encode) represent *how;* while the right-hand concepts (requests & responses, untrusted input, hidden fields & cookies) indicate *what*.

The final sub-model combines both *least privilege* and *output securely* secure software design principles. These are incorporated into the Observer pattern in Figure 5.8. This SecMVC Observer sub-model includes concepts from both secure software design principles. It includes lists and relational depictions of the summarised concepts.

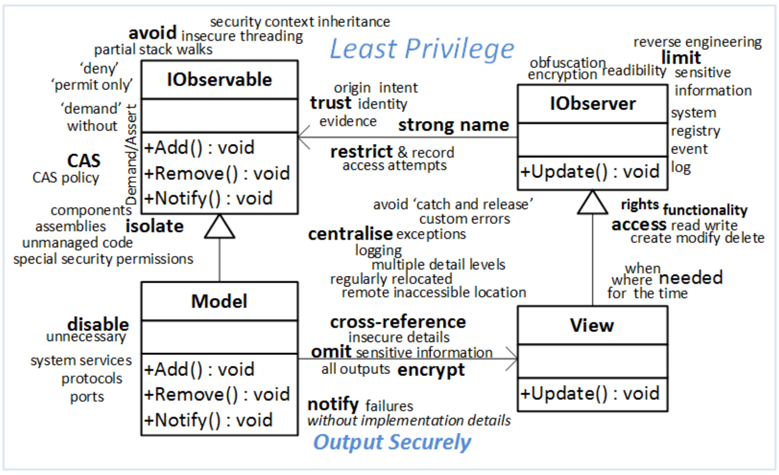


Figure 5.8: SecMVC Observer Sub-model

These summarised secure software design concepts permit far more detail than in the SecMVC model. Particularly, some smaller font concepts without a bold font style are used, beside a prominent concept, in order to give further context to its relevance around the surrounding concepts. Examples of this in the SecMVC Observer sub-model include: 'trust' and 'evidence'; 'disable' and 'unnecessary'; as well as 'when', 'where', 'for the time', and 'needed'.

Notably, in the Code Access Security (CAS) section, 'demand' should either be *avoided* or *isolated*, and if used, it should utilise the Demand/Assert technique. Furthermore, 'system', 'registry', 'event', and 'log' *access* should be *limited*.

These three sub-models collectively provide further detail to the SecMVC model, as such satisfying the suggested amendments. The following section concludes on the validation of the SecMVC model.

## 5.9. Conclusion

This chapter presents and discusses the validation of the SecMVC model through the expert review method as described in Chapter 1, Section 1.6.2. This began with an introduction to the expert review through similar research methods, which each shared one or more commonalities. Once presented and the relevance discussed, the expert review is argued as a suitable evaluation method and validation technique for the SecMVC model.

Potential expert reviewers were identified, with two respondents able to provide feedback before the prescribed deadline. The SecMVC model was evaluated by the participant experts. This evaluation is presented and discussed in isolation, followed by an overall validation discussion featuring additional input from a third expert. The SecMVC model is determined to be of research value. However, in order to achieve this, suggested improvements by the experts have been implemented. These improvements resulted in the development of three detailed SecMVC sub-models relating to the MVC Compound pattern’s contained Composite, Strategy, and Observer patterns.

# Chapter 6: Conclusion

## 6.1. Introduction

This chapter serves to discuss the achievement of the primary research objective and its supporting secondary objectives. This purpose is complemented by an exploration of the contributions this research study has made to the body of knowledge, as well as their significance, and what lessons have been learnt as a result of its production. Finally, a call is made with suggestions for further related research.

The problem this research study identified in Chapter 1, and sought to address is that ***software security is not consistently addressed during design, which undermines core security concerns, and leads to the development of insecure software***. This problem is determined in Chapter 1 to be prevalent in industry, leading to both tangible and intangible losses, when exploited.

The stated problem pertains to two major fields: Software Security, and Software Development. Areas within these fields are secure software development and software design patterns. Further delineation into the secure software development area brought the focus to secure software design, a component within secure software development, which considers primarily the design phase of the software development lifecycle.

These two areas converged into the topic of this research study. In addition to research objectives, this chapter determines to what extent ***integrating secure software design principles and concepts into a model using software design patterns*** has been achieved. These issues are discussed in the following section.

## 6.2. Achievement of Research Objectives

The primary objective of this research study is ***to develop a model, based on the MVC pattern, which addresses software security through secure software design***. The Model-View-Controller (MVC) pattern is chosen as a medium for designing the model, based on its popularity in industry, its applicability to web contexts, and its contained patterns – as a Compound pattern. This primary research objective serves to contribute to a candidate solution for the research problem.

Supporting this primary research objective are four secondary objectives, which were instrumental in its achievement. The secondary research objectives defined for this research study are as follows:

- To identify secure software design principles and concepts from the secure software development literature;
- To analyse software design patterns from the literature and the MVC pattern's suitability for this research study;
- To integrate the identified secure software design principles and concepts into the MVC pattern;
- To validate the proposed model, verifying its utility, quality and efficacy.

For the purpose of this chapter, these secondary objectives are summarised as 'Secure Software Development', 'Software Design Patterns', 'The SecMVC Model', and 'Validation', respectively. These secondary objectives are met as discussed in the following sections, titled after their summarised descriptions. Finally, the achievement of the primary research objective is discussed in Section 6.2.5. The first of these is 'Secure Software Development'.

### 6.2.1. Secure Software Development

The first of the secondary objectives sought to ultimately identify both principles and concepts in secure software design, a specific area in secure software development. In order to achieve this, an extensive literature review was performed on the existing research and authoritative literature with regard to broader software security, secure software development, and finally secure software design. The findings from this literature review are discussed and explored in Chapter 2.

This chapter first discusses software security in general, revisiting the stated research problem and its relevance to industry. Various prevalent issues in this field are highlighted. The importance of security and the implications of overlooking its importance are also noted.

Information Security, and the Information Security Goals, are discussed along with insights from the Open Systems Interconnection (OSI) reference model, its components, and its security services. These are used to explore security mechanisms, as they pertain to security in a development context. The focus was reinforced into the design phase of the software development lifecycle, along with specific attention to .NET framework related insights.

From a list of nine related security principles, four general secure software design principles are argued. Identified secure software development concepts from the literature are considered for secure software design specific concepts under the four secure software design principles. These concepts are discussed in detail, and briefly compared with ISO 27001 in Chapter 5.

With the above taken into account, secure software design principles and concepts have been identified from the broader secure software development literature. **Therefore, the secondary objective: 'To identify secure software design principles and concepts from the secure software development literature' has been met**. The next section discusses the following secondary objective, which concerns software design patterns.

### 6.2.2. Software Design Patterns

The secondary objective, which follows the secure software design principle and concept identification objective, sought to analyse a range of software design patterns and their supporting background content, in order to establish whether the MVC pattern is suitable for conveying secure software design principles and concepts.

It is argued that the MVC pattern is common enough to be recognisable and relevant enough to repurpose for a security context. This includes a need to be relevant from a web security context, as many security insights are web specific. Additionally, the MVC pattern needs to be ideal for conveying as many secure software design principles and concepts as possible – without making the model overly complex.

Chapter 3 meets this secondary objective. It firstly introduces the concept of software design patterns, and the format in which these exist, and are shared amongst developers. It also discusses their usefulness and justification for utilisation in practice. Following this, the components of the MVC Compound pattern are discussed, prior to the MVC pattern itself, as well as the related Model-View-ViewModel and Model-View-Presenter patterns.

From these software design patterns, the MVC pattern is determined to be the most common – due to it being the original pattern. The MVC pattern is also well-known, due to its use in the ASP.NET MVC iterations. As it is a Compound pattern, it contains other patterns, which make it ideal for use as a model, and allows its extension in the improvements suggested by the respondents to the expert review.

While the Observer pattern is identified as one of the most useful patterns, it is a component of the MVC pattern. As such, the Observer pattern is used for a sub-model in the amended SecMVC model. This is also the sub-pattern, which used two secure software design principles, as opposed to the others, which used one only.

Considering the analysis of software design patterns in Chapter 3, specifically those relating to the MVC pattern, as well as the determined suitability of the MVC pattern as a foundation for a model conveying secure software design, **the secondary objective entitled: 'To analyse software design patterns from the literature and the MVC pattern's suitability for this research study' has been met**.

The following section discusses the achievement of the secondary objective concerning the output for this research study.

### 6.2.3. The SecMVC Model

The SecMVC model is presented and discussed in Chapter 4. This chapter describes the foundations of the model, which includes the MVC pattern and the SecMVC model components. These components are secure concepts, initially presented under security mechanisms, where in future revisions, secure software design principles and concepts are used.

Subsequently, these variations of the SecMVC are presented in the stages of their production. These iterations have been improved over the course of the study, beginning with the SecMVC model presented in the 2013 South African World Wide Web (ZAWWW) conference (Appendix A). Improvements to the model are discussed along with the iterations.

Beyond the fourth version of the SecMVC model, the model is argued as usable in practice. This argument makes use of the top three security risks and weaknesses, as identified by the Open Web Application Security Project (OWASP) and the Common Weakness Enumeration (CWE).

Through this, **the secondary research objective: 'to integrate the identified secure software design principles and concepts into the MVC pattern' has been met**. The following section discusses the final secondary objective.

### 6.2.4. Validation

The final secondary objective involves the validation of the SecMVC model, the research output and design science artefact for this research study. Specifically, the secondary objective includes the evaluation of utility, quality and efficacy. These are the measures used to evaluate a design science artefact for its research value.

The chosen method for validation of the SecMVC model is an expert review. This specialised research method required some justification, and as such is compared with its closest neighbours, namely:

- Focus groups;
- Delphi studies;
- Elite interviews;
- Expert interviews;
- Expert panels; and
- Expert surveys.

It is argued that an expert review shares similarities with all the above research methods. However, it is more suitable for this study – due to the evaluative nature of the method; the high level of relevant qualitative feedback; the required level of expertise in participants; and the required participant count.

The expert review method is discussed and an instance is designed for use in this study. This instance is explained in Section 5.3, with the following section detailing the chosen potential experts. Of these, two expert reviewers returned their feedback in time.

The feedback from Expert 1 and Expert 2 provided valuable insights into how the SecMVC model could be improved. These improvements are implemented, as discussed later in the chapter. The returned expert reviews, in the form of questionnaires, were analysed – first in isolation – and then together, along with some accompanying comments. Feedback from the third reviewer is included in the consolidated analysis.

Overall, the feedback clarified that after the amendments, implemented in Section 5.8, the SecMVC model is indeed of sufficient utility, quality and efficacy. Therefore, **the secondary research objective entitled: 'to validate the proposed model, verifying its utility, quality and efficacy', has been met**.

The following section discusses the achievement of the primary objective.

### 6.2.5. The Primary Objective

The primary objective for this research is t***o develop a model, based on the MVC pattern, which addresses software security through secure software design***. The secure software design principles and concepts used to address software security, the MVC pattern, their usage in the SecMVC model, as well as the validation of this model were accomplished in supporting secondary objectives. **These secondary objectives collectively meet the requirements of the primary objective**.

The research process used to achieve this followed the design science guidelines, as introduced in Chapter 1, Section 1.6.1. The seven guidelines as noted by Hevner, March, Park, and Ram (2004, p. 83) are as follows:

1. Design as an artefact;
2. Problem relevance;
3. Design evaluation;
4. Research contributions;
5. Research rigour;
6. Design as a research process; and
7. Communication of research.

This research study adheres to these guidelines in the following ways:

1. **Design as an artefact**: The research output for this study, the SecMVC model, was designed as a design science artefact. The artefact was successfully evaluated as such.
2. **Problem Relevance**: The problem identified in Chapter 1, and stated in Section 1.3, notes the lack of consideration for security, and the repercussions this can have on businesses and organisations.
3. **Design Evaluation**: The original SecMVC model presented at the ZAWWW conference was evaluated through peer review; while the current version of the model was evaluated through an expert review by three knowledgeable experts.
4. **Research Contributions**: The output for this research contributes not only the SecMVC model, but is also a proof of concept – thereby indicating that secure software design concepts can be conveyed through software design patterns. In addition, this notion can potentially be extended to other fields.
5. **Research Rigour**: The literature review performed on the methodology, and on both of the focus areas of this research study, was extensive and allowed for rigorous construction and evaluation of the model.
6. **Design as a Research Process**: The SecMVC model was designed as a result of a strict process, from the review of all the relevant literature, to iterative design – with feedback, validation, and finally communication.
7. **Communication of Research**: Part of this research study was presented at the ZAWWW conference; and as a result of this, it evolved into a validated dissertation. The final version is to be published by the Nelson Mandela Metropolitan University (NMMU).

In addition, Peffers, Tuunanen, Rothenberger and Chatterjee (2007, p. 54) refer to these guidelines as 'practice rules', and propose their own design science research methodology process. This process includes: Problem identification and motivation; the objectives of a solution; design and development; demonstration; evaluation; and communication. These were addressed in the following ways:

1. The **problem was identified and motivated** through the literature review.
2. The **objectives were defined**, and then formulated through a further literature review.
3. An **artefact was designed and developed** by means of modelling the SecMVC model.
4. The **artefact was demonstrated** through three examples.
5. The **artefact was evaluated** with an expert review.
6. The **artefact was communicated** in the ZAWWW publication, and through this dissertation.

Therefore, considering the above, it may be said that **the primary research objective has been met, while following a rigorous design science process**.

## 6.3. Contributions

The SecMVC model achieves its purpose of being able to improve the likelihood of consideration for security by developers; and it therefore contributes to the candidate solution of better concern for security. This success is an indication that further software design patterns could be used to achieve similar objectives. While the SecMVC model might not completely solve the problem statement posed in Chapter 1, it is a step in the right direction.

The SecMVC model could be used in practice, to remind developers familiar with the MVC pattern – of their responsibility to security – and of the various ways whereby they can achieve this in their designs. It can also be used in an educational context, using the acquired understanding of the MVC pattern and its components to facilitate further learning in the area of secure software design.

Further contributions include the secure software design principles and concepts themselves, as well as the sub-models accompanying the SecMVC model. In addition, expert reviews, at the time of writing this dissertation, comprise a sparsely documented research method. This research study presents an argument and discussion for the research method, which could be used to further formalise it in future.

## 6.4. Lessons Learned

The expert review and its similar research methods are some of a few lessons learned in this research study. In particular, the way in which an expert review relates to so many varying methods was interesting to discover. It is a meaningful research method, which can provide a great deal of useful feedback – at low cost and in a short time.

The difficulty is in identifying suitable experts as potential participants, and in acquiring a suitable response rate. Notably, a possible extension to this research method, an 'elite review', might provide even more valuable insight if elite experts can be identified and are willing to participate.

It has been fulfilling to carry out this research, as secure software design and development are of particular interest to the researcher. The inclusion of software design patterns in this research study has also strengthened the researcher's understanding of them. An appreciation for patterns in general has been acquired. Furthermore, the researcher's appreciation for security in the software development context has been reinforced. In particular, further insight into other areas in the broader pattern and security domains will be pursued.

## 6.5. Call for Further Research

While the SecMVC model serves its delineated and specific purpose, it also introduces various other directions, which could be undertaken. Firstly, additional software design patterns could be utilised as foundations for further content. Furthermore, an approach utilising a design pattern language could also yield interesting results.

The SecMVC model could be further validated through quantitative evaluation, perhaps in the form of action research in the School of Information and Communication Technology (ICT) at the NMMU. In this, the model could be utilised as a learning device – adapted to other disciplines – or for secure software development in particular, as this is a field which should receive further consideration.

## 6.6. Conclusion

This chapter discusses the conclusion of this research study. It demonstrates how each secondary research objective has been met, as well as how the primary research objective was achieved. It provides a discussion on the research contributions provided by this research study, reflections on lessons learned; and it proposes further relevant research.

This research study has contributed to the researcher's abilities to perform rigorous research; and it has furthered the researcher's understanding of two interesting and diverse disciplines. Furthermore, this study has provided a research output of suitable research value – into which further research could venture.

# References

Ali, M., & Elish, M. O. (2013). A Comparative Literature Survey of Design Patterns Impact on Software Quality. *2013 International Conference on Information Science and Applications (ICISA)*, 1–7. doi:10.1109/ICISA.2013.6579460

Alkussayer, A., & Allen, W. H. (2010). The ISDF Framework: Towards Secure Software Development. *Journal of Information Processing Systems*, *6*(1), 91–106. doi:10.3745/JIPS.2010.6.1.091

Allen, L., Jones, C., Dolby, K., Lynn, D., & Walport, M. (2009). Looking for landmarks: the role of expert review and bibliometric analysis in evaluating scientific publication outputs. *PloS One*, *4*(6), e5910. doi:10.1371/journal.pone.0005910

Ampatzoglou, A., Frantzeskou, G., & Stamelos, I. (2012). A methodology to assess the impact of design patterns on software quality. *Information and Software Technology*, *54*(4), 331–346. doi:10.1016/j.infsof.2011.10.006

Anguelo, A. (2009). Developer, Engineer, or Architect? *IEEE Computer Society*. Retrieved May 21, 2014, from http://www.computer.org/portal/web/buildyourcareer/careerwatch/jt15

Ayoub, R. (2011). The 2011 (ISC)$^2$ Global Information Security Workforce Study. *ISC$^2$ (Ed.), 2011*, 2–27.

Bhattacherjee, A. (2012). *Social Science Research: Principles, Methods, and Practices* (2nd ed.). Textbooks Collection. Retrieved from http://scholarcommons.usf.edu/cgi/viewcontent.cgi?article=1002&context=oa_textbooks

Bishop, M., & Engle, S. (2006). The Software Assurance CBK and University Curricula, 14–21.

Bourgeois, J., Pugmire, L., Stevenson, K., Swanson, N., & Swanson, B. (1948). The Delphi Method : A qualitative means to a better future.

Breu, R., Burger, K., Hafner, M., & Popp, G. (2004). Towards a Systematic Development of Secure Systems.

Burzstein, E., & Picod, J. M. (2010). Recovering Windows Secrets and EFS Certificates Offline. In *Black Hat* (pp. 1–9). Retrieved from http://cdn.ly.tl/publications/Recovering-Windows-Secrets-and-EFS-Certi%EF%AC%81cates-Of%EF%AC%82ine.pdf

Cherdantseva, Y., & Hilton, J. (2013). A Reference Model of Information Assurance & Security. *2013 International Conference on Availability, Reliability and Security*, 546–555. doi:10.1109/ARES.2013.72

Christey, S. M., Kenderdine, J. E., Mazella, J. M., & Martin, R. A. (2013). *Common Weakness Enumeration*. Retrieved from http://cwe.mitre.org/data/published/cwe_v2.3.pdf

Chu, C. (2008). Introduction to Microsoft. NET Security. *IEEE Security & Privacy*. Retrieved from http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000004753678

Collins, W., & Anderson, S. (2006). Collins English Dictionary: Complete & Unabridged.

Dorrans, B. (2014). Microsoft Web Protection Library - Home. *CodePlex*. Retrieved August 23, 2014, from http://wpl.codeplex.com/

Ferrara, P., Logozzo, F., & Fahndrich, M. (2008). Safer unsafe code for .NET. *ACM SIGPLAN Notices*, *43*(10), 329. doi:10.1145/1449955.1449791

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head first design patterns*. *Health education research* (Vol. 28, p. i3). doi:10.1093/her/cyt032

Futcher, L. (2011). *An Integrated Risk-Based Approach to Support IT Undergraduate Students in Secure Software Development*. NMMU. Retrieved from http://dspace.nmmu.ac.za:8080/jspui/handle/10948/1673

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software* (p. 395). Addison-Wesley. Retrieved from http://books.google.com/books?hl=en&lr=&id=6oHuKQe3TjQC&oi=fnd&pg=PT2&dq=Design+Patterns:+Elements+of+Reusable+Object-Oriented+Software&ots=lOiFFVfPGB&sig=_tCX5mkpuzrytJ4bXsxEHGADsCM

Haworth, D. (2002). Security Scenarios in Analysis and Design. *The SANS Institute*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.197.2657&rep=rep1&type=pdf

Hevner, A., March, S., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105. Retrieved from http://dl.acm.org/citation.cfm?id=2017217

Hight, S. D. (2005). The importance of a security , education , training and awareness program ( November 2005 ), *27601*(November), 1–5.

Hofstee, E. (2006). *Constructing a Good Dissertation: A Practical Guide to Finishing a Masters, MBA or PhD on Schedule*. Johannesburg: EPE.

Holmström, A. (2011). Performance and Usability Improvements for Massive Data Grids using Silverlight. Retrieved from http://umu.diva-portal.org/smash/record.jsf?pid=diva2:414906

Hornby, A., Cowie, A., Gimson, A., & Lewis, J. (1986). *Oxford advanced learner's dictionary of current English* (3rd ed.). Oxford: Oxford University Press. Retrieved from http://www.abebooks.com/servlet/BookDetailsPL?bi=8275115700&searchurl=bsi=0&ds=30&pics=on&tn=oxford+advanced+learner%27s+dictionary+current+english

Howard, M., & LeBlanc, D. (2009). *Writing secure code*. Retrieved from http://books.google.com/books?hl=en&lr=&id=sjaweoe5-sYC&oi=fnd&pg=PT1&dq=Writing+Secure+Code&ots=3QgoiBPMyn&sig=zQjwoVnk0fqaOW5u7OHC9Xt_KwM

Hussey, J., & Hussey, R. (1997). *Business Research: A Practical Guide for Undergraduate and Postgraduate Students*. London, Macmillan. Retrieved from http://eprints.kingston.ac.uk/id/eprint/5216

International Telecommunication Union. (1991). X. 800 Security Architecture for Open Systems Interconnection for CCITT applications. *ITU-T (CCITT) Recommendation*.

ISO/IEC 13335-1. (2005). *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*. Switzerland.

ISO/IEC 27000. (2009). *ISO/IEC 27000. Information Technology, Security Techniques, Information Security Management Systems, Overview and Vocabulary*. *ISO/IEC* (Vol. 2009). Switzerland. Retrieved from http://www.cqvip.com/qk/96261x/200906/30625262.html

ISO/IEC 27001. (2005). *ISO/IEC 27001. Information technology - Security techniques - Information security management systems - Requirements* (1st ed.). Switzerland.

ISO/IEC 27005. (2008). *ISO/IEC 27005. Information technology - Security techniques - Information security risk management*. *Information Security Risk Management*. Switzerland. Retrieved from http://link.springer.com/chapter/10.1007/978-3-8348-9870-8_3

ISO/IEC 7498-2. (1989). *ISO/IEC 7498-2. Information processing systems - Open Systems Interconnection - Basic Reference Model. Part 2: Security Architecture*. *ISO Geneva, Switzerland*. Switzerland.

Jiang, S., Vaidya, N., & Zhao, W. (2003). Energy consumption of traffic padding schemes in wireless ad hoc networks. *Real-Time System Security*. Retrieved from http://dl.acm.org/citation.cfm?id=903870

Jones, R., & Rastogi, A. (2004). Secure coding: building security into the software development life cycle. *Information Systems Security*. Retrieved from http://www.tandfonline.com/doi/abs/10.1201/1086/44797.13.5.20041101/84907.5

Jürjens, J. (2002). UMLsec: Extending UML for secure systems development. *≪UML≫ 2002—The Unified Modeling Language*. Retrieved from http://link.springer.com/chapter/10.1007/3-540-45800-X_32

Kachurka, V. (2013). Design Patterns in N-tier Architecture. *XV International PhD Workshop*, (October). Retrieved from http://mechatronika.polsl.pl/owd/pdf2013/kachurka.pdf

Khan, R. A., & Mustafa, K. (2008). Secured Requirement Specification Framework (S RSF). *American Journal of Applied Sciences*, *5*(12), 1622–1629.

Khodyakov, D., Hempel, S., Rubenstein, L., Shekelle, P., Foy, R., Salem-Schatz, S., … Dalal, S. (2011). Conducting online expert panels: a feasibility and experimental replicability study. *BMC Medical Research Methodology*, *11*(1), 174. doi:10.1186/1471-2288-11-174

Khwaja, S., & Alshayeb, M. (2013). A framework for evaluating software design pattern specification languages. *2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS)*, 41–45. doi:10.1109/ICIS.2013.6607814

Klinkoff, P., Kirda, E., Kruegel, C., & Vigna, G. (2007). Extending .NET security to unmanaged code. *International Journal of Information Security*, *6*(6), 417–428. doi:10.1007/s10207-007-0031-0

Littig, B. (2008). Interviews mit Eliten–Interviews mit ExpertInnen: Gibt es Unterschiede. *Forum Qualitative Sozialforschung*, *9*(3). Retrieved from http://www.ihs.ac.at/publications/soc/misc/InterviewsEliten.pdf

Martin, B., Brown, M., Paller, A., Kirby, D., & Christey, S. (2011). 2011 CWE / SANS Top 25 Most Dangerous Software Errors. Retrieved from http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.pdf

Meier, J., Mackman, A., Vasireddy, S., Dunner, M., Escamilla, R., & Murukan, A. (2003). *Improving Web Application Security: Threats and Countermeasures* (p. 919). Microsoft Press. Retrieved from http://books.google.com/books?hl=en&lr=&id=Spti0mHhlsUC&oi=fnd&pg=PP2&dq=Improving+Web+Application+Security+Threats+and+Countermeasures&ots=KEfBrKEhQM&sig=mnrDoHKZH93NkQWktonnAw9greE

Microsoft. (2012). ASP.NET Partial Trust does not guarantee application isolation. *Microsoft Knowledge Base*. Retrieved March 16, 2014, from http://support.microsoft.com/kb/2698981

Miller, D. (2003). .NET from the Hacker's Perspective. In *Black Hat Windows Security 2003 Briefings & Training*. Seattle, WA. Retrieved from http://www.blackhat.com/presentations/win-usa-03/bh-win-03-miller/bh-win-03-miller.pdf

Miller, R. L. (2003). *The OSI Model: An Overview*. SANS Institute. Retrieved from http://www.sans.org/reading_room/whitepapers/standards/osi-model-overview_543

Mingers, J. (2001). Combining IS research methods: towards a pluralist methodology. *Information Systems Research*. Retrieved from http://infosys.highwire.org/content/12/3/240.short

Mouratidis, H., Giorgini, P., & Manson, G. (2005). When security meets software engineering: a case of modelling secure information systems. *Information Systems*, *30*(8), 609–629. doi:10.1016/j.is.2004.06.002

Nakahara, S. (2000). Electronic Notary System and its Certification Mechanism. *ECIS 2000 Proceedings*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.5051&rep=rep1&type=pdf

National Institute of Standards and Technology. (2011). *NIST Special Publication 800-53 Information Security* (3rd ed., Vol. 2009). Paramount, CA.: CreateSpace. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.7599&rep=rep1&type=pdf

Olivier, M. (2009). Information technology research: a practical guide for computer science and informatics, 184.

Paul, M. (2008a). Software Assurance: A Kaleidoscope of Perspectives. *The International Information Systems Security Certificate Consortium Inc (ISC)$^2$*. Retrieved from https://www.isc2.org/uploadedFiles/(ISC)2_Public_Content/Certification_Programs/CSSLP/CSSLP_WhitePaper_2-ONLINE(1).pdf

Paul, M. (2008b). The Ten Best Practices for Secure Software Development. *The International Information Systems Security Certification Consortium Inc (ISC)$^2$*. Retrieved April 28, 2013, from http://www.isc2.org/uploadedFiles/(ISC)2_Public_Content/Certification_Programs/CSSLP/ISC2_WPIV.pdf

Paul, M. (2011). Official (ISC)$^2$ Guide to the CSSLP. *Software Community (ISC)$^2$ Whitepapers*.

Peffers, K., Tuunanen, T., Rothenberger, M., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. doi:10.2753/MIS0742-1222240302

Pirnau, M. (2013). The analysis of the .NET architecture security system. *Electronics, Computers and Artificial Intelligence*, 1–6. doi:10.1109/ECAI.2013.6636185

Polydys, M. L., Ryan, D. J., & Ryan, J. J. C. H. (2006). Best Software Assurance Practices in Acquisition of Trusted Systems.

Posthumus, S. M., & von Solms, R. (2009). *A Model for Aligning Information Technology Strategic and Tactical Management*. Nelson Mandela Metropolitan University.

Saunders, M., Lewis, P., & Thornhill, A. (2003). *Research Methods For Business Students* (3rd ed., p. 504). London: Prentice Hall.

Siles, R. (2014). Session Management Cheat Sheet - OWASP. *OWASP Cheat Sheets*. Retrieved August 22, 2014, from https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

Slocum, N., Steyaert, S., & Lisoir, H. (2005). Participatory Methods Toolkit: A Practitioner's Manual - Expert Panel. *King Baudouin Foundation and Flemish Institute for …*. Retrieved from http://www.kbs-frb.be/uploadedFiles/KBS-FRB/Files/EN/PUB_1540_Toolkit_7_ExpertPanel.pdf

Smith, J. (2009). WPF Apps With The Model-View-ViewModel Design Pattern. *MSDN Magazine*. Retrieved from http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

Sodiya, A. S., Onashoga, S. A., & Ajayi, O. B. (2006). Towards Building Secure Software Systems.

Stoneburner, G., Hayden, C., & Feringa, A. (2001). Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A. Retrieved from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA393550

Swanson, M. (1996). Generally Accepted Principles and Practices for Securing Information Technology Systems.

Syromiatnikov, A., & Weyns, D. (2014). A Journey Through the Land of Model-View-* Design Patterns. *WICSA*. Retrieved from http://lnu.se/polopoly_fs/1.99215!Artem Syromiatnikov paper WICSA2014.pdf

Tull, D., & Hawkins, D. (1987). Marketing research: measurement and method: a text with cases. Retrieved from http://www.getcited.org/pub/101963251

Viega, J., & McGraw, G. (2006). *Building secure software*. Retrieved from http://books.google.com/books?hl=en&lr=&id=aYAj8W7z_sUC&oi=fnd&pg=PA1&dq=building+secure+software&ots=5cNl2iy37B&sig=T8j1es8gpAoiCaB78ED_1i_bhT0

Vom Brocke, J., & Buddendick, C. (2006). Reusable conceptual models–requirements based on the design science research paradigm. *Conference on Design Science Research*.

Wang, H. J., & Wu, H. (2009). Supporting Business Process Design through a Web 2.0 Process Repository, (0713290), 0–5.

Williams, J., Manico, J., & Mattatall, N. (2014). XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP. *OWASP Cheat Sheets*. Retrieved August 23, 2014, from https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Williams, J., & Wichers, D. (2013). OWASP Top 10. *OWASP Foundation*. Retrieved from http://wiki.owasp.org/images/1/17/OWASP_Top-10_2013--AppSec_EU_2013_-_Dave_Wichers.pdf

Yi, S., Naldurg, P., & Kravets, R. (2002). A security-aware routing protocol for wireless ad hoc networks. *Urbana*. Retrieved from http://www.cis.temple.edu/~wu/teaching/Spring 2013/secure6.pdf

Yoshioka, N., Washizaki, H., & Maruyama, K. (2008). A survey on security patterns. *Progress in Informatics*, (5), 35. doi:10.2201/NiiPi.2008.5.5

Zhang, C., & Budgen, D. (2012a). A survey of experienced user perceptions about software design patterns. *Information and Software Technology*. doi:10.1016/j.infsof.2012.11.003

Zhang, C., & Budgen, D. (2012b). What Do We Know about the Effectiveness of Software Design Patterns? *Software Engineering, IEEE Transactions …*, *38*(5), 1213–1231. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5975176

# Appendix A: ZAWWW Conference Paper

## Design patterns for secure software development: demonstrating security through the MVC pattern

MR Colesky
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Michael.Colesky@nmmu.ac.za

LA Futcher
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Lynn.Futcher@nmmu.ac.za

JF Van Niekerk
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Johan.VanNiekerk@nmmu.ac.za

**With the current interconnected state of our world, the security of every system is important. More often than not, however, security is treated independent to software. This overlooks a vital security principle; security should be grounded in the same early design and development stages as an application itself. By instead favouring immediate returns over long term benefits, the value of concrete security is commonly undermined. To remedy this, this paper seeks to explore the relaying of security into a fundamental building block of software; namely design patterns.**

**Keywords: Design patterns, information security, security principles, secure software, development software**

### 1. Introduction

A well-argued point in information security is its need for human education, training and awareness. Quite often, it is the human aspect which is the weakest link. While this factor cannot be stressed enough, the technical factors should not be ignored. It does not help if users are well trained to make use of applications securely if the application itself is not designed with security in mind (Hight 2005:1). If the underlying code is not inherently secured then the application's overall ability to resist security attacks falls flat.

The introduction of the .Net Framework, Python, Java and other Runtime Environments, however, has been able to reduce some security risks. Much of what these managed frameworks reduce is the need for repetitive tasks such as checks and exceptions (Howard & LeBlanc 2009:540–556). However, this does not guarantee secure coding.

A single overlooked weakness can be devastating. Without incorporating security at every point in an application’s lifecycle, the potential for oversight is substantial (Breu, Burger, Hafner & Popp 2004:1). Without ensuring that it is applied at the root, the coating of security will not effectively integrate with the software.

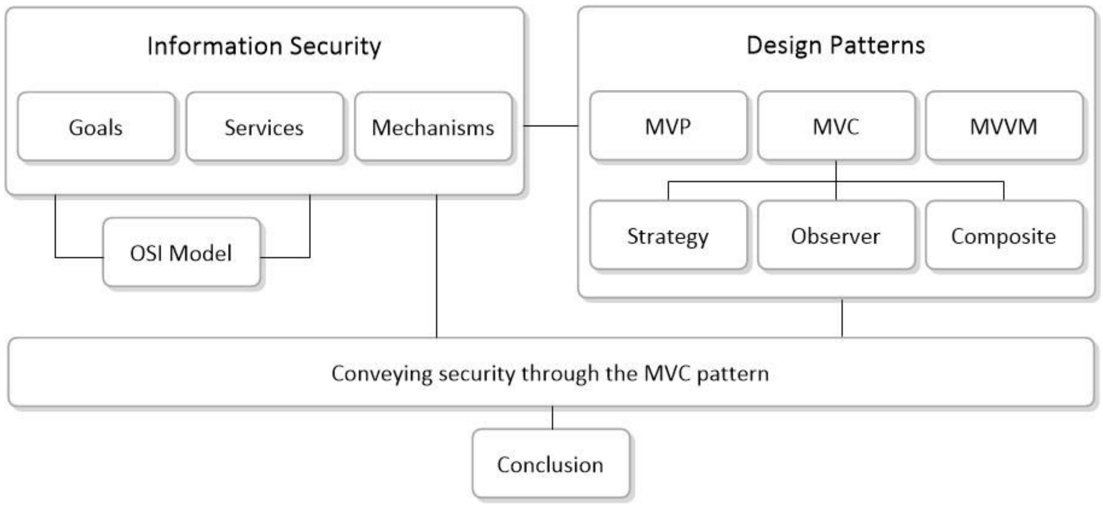
This falls back to the original statement. There is a need for human training, education and awareness; however, this applies to developers as well. Developers need to be aware of how easily an insecure application can be exploited, and how heavily such an exploit affects everything around it. Developers therefore need proper training in being able to secure their software (Stoneburner, Hayden & Feringa 2001:17). This is sometimes catered for, though often programmers gain their qualifications without any practical knowledge of secure software development.

One effective tool in conveying good design principles is the use of software design patterns (Freeman, Robson, Bates & Sierra 2004:9). In the study and practical application of these development blueprints, students begin to grasp not only the concepts of re-usable design, but also of good coding practice.

Effectively, the main aim is to make secure coding as natural as coding itself. If this can be done, then these ideas of how to design secure solutions will permeate throughout industry. Rather than having to incorporate security as a separate and time-consuming after-thought, security can be integrated in the same early design stages as the code itself.

This paper examines how design patterns can play a role in introducing security to the groundwork of applications by influencing coding habits of developers. Figure 1 shows the structure that this paper will follow.

**Figure 1: Paper outline**



First to be addressed is information security. This will funnel from the general into the specific, using network layering to guide from goals into lower level mechanisms. A discussion of design patterns follows. These are then combined in the relaying of secure concepts through patterns; the precursor to the conclusion.

## 2. Information security

Security is like a well-trained pugilist on defence. Adapted from Khan and Mustafa (2008:1622), ‘secure software’ provides solutions which block the vast majority of attacks, withstand most of what it cannot block, and recover quickly with minimal damage.

This notion of security being no impenetrable wall is one also acknowledged by Bishop and Engle (2006:15). In this, there is no silver bullet. A best effort must be made to cover all the bases. Breu et al (2004:1) suggest that this effort be handled in every stage of the development life cycle. This tactic is again given mention by Jones and Rastogi (2004:1).

In particular, emphasis is made on security being a specific requirement. However, it has been identified that in industry, security is often seen as a diversion from productivity (Jürjens 2002:2; Khan & Mustafa 2008:1622). Companies are all too eager to cut development time and costs, and quite often good security is ignored when not explicitly requested.

Instead of designing code securely, structurally from the inside out, programmers try to rely on instinct. This aspiration to successfully implement security as a habit is one commonly left unsatisfied. All too often developers are not given the training to use such intuition (Khan & Mustafa 2008:1622).

Furthermore, in a slideshow presentation given by Black Hat’s Drew Miller (2003) at a Windows security conference, it is shown how the introduction of managed code in the .Net Framework changes the landscape for application security. Notably, the risks of buffer overflows, code and role access security, as well as the Trojan-horse style virus are found to have been mitigated to a certain degree. Further examples of these include memory corruption and overflows in general, as mentioned in Mano Paul’s "The Ten Best Practices for Secure Software Development" whitepaper from the book: Official (ISC)$^2$ Guide to the CSSLP (2011).

To elaborate on this, many exploits are more difficult to produce when code is in a lowered trust state. With managed code, buffer overflows would have to be exploiting a weakness of the managed framework rather than a weakness of the application. It is when unsafe and unchecked constructs are used that the issue becomes a noticeable threat. In properly managed code, the introduction of self-resizing variables makes overflows less of a concern (Drew Miller 2003:5).

With the aforementioned lack of training, as well as with the popularity of managed coding languages already mitigating a few of the risks found traditionally, developers are less likely to pay specific attention to security. The following section begins with information security goals.

## 2.1. Information security goals

In information security, controls are implemented to protect the confidentiality, integrity and availability (CIA) requirements of a system and its information (National Institute Of Standards and Technology 2011:1). In the business world, this means creating an information security programme, which leads to the identification of risks, and the mitigation of those risks through technical controls and policies. These need to be re-evaluated continually. However, in the context of a software solution, a different approach is taken.

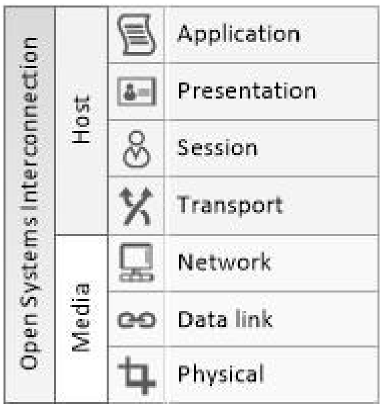
As previously mentioned, software development does not always cater for good security. When it does, however, according to Polydys, Ryan and Ryan (2006:56), it should address security at every stage of development. This is essential in maintaining an image of trustworthiness, and without security 'baked in', it is 'difficult or impossible' to provide a secure solution.

Polydys et al (2006:56) also mention predictability and conformance. When considering the information security goals we can map these to one another. A client needs to trust that their data will be secured; the software needs to follow strict information integrity policies to ensure that it is not modified without the required permissions and authorisation; and functionality should work whenever needed without misbehaving or disappearing. These goals are addressed through the information security services found in the Open Systems Interconnection (OSI) Reference Model as discussed in the following section.

## 2.2. The OSI model

The OSI model describes the communication between abstracted 'layers' of a system in communication networks. These layers can represent progressively higher-level forms of data as one goes up each layer as shown in Figure 2.

Figure 2: The OSI model (adapted from Miller 2003)



The bottom-most layer describes the physical communication media, signal and binary transmissions. The data link layer handles physical addressing, where above it, the network layer handles logical and port addressing. These make up the media section of the OSI model.

The transport layer handles connections as a whole, while the communicative session layer is commonly handled by web sites and services. The presentation and application layers are the most addressed in code. These handle actual data objects and functionality respectively. This is the host section of the OSI model.

As noted by Rachelle L. Miller (2003:7), the OSI model is not designed to enforce a structure. This paper's main focus is on the development related aspects derived from the security services of this model. These security services are discussed in section 2.3. together with security mechanisms in section 2.4.

## 2.3. Information security services

In securing Information Systems (IS), there exists a well-used framework which addresses how software should serve an organisation's security as well as the means to do so. The security services of the OSI Reference Model in ISO/IEC 7498-2 (1989) are often provided as a base for analysis of software security. These are:

1. Identification & Authentication
2. Access Control
3. Data Confidentiality
4. Data Integrity
5. Non-repudiation

These services are provided with greater depth. Firstly, identification & authentication refers to both data origin (that the peer is the source of the data) as well as the peer entity authentication (that an entity is who it claims to be).

Secondly, access control manages what resources are allowed to be allocated. Resources refer to not only the ability to read and write to devices, but also to execute a process or service. Hight (2005:2) mentions that a system's access control is largely affected by how well users play their part in security.

Thirdly, in data confidentiality, the protection of data from unauthorised disclosure, specific lower level services are provided (International Telecommunication Union 1991:11). There is firstly, the confidentiality of the traffic flow itself. If this is left open to inspection, conclusions may be derived from it. There is then the confidentiality of an entire connection, or of a single Service Data Unit (SDU). SDU refers to a single instance of sent data which has not yet been encapsulated. Within either of these, there lies selective field confidentiality – providing protection to only the specific fields that require it.

Fourthly, data integrity refers to the assurance that the data is unchanged. This can take place in the same full connection, connectionless and selective field options as data confidentiality. Data integrity may be checked for a full connection with or without the ability to recover from detected disparities. It may check for a 'connectionless' single SDU. One may elect to only check selective fields in either option.

Finally, there is non-repudiation. This refers to the inability to deny that a transaction took place (Alkussayer & Allen 2010:98). The service records transactions so that there is evidence of everything which transpires. This can take place in either or both of the recipient and sender receiving transactional evidence.

### 2.4. Information security mechanisms

As described by ISO/IEC 7498-2 (1989) and X.800 (International Telecommunication Union 1991:10–12), the following eight mechanisms may be utilised to achieve the security services:

1. Encipherment
2. Digital Signatures
3. Access Control
4. Data Integrity
5. Authentication Exchange
6. Traffic Padding
7. Routing Control
8. Notarization

Encipherment, or encryption, refers to the rendering of all data to be illegible. When reversible, encipherment may make use of either symmetric or asymmetric encryption. Irreversible encipherment however, need not make use of either if desired; this type of encryption can make use of hash functions to irreversibly transform information. In symmetric encryption, a single secret key may encrypt data as well as decrypt that same data. Asymmetric encryption, on the other hand, requires two keys: one to encrypt, and one to decrypt. The use of reversible encryption requires strong key management (Swanson 1996:51).

Digital signatures are identifiers attached to messages to uniquely differentiate between peers. This includes both the signing and verification of signatures. Encryption may also be used to facilitate signatures (Swanson 1996:57). Like asymmetric encryption, this makes use of both private and public keys. Verification makes use of the public key to deduce whether the message has been manipulated with the private key. Because the private key is only held by the sender, verification of the public key proves who sent the message.

Access control mechanisms refer specifically to the actual allowing and disallowing of access to resources. The relation can be understood by comparing access control to a private establishment's bouncer. If you do not have access rights, you will not be granted access. This metaphor can go further in describing access control lists, pre-authenticated passwords, status of the entity, time/route of access and duration without much elaboration.

Data integrity mechanisms check for equality between data before and after retrieval. This can be at block or stream level. Data integrity can be ensured through the use of checkvalues or block checks which may be encrypted to ensure no modification has been made. If a modification is identified, this may include automatic recovery, error-correction or retransmission.

The authentication process is handled by authentication exchange mechanisms. This utilises typical password usage, but can also make use of encryption and credentials as authentication indicators (Swanson 1996:43–45). Authentication may also involve other mechanisms such as with signatures and notaries. Of particular interest in this regard is that of unilateral (two-way) and mutual (three-way) authentication through handshakes. This can be used for encryption to protect against replaying an encrypted transmission.

Combined with encipherment or a similar confidentiality service, traffic padding mechanisms are used to obscure the patterns of network traffic. This prevents external analysis from deriving important weaknesses in the network. According to Jiang, Vaidya and Zhao (2003:1–2), traffic packets are spoofed to disrupt any analysis – this is referred to as a *cover mode*. In doing this, potential attackers come to false conclusions.

As indicated by Yi, Naldurg and Kravets (2002:1), routing controls may be implemented to bypass less trusted nodes in a network when secure information is concerned. The ability to dynamically, or through prearranged metrics, determine the most secure route through feedback is the basis of the routing control mechanism (International Telecommunication Union 1991:12).

As described by Nakahara (2000:1), "[a notarization] safely stores the evidence of events and actions during trading for a long term and subsequently certifies the fact by presenting formatted evidence (notary token)". The X.800 states much the same, adding that this evidence includes integrity, origin, time and destination. The notary can be viewed as a third party that is trusted by both persons, handling all transferred information. This communication can also make use of other non-repudiation mechanisms as well as encipherment.

So far this paper has examined security from the top down. In the following section, approaches to software design are described in the form of software design patterns.

## 3. Design patterns

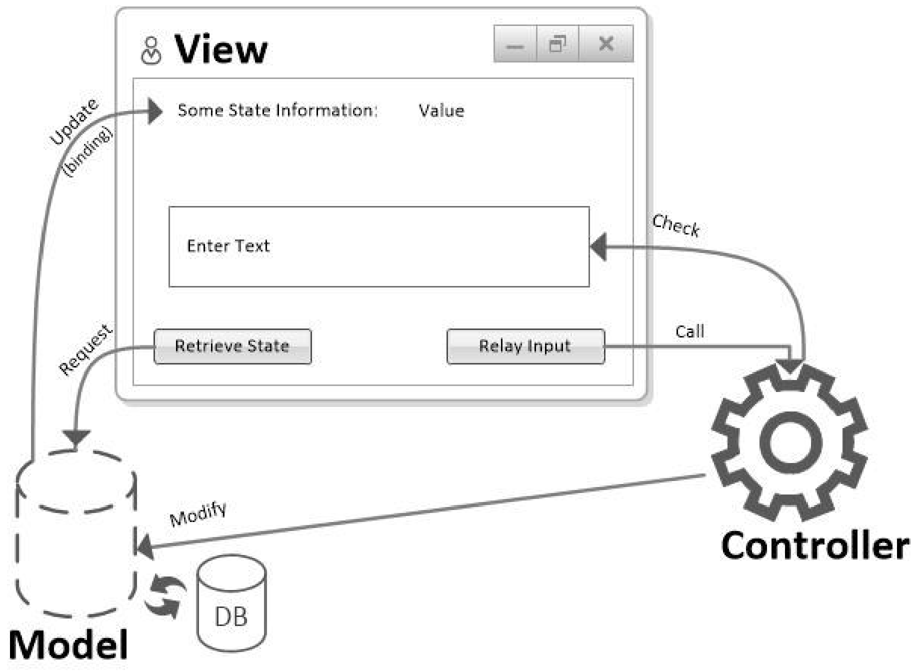
In software development, software design patterns are used to provide a guideline for the resolution of commonly occurring problems. These patterns typically make use of *class diagrams* or similar visual representations which represent the core of the generalised solution. This is the method of demonstration in use for this section, which focuses on areas around the Model-View-Controller (MVC) pattern.

### 3.1. Model-View-Controller

The MVC pattern focuses on separating application design into common developer roles: back-end components, User Interface (UI) design and interfacing with functionality between these (Smith 2009:2).

The view shows the windows, buttons, and other controls to the user; the controller interprets clicks and other commands; and the model does the business logic and object retrieval – then relaying the changes to the view again (Freeman et al. 2004:536–540). The view can also request state information from the model, and the controller can ask the view to update its display. This relationship is shown in Figure 3.

Figure 3: The Model-View-Controller (adapted from Freeman et al. 2004)



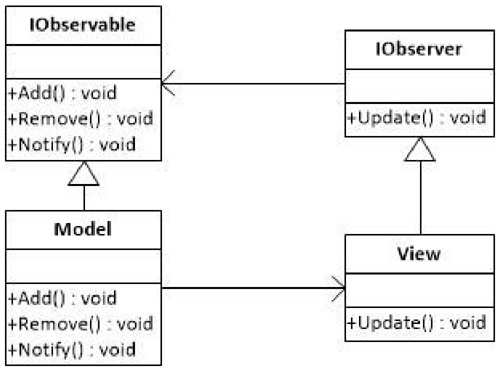
The Model-View-Controller features concepts found in other patterns (Freeman et al. 2004:541). These concepts are briefly explained below.

### 3.2. Observer

The view is an observer of the model. It registers to be updated when state changes. This registration is shown in Figure 3 on the button entitled ‘Retrieve State’. However, in reality, this happens behind the scenes. Using Figure 4, this can be understood in the following manner.

The view can access the model to add itself as an observer. When something changes, notify calls for all observers, in this case the view, to update their state. Observer has been identified as one of the most useful patterns (Zhang & Budgen 2012:7).

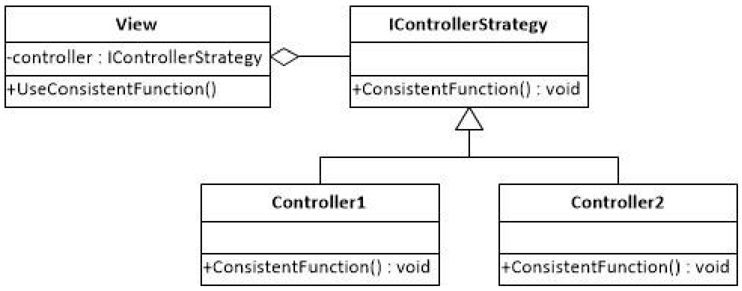
Figure 4: MVC perspective observer (adapted from Freeman et al. 2004)



### 3.3. Strategy

The view is also able to make use of a selection of controllers which implement its required functions. The ability to swop out these controllers works in the same way as the strategy pattern (Figure 5).

Figure 5: MVC perspective strategy (adapted from Freeman et al. 2004)

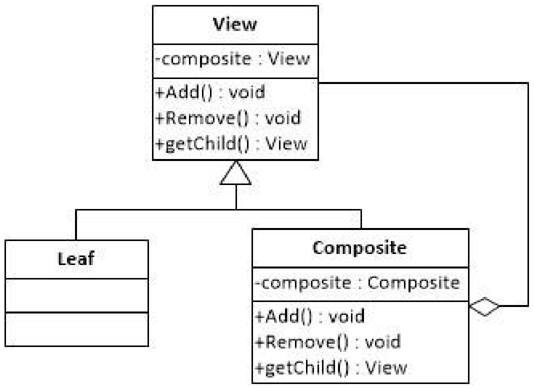

View
-controller : IControllerStrategy
+UseConsistentFunction()
IControllerStrategy
+ConsistentFunction() : void
Controller1
+ConsistentFunction() : void
Controller2
+ConsistentFunction() : void

This pattern describes keeping the structure of components consistent so that they can be used interchangeably (Freeman et al. 2004:24). The view is effectively able to use any controller without having to worry about specific implementation.

### 3.4. Composite

Finally the view is also able to compose of further views. A control can comprise of a set of other controls and each of their functions will be handled as if only one control existed. The ability to contain multiple objects and act interchangeably as one object or many is shown in the composite pattern (Figure 6).

Figure 6: MVC perspective composite (adapted from Freeman et al. 2004)



A view can compose of any number of components. Composite components can do the same. Leaf components do not. An understanding of recursion is needed to really grasp this pattern; however, some consider this pattern crucial (Zhang & Budgen 2012:12).

### 3.5. Model-View-Presenter

The MVC is an age-old pattern which has been revisited many times (Holmström 2011:18; Smith 2009:2). First the Model-View-Presenter (MVP) modified the concept of a controller to a presenter. The presenter handles interaction between the model and view; updating the view and reacting to user interaction.

The MVP splits into two flavours; the passive view where all view logic must be separately coded; and the supervising controller where the view is bound to the model via data-binding and the controller handles user interaction. The disadvantage of the supervising controller is the coupling between view and model – which is less favourable for testing.

### 3.6. Model-View-ViewModel

Abstracting the view of the MVC, the presentation model was the basis for the Model-View-ViewModel (MVVM). The MVVM directly binds view and viewmodel. The viewmodel encapsulates all view behaviour and this allows for potentially no controller. As such, instead of saying that the MVVM is a specialised version of the MVC, it is more correct to say that it is a specialisation of the presentation model towards Windows Presentation Foundation (WPF) (Smith 2009:2).

The MVVM is rarely used within web applications as it is tailored for WPF. Similarly, MVP is tailored to forms. However, the standard MVC is widely used in web development, especially for ASP.NET. As such the pattern selected to relay security concepts is the MVC pattern.

## 4. Conveying security through the MVC pattern

As stated in the beginning of this paper, security mechanisms will be used to structure the delivery of security concepts. The MVC pattern will convey them. The mechanisms most relevant are access control, authentication exchange and encipherment. However, for web development, notarization, data integrity and digital signage are also relevant. The remaining two are constrained by specific network applications.

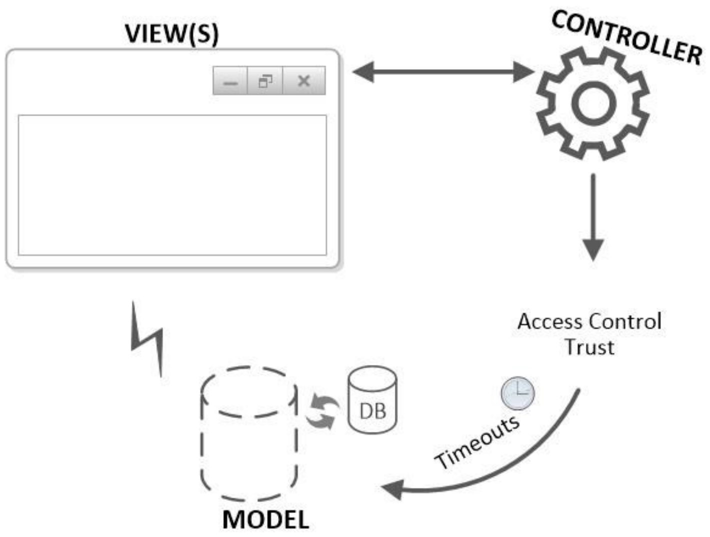
### 4.1. Access control

Access rights, privileges and trust are a very important aspect of software security. A substantial amount of the enhanced security from managed frameworks, specifically the .NET Framework, is based off of trust (Howard & LeBlanc 2009:536).

This embedded security makes the developer’s life easier. Though, this does not mean that a programmer should ignore security concepts which are handled by their choice of runtime, as its contributions are invalidated if the coder does not use them properly (Howard & LeBlanc 2009:535).

The principle of ‘least privilege’, for one, is not only to provide an entity with as little rights as possible, but also requires that access be permitted for the shortest duration possible. This aspect is depicted in Figure 7.

Figure 7: Access control using MVC (adapted from Freeman et al. 2004)



This goes further in allowing deliberate restriction and demanding of access when an application is strong-named. The idea of limiting privileges applies especially to web development because web functionality is based on partial trust. The .NET Framework restricts these calls by default, which protects against many online attacks (Howard & LeBlanc 2009:540–556).

### 4.2. Authentication exchange

The fifth most overlooked vulnerability according to the Common Weakness Enumeration (CWE) top 25 (Martin, Brown, Paller, Kirby & Christey 2011:3) is the ‘missing authentication for [a] critical function’. Following closely are ‘missing authorization’ and ‘hard-coded credentials’ at sixth and seventh place respectively.

The first refers to only guarding the ‘front door’ (Martin et al. 2011:14). As with access control, assess what areas require what role authentication for access. Otherwise, deny by default. When designing authentication measures, stick to using known solutions provided by the framework, and authenticate on the server rather than the client. For automatically denying access, turn to application firewalls.

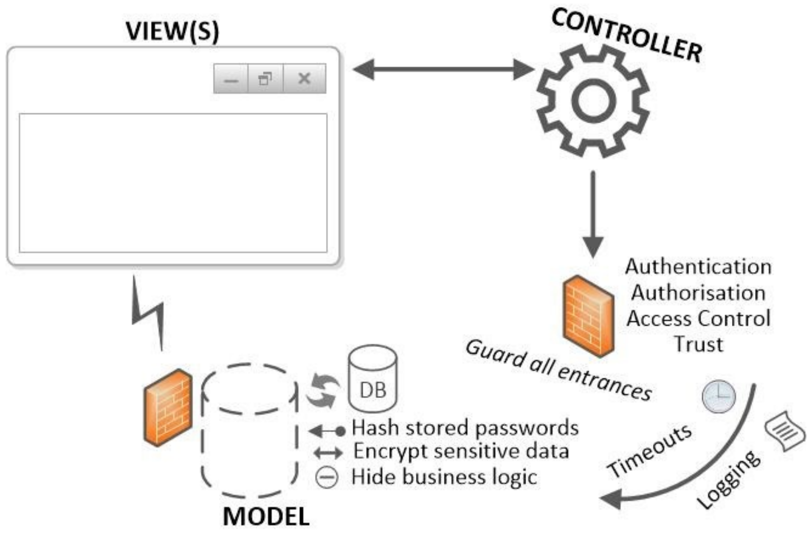
The caller must be able to provide evidence that it is who or what it says it is (Howard & LeBlanc 2009:109–118). Even so, its access may only be granted with certain levels of trust. Every request should be recorded, attempts should be restricted, sessions should timeout, and this should happen in a manner which does not permit abuse.

Though less severe than missing authentication, storing hard-coded access is forbidden; it provides another back door for the determined hacker. These aspects can be seen along with encipherment concepts in Figure 8.

### 4.3. Encipherment

The CWE top 25 lists ‘missing encryption of sensitive data’ as the eighth most common weakness (Martin et al. 2011:3). At nineteenth position is the use of a ‘broken or risky’ algorithm. Similarly, in Howard and LeBlanc’s Writing Secure Code (2009:259–269), mention is given to the unintentional use of a predictable random generator for encryption. In .NET for one, the Cryptography namespace should be used rather than any random function.

Figure 8: Access control, authentication and encipherment using MVC (adapted from Freeman et al. 2004)



Data held within the model should be kept secure. Where it is identified to be sensitive, the data should be encrypted with tried and tested algorithms. In the case of passwords, however, there is no need to have a reversible cipher. Store the hash of the password and then hash whatever entry requests comparison as their values will be the same. This and all other business logic should be kept confidential as well. These aspects are also reflected in Figure 8.

### 4.4. Notarization, data integrity and digital signage

While the remaining mechanisms can be approached separately, they do not all correlate to individual principles. Integrity is enforced by comparing checksums or hashes before and after transfer, though in terms of poor software security, this mechanism can also cover injections and overflows. Notarization and signage fall under non-repudiation, which is less relevant within the strict development context.

Non-repudiation is not something to ignore, however, and it should be accounted for in functions which specifically require it. As mentioned by Stoneburner, Hayden and Feringa (2001:24), securing an application does not require the use of all mechanisms.

### 4.5. ‘All input is evil’

As noted by Howard and LeBlanc (2009:341), “All input is evil”. The top four weaknesses defined by Bob Martin et al (2011:3) are SQL (Structured Query Language) injection, operating system command injection, buffer overflows and cross-site scripting respectively. These all correlate to malicious input.

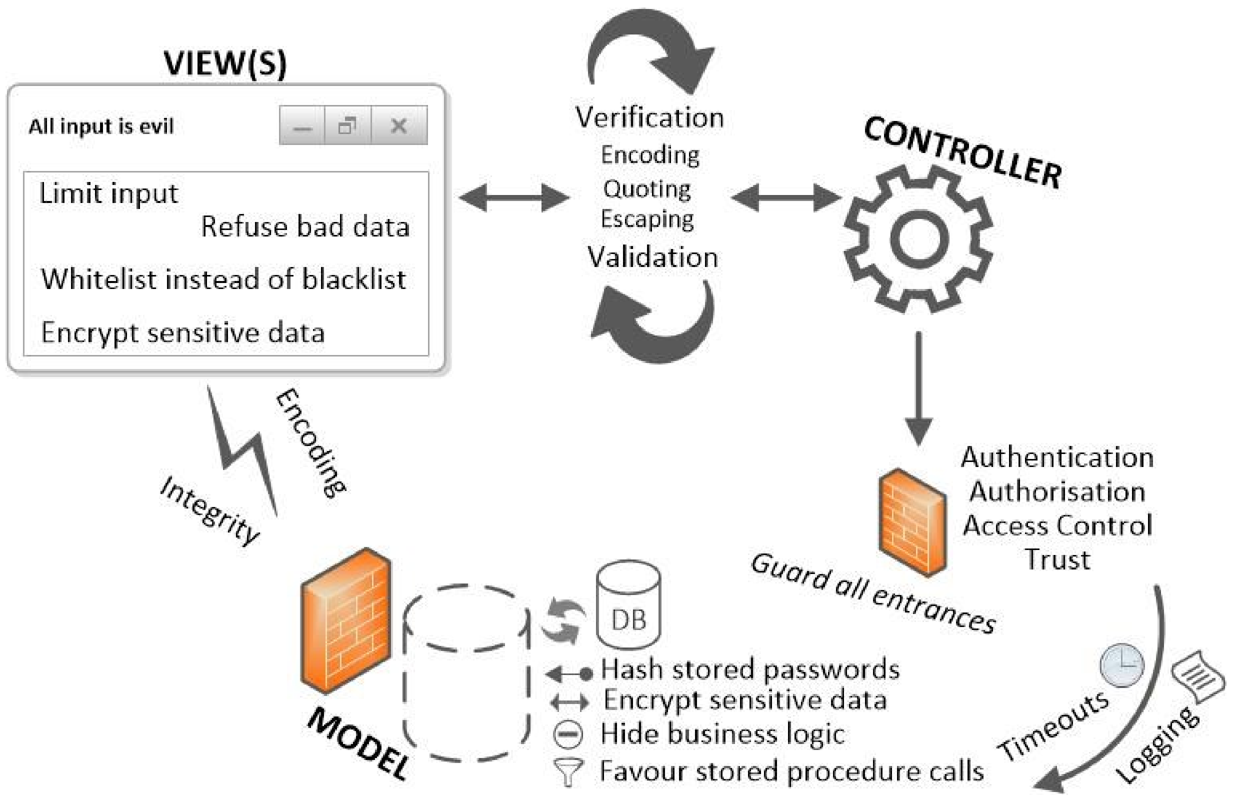
These threats can, however, be reduced (Martin et al. 2011:3–14). First and foremost, suspect all input as being malicious; favour drop-down lists over combo boxes; and reduce any and all freedom for expression where possible, rather whitelisting than blacklisting.

This includes escaping, encoding and quoting characters properly, using validation, and cleansing input from all sources, even hidden fields, cookies, and URIs (Uniform Resource Identifier). These tactics are commonly utilised by browser and website server software. This includes *HttpOnly* session cookies for Internet Explorer and FireFox as well as the *filter* option for Apache Struts (Martin et al. 2011:13).

In the interest of further protection from injection, make sure that all calls to the database are either stored procedures, previously prepared statements, or assigned only by parameter. Ensure that retrieved data is not modified during transmission. This is done by creating checkvalues which uniquely identify the data. Note that checkvalues should be encrypted, though, since a checkvalue may be recalculated.

The interaction between the view and controller needs to be closely checked against malicious input. This is best handled by limiting the scope for unexpected submissions. In validating and again ensuring that received information conforms to strict encoding, the controller can protect against these threats. The same applies to client side code. The controller should not assume that all script is legitimate.

Figure 9: Conveying security through MVC (adapted from Freeman et al. 2004)



Similarly, when the controller communicates with the model, rights to access should not be assumed. In all points of access there must be stringent procedures to verify that whatever calls a server-side business logic function has the authorisation, credentials and rights to do so.

Figure 9 depicts the correlation between some key security aspects and each component of the Model-View-Controller. The model shows how a pattern's representation can encourage emphasis on security in certain areas of an MVC web application.

In general, the use of structured frameworks which protect against traditional threats is recommended. In addition, every aspect adding to the difficulty to exploit a weakness is helpful.

## 5. Conclusion

This paper set out to determine how software design patterns can be utilised to convey security concepts and principles. In designing a model reminiscent of the MVC pattern's depiction, with additional software security placed strategically, the components of secure design may be observed.

This abstraction should allow web developers familiar with the pattern to grasp the security ideas embedded without the significant investment needed in reviewing these separately. The collaboration of these elements into one representation is useful in reviewing these concepts even if they are already known.

Akin to design patterns, these are there in simplified format for reference when needed; either conceptually or when in need of a design solution. Similarly, this illustration is by no means prescriptive. It is a generalisation to be consulted as a frame of reference; to be moulded into whichever form is most useful.

Particularly, individuals making use of design patterns in this way should be able to consider security from the beginning. As such, it is encouraged that further concepts be relayed through design patterns, especially with regard to security.

## 6. References

Alkussayer, A. and Allen, W.H. 2010. The ISDF Framework: Towards Secure Software Development. *Journal of Information Processing Systems* 6(1):91–106. Available http://koreascience.or.kr/journal/view.jsp?kj=E1JBB0&py=2010&vnc=v6n1&sp=91.

Bishop, M. and Engle, S. 2006. The Software Assurance CBK and University Curricula. :14–21.

Breu, R., Burger, K., Hafner, M. and Popp, G. 2004. Towards a Systematic Development of Secure Systems 2 Basic Concepts of an Object Oriented Software Process.

Freeman, E., Robson, E., Bates, B. and Sierra, K. 2004. *Head first design patterns*.

Hight, S.D. 2005. The importance of a security , education , training and awareness program ( November 2005 ). 27601(November):1–5.

Holmström, A. 2011. Performance and Usability Improvements for Massive Data Grids using Silverlight. . Available http://umu.diva-portal.org/smash/record.jsf?pid=diva2:414906 (accessed 11 May 2013).

Howard, M. and LeBlanc, D. 2009. *Writing secure code*.

International Standards Organization 1989. 7498-2. Information processing systems—Open Systems Interconnection—Basic Reference Model. Part 2: Security Architecture. *ISO Geneva, Switzerland*. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Information+processing+systems+--+Open+Systems+Interconnection+--+Basic+Reference+Model+--+Part+2:+Security+Architecture#0 (accessed 30 April 2013).

International Telecommunication Union 1991. X. 800 Security Architecture for Open Systems Interconnection for CCITT applications. *ITU-T (CCITT) Recommendation*. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:X.800+:+Security+architecture+for+Open+Systems+Interconnection+for+CCITT+applications#0 (accessed 30 April 2013).

Jiang, S., Vaidya, N. and Zhao, W. 2003. Energy consumption of traffic padding schemes in wireless ad hoc networks. *Real-time system security*. Available http://dl.acm.org/citation.cfm?id=903870 (accessed 1 May 2013).

Jones, R. and Rastogi, A. 2004. Secure coding: building security into the software development life cycle. *Information Systems Security*. Available http://www.tandfonline.com/doi/abs/10.1201/1086/44797.13.5.20041101/84907.5 (accessed 4 May 2013).

Jürjens, J. 2002. UMLsec: Extending UML for secure systems development. *≪UML≫ 2002—The Unified Modeling Language*. Available http://link.springer.com/chapter/10.1007/3-540-45800-X_32 (accessed 26 April 2013).

Khan, R.A. and Mustafa, K. 2008. Secured Requirement Specification Framework (S RSF). *American Journal of Applied Sciences* 5(12):1622–1629. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Secured+Requirement+Specification+Framework+(+S+RSF+)#0 (accessed 14 May 2013).

Martin, B., Brown, M., Paller, A., Kirby, D. and Christey, S. 2011. 2011 CWE / SANS Top 25 Most Dangerous Software Errors.

Miller, D. 2003. .NET from the Hacker's Perspective. In: *Black Hat Windows Security 2003 Briefings & Training*. Seattle, WA.

Miller, R.L. 2003. *The osi model: an overview*. SANS Institute.

Nakahara, S. 2000. Electronic Notary System and its Certification Mechanism. *ECIS 2000 Proceedings*. Available http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.5051&rep=rep1&type=pdf (accessed 1 May 2013).

National Institute Of Standards and Technology 2011. *Nist special publication 800-53 information security*. 3rd ed. Paramount, CA.: CreateSpace.

Paul, M. 2011. Official (ISC) 2 Guide to the CSSLP. *Software Community (ISC)$^2$ Whitepapers*. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Official(ISC)2+Guide+to+the+CSSLP#0 (accessed 5 May 2013).

Polydys, M.L., Ryan, D.J. and Ryan, J.J.C.H. 2006. Best Software Assurance Practices in Acquisition of Trusted Systems.

Smith, J. 2009. WPF apps with the model-view-ViewModel design pattern. *MSDN magazine*. Available http://www.techguyonline.com/wp-content/uploads/2010/08/wpf_mvvm.pdf (accessed 11 May 2013).

Stoneburner, G., Hayden, C. and Feringa, A. 2001. Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A. . Available http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA393550 (accessed 30 April 2013).

Swanson, M. 1996. Generally Accepted Principles and Practices for Securing Information Technology Systems. (September).

Yi, S., Naldurg, P. and Kravets, R. 2002. A security-aware routing protocol for wireless ad hoc networks. *Urbana*. Available http://www.cis.temple.edu/~wu/teaching/Spring 2013/secure6.pdf (accessed 1 May 2013).

Zhang, C. and Budgen, D. 2012. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*. Available http://linkinghub.elsevier.com/retrieve/pii/S0950584912002297 (accessed 5 March 2013).

## Appendix B1: Expert Review Email Template

Dear Expert,

Thank you for agreeing to take the time to review the Security-conscious Model-View-Controller (SecMVC) as part of my master's research under the supervision of Dr Futcher, Dr Gerber, and Prof van Niekerk. Your feedback will help validate as well as provide insight into improvements for the model.

Attached in this email are the documents which may be used to complete this expert review. These are as follows:

1. A short background document, which provides all the necessary information on the model, as well as its two parent focus areas (Software Design Patterns & Secure Software Development).
2. More detailed appendices, for further background information. This includes Appendix A, B, C, and references.
3. The Expert Review, in the form of a questionnaire document.

The attached Expert Review questionnaire document provides a means to acquire feedback on the SecMVC model, assessing it in terms of its usefulness, quality and efficacy. Some questions allow for open-ended responses, and comment is much appreciated. This is the only document that needs to be returned for collation of data. The process should take between 15 and 30 minutes.

Kindly attach the Expert Review questionnaire document to a reply email after completing it to your satisfaction. Please return this by no later than the 20th of September.

If you have any queries, please contact me on this email, or telephonically.

Kind regards,
Michael

NMMU Master's Student
michael.colesky@nmmu.ac.za
+27 72 525 5928

# Appendix B2: Expert Review Background

The purpose of this document is to provide some background on the focus areas pertaining to the model, as well as insight into and motivation for the model itself. This model is the Security-conscious Model-View-Controller (SecMVC).

The SecMVC is designed to convey secure software development principles and underlying concepts in a .NET context through a common and well-understood software design pattern; namely, the Model-View-Controller (MVC) pattern. It also serves to remind developers of the importance of security throughout an application, as each area in the MVC can be approached in a secure manner. This applies to web applications in particular, as they are exposed to a greater number of threats.

## 1. The Original Model-View-Controller (MVC) Software Design Pattern

Ampatzoglou, Frantzeskou, and Stamelos (2012, p. 1) describe software design patterns as 'documented solutions to common design problems that are expected to enhance software quality'. The pattern this model focuses on is the original MVC, which over the years has evolved in many different ways. As it is arguably the most familiar version of the pattern, it is most suitable for shifting the focus to security. The fundamentals of this pattern are depicted in Figure 1.

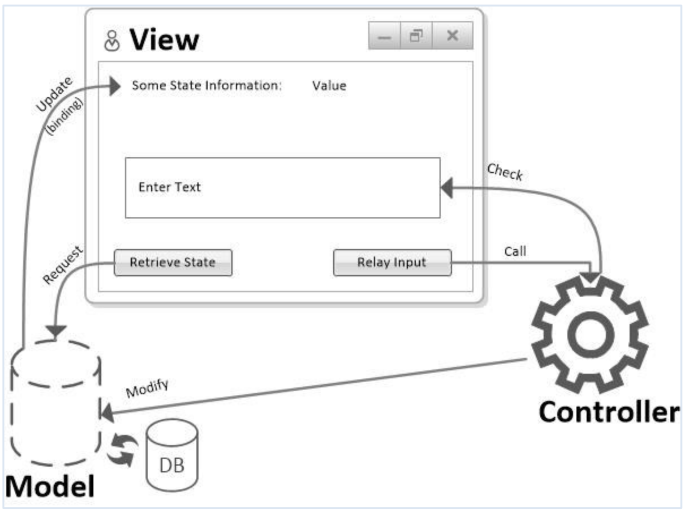


Figure 1: The Model-View-Controller (adapted from Freeman et al. 2004)

## 2. The Integrated General Secure Software Development Principles

The identification of secure concepts was achieved through the review of various sources of literature. Around these concepts, recurring software development principles were evident. These principles have remained relevant, and appear in different forms in the literature. One such .NET specific form is that of Meier, Mackman, Vasireddy, Dunner, Escamilla, & Murukan (2003, p. 11). Meier et al. define nine security principles for application developers to adhere to while designing solutions, as well as each component within them.  Based on these principles, this research derives four generalised principles. This is done as per the underlying descriptions of each principle. The comparison is shown in Table 1.

Table 1: Summary of Principle Adaptation (adapted from Meier et al. 2003)

| 9 Principles by Meier et al. | Correlation | General Principles |
|---|---|---|
| Reduce your attack surface | "…disabling or removing unused services, protocols, and functionality." | Least privilege |
| Compartmentalise | "Reduce the surface area of attack…Firewalls, least privileged accounts, and least privileged code are examples of compartmentalizing." | |
| Create secure defaults | "[Does] the default [use] least privilege? Is [it] disabled by default…?" | |
| Use least privilege | "…minimal privileges and access rights…" | |
| Do not trust user input | "Assume all input is malicious…" | All input is evil |
| Fail securely | "…do not leave sensitive data accessible." | Output securely |
| Check at the gate | "Authenticate and authorize callers early…" | Layer your defence |
| Secure the weakest link | "Any weak link in the chain is an opportunity for breached security." | |
| Apply defence in depth | "Defense in depth means you do not rely on a single layer of security." | |

As can be seen in Table 1, each principle is a part of the derived general principles by correlation to their underlying definitions. As such, these four general principles provide a compressed representation, and are considered ideal for use in the construction of the model. The motivation for the summarisation of these principles is shown in greater detail in Appendix A.

## 3. The Security-conscious MVC Model

The most recent iteration of the SecMVC is shown in Figure 2. Interactions between the View(s), Controller and Model are forfeited in favour of greater focus on application security. Instead of showing specifics, the model provides a generalisation towards secure ways of thinking when designing and implementing a solution. Principles and their underlying concepts, summarised in Appendix B, are integrated into the model by means of generalisation.

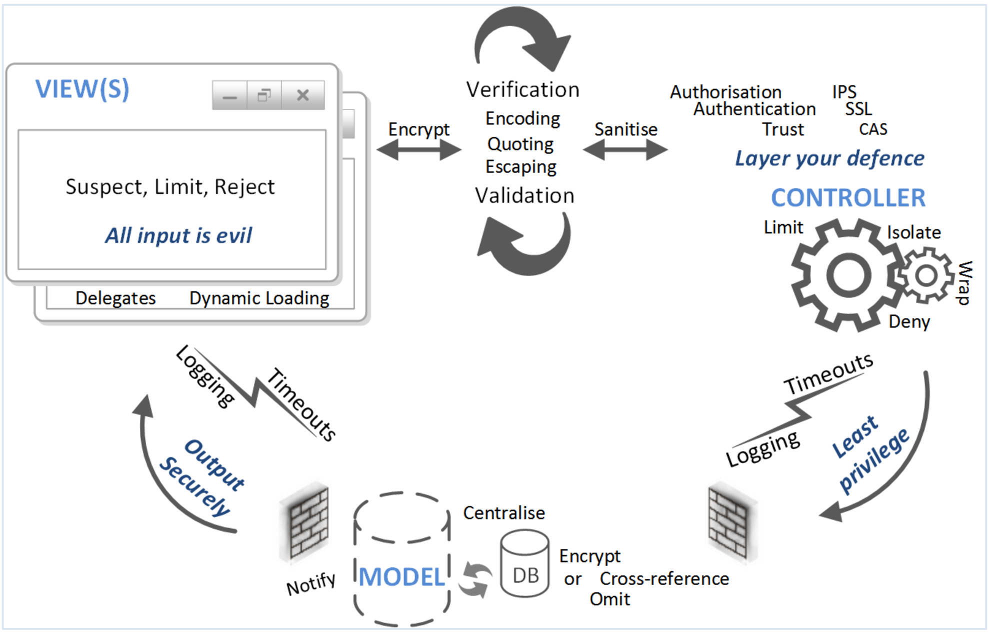


Figure 2: The Security-conscious MVC (SecMVC) model

The ways in which these generalisations can be derived into explicit implementations is shown in Appendix C. The examples within are conceived through the utilisation of top security risks and weaknesses from both the Common Weakness Enumeration (CWE) (Martin, Brown, Paller, Kirby, & Christey, 2011) and the Open Web Application Security Project (OWASP) (Williams & Wichers, 2013) rankings.

The top three weaknesses identified by the two rankings were Injection, Broken Authentication and Session Management, and Cross-Site Scripting (XSS). In general, the suggestions offered were specific implementations of the generalised principles and concepts reflected in the SecMVC and its underlying background. Commonly reoccurring elements in the suggested defences feature prominently in the SecMVC. The following mitigation strategies are supported in all three top weaknesses:

- Mistrust of all input data, from all possible sources, before stringent validation and encoding,
- Isolation and separation of untrusted elements from trusted ones and their interpretation,
- A focus on white-list input strategies, with known black-lists as a secondary defence,
- Lowest privilege for any functionality

These notions are taken into account by the SecMVC and its underlying general security principles and secure concepts. These concepts, as shown in the aforementioned Appendix B, can be seen in Figure 2 of this document. For example, the terms ‘suspect’, ‘validation’, and ‘encoding’ can be seen on the View(s) section and in-between the View(s) and Controller respectively.

# Appendix B3: Expert Review Appendix A

## Justification of General Security Principles

In their definition of 'reduce your attack surface', Meiers et al. (2003, p. 11) summarise "…disabling or removing unused services, protocols, and functionality." This correlates very closely to 'least privilege', which merely encourages the reduction of what is available to what is necessary.

Furthermore their description of 'compartmentalisation' features the question 'what resources can [an attacker] access?', which essentially aims to get the reader thinking about what can be retrieved from which section of an application, encouraging the limitation of scope. Two of the three examples of compartmentalisation are least privilege, and the third, 'firewalls' is a tool which can be (and should be) used to achieve it.

There is also 'create secure defaults', which encourages that defaults are as restrictive as possible. Their description encourages denying and disabling of access by default. This, too, seems to match the principle of least privilege very closely. However, their description also includes aspects one might expect in other principles entirely. 'When an error occurs' has the makings of an example one would expect to see under the 'fail securely' principle (Meier et al., 2003, p. 11).

With the aforementioned similarities between 'defaults', 'compartmentalisation' and 'attack surface' principles and the least privilege principle, this research proposes that they be grouped under one name to simplify the representation of knowledge into a smaller and therefore easier to remember format. The suggested term for this grouping is 'Least Privilege'.

A very important principle is not to trust user input. Inputs are an attacker's main point of entry when trying to exploit or search an application for vulnerabilities. This principle covers a large area, as this isn't just limited to the front end user interface. Non-exhaustively, other points of access include query strings, web service calls, using the application's libraries or ones which reference them with higher levels of access. As noted by Howard and LeBlanc (2009, p. 341), "All input is evil". Treating all input as though you know it will be malicious goes further than a mere mistrust. It also stresses the significance of this principle.

The next principle pertains less to exploitation and more to information gathering. 'Fail securely' covers hiding sensitive data after a failure in general, but focuses specifically on error messages (Meier et al., 2003, p. 11). By default, desktop applications show fully detailed exceptions, which while great for development is a serious security flaw if left that way. Web applications allow custom errors, however, these should not expose any more information than they need to.

This same principle can also apply to logging, especially since exceptions can be logged. Information that is logged should also be recorded securely. If discovered and unprotected, the information within could be used to find weaknesses – or the log could be modified to hide activity. Since not all logging is of failures, though all logs and exceptions are outputs, the principle can be broadened by naming it 'Output Securely'.

Three further related principles are ‘check at the gate’, ‘secure the weakest link’ and ‘apply defence in depth’. The first suggests authentication and authorisation should occur for each layer as soon as it can do so, so that no access is provided to an unauthorised (though potentially authenticated) user. The second refers to ensuring that each layer between network, host and application (whichever is weakest) is addressed, and then the process repeated. Adapted from Carlos Lyons, ‘a vulnerability in one will allow exploitation of the others’ (Meier et al., 2003, p. 6). The third, ‘apply defence in depth’, refers to gatekeepers at each layer – keeping in mind that these should account for each other being compromised.

Because each of these concepts are addressed with the multi-layered nature of in-production software in mind, unlike the more isolated principles, they could be addressed together. This research suggests that these be grouped under the title ‘Layer Your Defence’. Though this grouping works towards better simplicity and cohesion, it does not hide detail by fully replacing the principles.

# Appendix B4: Expert Review Appendix B

## Summary of Secure Concepts per General Principle

Table 0.1: Summary of Least Privilege Concepts

| Least Privilege |
|---|
| 1. Permit only what rights, access and functionality is necessary – for as little time as is necessary. |
| 2. Decide what trust is necessary using evidence of origin, identity and intent. |
| 3. Restrict and record attempts to achieve access. |
| 4. Isolate the components with varying trust requirements. |
| 5. Isolate assemblies, for web applications use separate application pools. |
| 6. Disable system services, protocols and ports which are not required for applications to function. |
| 7. Provide only what file system, registry, event and application log access is required. |
| 8. Provide only what read, write, create, modify or delete permissions are required. |
| 9. Limit what sensitive information is maintained in the file system, registry, or logs. |
| 10. Limit the readability of the necessary sensitive records through obfuscation and encryption. |
| 11. Regularly relocate logs to a remote inaccessible location for analysis and review. |
| 12. Do not leave code open to reverse engineering. |
| 13. Strong name your assembly for deliberate restriction. |
| 14. Use Code Access Security Policy for general restriction. |
| 15. Use Code Access Security in code for detailed restriction. |
| 16. Use Code Access Security at class or method level declaratively if it does not need to be dynamic. |
| 17. Avoid partial stack walks as these are susceptible to luring attacks. |
| 18. Avoid the use of 'deny' and 'permit only' as these may be bypassed. |
| 19. Avoid 'demand' unless required, if used isolating it and following the Demand/Assert pattern. |
| 20. Avoid security sensitive calls in threading which is susceptible to race conditions. |
| 21. Avoid inheriting unneeded security context in threads, for instance through impersonation. |
| 22. Wrap functionality with special security permissions, such as unmanaged code, in isolated assemblies. |
| 23. Ensure that whatever calls a critical function has the authorisation, credentials and rights to do so. |

Table 0.2: Summary of All Input Is Evil Concepts

| All Input Is Evil |
|---|
| 1. Suspect all input of being potentially malicious. |
| 2. Avoid open-ended input, instead providing a choice of defaults or a dynamically populated list. |
| 3. Permit only what characters could potentially be needed, rather than only filtering out invalid ones. |
| 4. Do not provide unlimited or unnecessarily large length bounds. |
| 5. Sanitise and validate input before processing it. |
| 6. Sanitise any input which might be tampered with, including hidden fields and cookies. |
| 7. Ensure that input is filtered, escaped, quoted or encoded where appropriate. |
| 8. Make use of any additional precautions offered by server and browser features. |
| 9. Both requests and responses should be encoded and sanitised. |
| 10. Do not rely on the integrity of client-side validation or script. |
| 11. Canonicalise filenames using Path.GetFullPath(). |
| 12. Interact with databases and other external storage in a predefined and strict manner. |
| 13. Ensure that transmitted data is not compromised by utilising encryption and encrypted checkvalues. |
| 14. Be wary of input through non-standard channels. |
| 15. Functions which accept generalised parameters, such as delegates, should account for maliciousness. |
| 16. Delegates should not be exposed unless the assembly is strong named without partially trusted callers. |
| 17. Serialised data should not contain sensitive information, and should not use partially trusted callers. |
| 18. Dynamic loading should avoid input from untrusted sources, unless using exceedingly strict validation. |

Table 0.3: Summary of Output Securely Concepts

| Output Securely |
|---|
| 1. Omit sensitive information and insecure details from all outputs of an application. |
| 2. Where sensitive information must be stored it may instead be cross-referenced, or encrypted. |
| 3. Access to each output should only read or write what it has to, if it has to, where it has to. |
| 4. Centralise all logging, providing only useful secured information. Do not log excessively. |
| 5. Log at multiple levels of detail for both easier analysis and better security for less verbose records. |
| 6. Logs should be regularly transferred to further secure locations, such as off site. |
| 7. Centralise, manage and handle exceptions, logging usable, but secure information. |
| 8. Notify the user of the failure, and any helpful information for the user, without implementation details. |
| 9. Do not allow exceptions to relay debug information to the user. Turn on custom errors in ASP.NET. |
| 10. Do not use exceptions in place of logical checks, or for expected errors – do not 'catch and release'. |
| 11. When using Visual Basic .NET filters, take into account that these may execute prior to the finally block. |

Table 0.4: Summary of Layer Your Defence Concepts

| Layer Your Defence |
|---|
| 1. All critical functions should have their callers authenticated and authorised. |
| 2. Determine what level of access should be required for each area in an application. |
| 3. Deny by default. Firewalls help achieve this. |
| 4. Routers should use restrictive footprinting responses. |
| 5. Make use of Secure Sockets Layer, Internet Protocol Security and a segmented network. |
| 6. Incoming external packets with internal Internet Protocol addresses should be denied. |
| 7. Outgoing packets with invalid internal Internet Protocol addresses should be denied. |
| 8. Make use of Intrusion Detection Systems to prevent Denial of Service attacks. |
| 9. Make use of the available Windows registry settings to adjust timeouts, backlogs and queues. |
| 10. Credentials, keys and authenticating functions should not be 'hard-coded' into an application. |
| 11. Encrypt sensitive data. |
| 12. Only use reversible ciphers for information which needs to be recovered. |
| 13. Authentication measures should make use of well-known tried and tested framework solutions. |
| 14. Authentication must take place at each entry point, as well as on the server as opposed to on the client. |
| 15. Ensure that any interaction with the host operating system is secured and isolated. |
| 16. Sign the application with a strong name – this allows checking for integrity and pointer manipulation. |
| 17. Seek full trust so that a strong name is possible, partial trust is no longer necessary for web applications. |

# Appendix B5: Expert Review Appendix C

## The SecMVC Model in Practice

In order to demonstrate the effectiveness which the SecMVC model would have in practice, some meaningful examples have been selected, incorporating three of the most notable security risks.

The most recent effort to date by the Open Web Application Security Project (OWASP) to rank prevalent security risks is the OWASP Top 10 of 2013 (Williams & Wichers, 2013, p. 6), which maintained 'injection', 'broken authentication and session management', and 'cross-site scripting' as the top three. This is compared to the Common Weakness Enumeration (CWE) (Martin, Brown, Paller, Kirby, & Christey, 2011, pp. 3–4); which ranked 'injection' weaknesses at both first and second place, the classic 'buffer overflow' at third, 'cross-site scripting' fourth, and 'missing authentication' fifth; the two rankings reinforce each other's conclusions.

In the delineation for this research, it is specified that a managed framework will be assumed when assessing security aspects. The chosen managed framework on which this research relies is Microsoft .NET. Therefore, classic 'buffer overflow' attacks are mitigated by internal checks. This sharpens the focus in terms of top weaknesses, or risks, to be reduced or mitigated as:

1. SQL, OS, and LDAP Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)

A developer who takes the SecMVC model into consideration, with an understanding of its elements and their relation to the underlying secure concepts and principles, should be less likely to create a solution with these weaknesses. The way in which such a developer would take note of these risks or weaknesses is demonstrated in the following discussed examples.

### Example 1: SQL, OS, and LDAP Injection

The OWASP Top 10 (Williams & Wichers, 2013, p. 7) has the following derived suggestions regarding ways to reduce injection:

- Consider all possible users and administrators as potential input for untrusted data;
- Almost any source of input can be an injection, including unconventional or internal inputs;
- Injection flaws come from interpreting untrusted data (separate this data from interpreters);
- Parameterise the input, and escape any special characters carefully; and
- White-list input rather than black-list, providing closed-ended options.

Noticeably, these suggestions coincide with the secure concepts outlined within the SecMVC's four general principles. The model encourages considering all input as potentially malicious, even from unexpected sources such as internally through delegates and dynamic loading. This is seen in the View(s) section of the model.

The Controller element encourages isolation, requiring trust before manipulating input. Between these the verification and validation process surrounds the need for adequate encoding, quoting and escaping. This canonicalisation exists for file inputs as well, which are also a noted Common Weakness Enumeration risk (Martin et al., 2011, pp. 3–4).

The CWE provides similar suggestions for SQL Injection in particular (Martin et al., 2011, pp. 7–8). It suggests the use of ‘prepared statements, parameterized queries, or stored procedures’ along with the use of ‘strong typing’. It advocates the separation of data and code, much like OWASP does between untrusted data and interpreters. Whitelisting, along with limited blacklisting are deemed useful, while ‘encoding, escaping, and quoting’ is described as the most effective solution. It also suggests lowest privilege, which is another of the SecMVC’s principles, as well as the use of isolation of privileges per specific functions.

The CWE’s suggestions for OS Injection include ‘sandboxing’, isolating processes from one another and the operating system (Martin et al., 2011, p. 9). Managed code, such as that in the .NET framework, is also noted to provide protection. Again, the use of escaping is shown to mitigate some of the risk involved.

## Example 2: Broken Authentication and Session Management

In the CWE’s suggestions for Missing Authentication for Critical Function, specific mention is given to only guarding the ‘front door’ (Martin et al., 2011, p. 14). It suggests using server-side checks, even for checks already done on the client-side. Again, the isolation of different areas of access within an application are recommended, and the centralisation of authentication procedures is encouraged. The use of tried and tested technology is shown to be important, with note given to the dangers of custom implementation.

OWASP provides a ‘cheat sheet’ with regard to session management (Siles, 2014). In this, advice is given regarding factors which developers can consider and influence. A non-exhaustive list which summarises this advice is shown below:

- Session ID names should be generic, not describing their technology as per default;
- Session IDs should be long enough to prevent brute force attacks;
- Session IDs must be unpredictably random, revealing no meaningful information;
- Session IDs must be strict, only originating from server generation, and are untrusted input;
- Session IDs must be renewed when the level of privilege changes, and after authentication;
- Sessions should have an idle and absolute timeout, and may use a renewal timeout;
- Sessions should allow users to manually terminate their sessions;
- Cookies are the preferred session mechanism, as they allow advanced token properties;
- The accepted session mechanisms should be limited to those chosen by design;
- Custom implementations are discouraged, though default settings may not be sufficient; and
- Both the session mechanism and the web session should make use of secure technologies.

Considering these suggestions, a developer adhering to the SecMVC model would already avoid ‘only guarding the front door’. Mindful of all forms of input being potentially malicious, the developer would not expect client-side checked input to be trusted data.

The developer would 'isolate' different areas of access, and still centralise the implementation of error handling, logging, and authentication. Known solutions using secure technologies would further promote the security of the application.

Using the SecMVC, the developer would apply the principles of 'least privilege' and 'all input is evil', as well as their concepts to session management. With limited representation of sensitive information taken into account including isolation and a need for trust, the construction of both session IDs and session ID names would closely follow the suggestions outlined in the OWASP 'cheat sheet'.

As the developer would prefer a higher degree of control over the session mechanism, if possible, session cookies would also be used. When privilege level changes, including post authentication, the developer would renew the cookie. This also provides a means against XSS attacks. In the context of the .NET Framework, ASP .NET function calls could be used. This includes the "Session.Abandon()" and "Response.Cookies.Add()" methods.

In these session cookies, the developer would use 'Secure' and 'HTTPOnly' attributes to prevent Man-in-the-Middle and XSS attacks respectively. 'Secure' ensures that cookies are only transported over secure channels, and 'HTTPOnly' prevents client-side scripting from retrieving the cookie. This is due to the need for SSL usage, as well as a strong mistrust for client-side script in terms of the session cookie, which is a sensitive form of information.

The developer would also be restrictive on the 'Domain' and 'Path' attributes of the cookie, using the default domain as the origin server (not setting the attribute). This adheres to the notion to 'limit' where the cookie may be sent, as well as to 'isolate' session IDs from other entities on the host.

In terms of 'timeouts', the 'Expire' or 'Max-Age' attributes would not be set, so as to avoid persistent cookies when security is a higher concern than convenience. However, session idle and absolute timeouts would need to be set so as to prevent open browser windows from continuing the session over dangerous periods of time. Should an attacker hijack the session ID, a renewal timeout implementation could additionally limit the attacker's time to cause damage. The OWASP 'cheat sheet' recommends between two and five minutes idle timeout for secure applications.

## Example 3: Cross-Site Scripting

The most prominent form of prevention against cross-site scripting, according to the OWASP Top 10 (Williams & Wichers, 2013, pp. 9–10), is the 'separation of untrusted data from active browser content'. That is, once again the isolation of untrusted data from interpretation. Another repeated suggestion is the consideration of all forms of input for their potential to cause damage. Further encouragement is given to escaping, based on HTML context. The same applies to whitelist input validation, although it is noted that the practice does not guarantee protection. Additional considerations are listed below:

- All untrusted input must be escaped, validated and verified as safe before processing;
- If AJAX is in use, safe client-side scripting APIs are recommended;
- If unsafe APIs are in use, strict validation and encoding are recommended; and
- While automated tools can assist detection, favour code review and penetration testing.

OWASP also provides a ‘cheat sheet’ for the prevention of cross-site scripting (Williams, Manico, & Mattatall, 2014). In this, various known XSS attack vectors, specifications and manual testing have been used to provide rules towards the prevention of XSS attacks. These rules are summarised in the following list:

- In any web page, untrusted input must be placed in specific, isolated and consistent slots;
- The escaped syntax must be used for non-alphanumeric characters in any of these slots;
- HTML escape untrusted data in content, specifically for: &, <, >, “, ‘, and /;
- HTML escape untrusted data in JSON values;
- Attribute escape untrusted data in common HTML attributes;
- Script escape untrusted data values in script if needed, untrusted data in script is dangerous;
- CSS escape and strictly validate untrusted data before using it in style properties;
- URL escape untrusted data in query strings;
- As with session management, use specialised sanitisation libraries, and HTTPOnly in cookies; and
- Consider the use of a Content Security Policy.

In the .NET context, Microsoft provides the AntiXSS library (Dorrans, 2014). In ASP.NET specifically, there also exists the “ValidateRequest()” method, which assists in sanitising requests for XSS (Williams et al., 2014). As the developer would be suspicious of all forms of input, these features or similar would be used to verify the safety of the data.

One of the ways in which the developer would limit the risks of XSS is through proper organisation of untrusted data, avoiding the use of this data in the areas depicted in Figure 1. Notably, contained script elements, such as those in ‘onClick’ attributes and the like, should avoid the use of untrusted data as well.

```
<script>HERE</script>   --clientscript
<!--HERE-->             --comments
<div HERE="e.g."/>      --attribute name
<HERE href="e.g."/>     --tag name
<style>HERE</style>     --stylesheets
```

Figure 1: Where Not To Put Untrusted Data (adapted from Williams et al., 2014)

As the SecMVC model advocates the use of encoding, quoting, and escaping, the developer would take steps to canonicalise all forms of input. Different elements have different encoding requirements, however. The different elements which can hold untrusted data, as such being at risk to XSS attacks, and their escaping strategies are shown in Table 1.

Table 1: Escape Strategies (adapted from Williams et al., 2014)

| HTML Escaping Conversions | & | < | > | “ | ‘ | / |
|---|---|---|---|---|---|---|
| | &amp; | &lt; | &gt; | &quot; | &#x27; | &#x2F; |
| Attribute Escaping | All non- alphanumeric characters with HTML &#xHH; format | | | | | |
| URL Escaping | Standard percent encoding for parameter values in %HH format | | | | | |
| Script Escaping | All non-alphanumeric characters with Unicode \uXXXX format | | | | | |
| CSS Escaping | All non-alphanumeric characters with \XX or \XXXXXX format | | | | | |

# Appendix B6: Expert Review References

Ampatzoglou, A., Frantzeskou, G., & Stamelos, I. (2012). A methodology to assess the impact of design patterns on software quality. *Information and Software Technology*, *54*(4), 331–346. doi:10.1016/j.infsof.2011.10.006

Dorrans, B. (2014). Microsoft Web Protection Library - Home. *CodePlex*. Retrieved August 23, 2014, from http://wpl.codeplex.com/

Howard, M., & LeBlanc, D. (2009). *Writing secure code*. Retrieved from http://books.google.com/books?hl=en&lr=&id=sjaweoe5-sYC&oi=fnd&pg=PT1&dq=Writing+Secure+Code&ots=3QgoiBPMyn&sig=zQjwoVnk0fqaOW5u7OHC9Xt_KwM

Martin, B., Brown, M., Paller, A., Kirby, D., & Christey, S. (2011). 2011 CWE / SANS Top 25 Most Dangerous Software Errors. Retrieved from http://cwe.mitre.org/top25/

Meier, J., Mackman, A., Vasireddy, S., Dunner, M., Escamilla, R., & Murukan, A. (2003). *Improving Web Application Security: Threats and Countermeasures* (p. 919). Microsoft Press. Retrieved from http://books.google.com/books?hl=en&lr=&id=Spti0mHhlsUC&oi=fnd&pg=PP2&dq=Improving+Web+Application+Security+Threats+and+Countermeasures&ots=KEfBrKEhQM&sig=mnrDoHKZH93NkQWktonnAw9greE

Siles, R. (2014). Session Management Cheat Sheet - OWASP. *OWASP Cheat Sheets*. Retrieved August 22, 2014, from https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

Williams, J., Manico, J., & Mattatall, N. (2014). XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP. *OWASP Cheat Sheets*. Retrieved August 23, 2014, from https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Williams, J., & Wichers, D. (2013). OWASP Top 10. *OWASP Foundation*. Retrieved from http://wiki.owasp.org/images/1/17/OWASP_Top-10_2013--AppSec_EU_2013_-_Dave_Wichers.pdf

# Appendix B7: Expert Review of the SecMVC Model

This document is provided in a questionnaire format in order to ascertain expert perspectives and to acquire feedback on the utility, quality, and efficacy of the SecMVC model depicted in Figure 2 of the background document. Some of these items are statements, for which a scale is used to determine the extent to which you agree. These statements need to be rated on a scale from 1 (totally disagree) to 4 (totally agree).

| **Expert Reviewer Information** | | | | |
|---|---|---|---|---|
| 1. Please provide the following details | | | | |
| Full Name and Title: | 1 | 2 | 3 | 4 |
| Confidence level in Software Design Patterns: (1=low; 4=high) | ☐ | ☐ | ☐ | ☐ |
| Confidence level in Secure Software Development: (1=low; 4=high) | ☐ | ☐ | ☐ | ☐ |
| Other relevant Fields of Expertise: | | | | |
| Any further personal information you feel is relevant to this review process: | | | | |

| **Utility of the SecMVC Model** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 2. The background content is sufficient for understanding the proposed model. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |
| | 1 | 2 | 3 | 4 |
| 3. The model can be recalled to memory in sufficient detail after establishing familiarity. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |
| | 1 | 2 | 3 | 4 |
| 4. The model can easily be understood and utilised. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |

| | | | | |
|---|---|---|---|---|
| **Quality of the SecMVC Model** | 1 | 2 | 3 | 4 |
| 5. The principles and concepts from Appendix B are represented accurately in the model. | ☐ | ☐ | ☐ | ☐ |
| 6. Are there secure concepts which have been left out, despite their importance? | No | ☐ | Yes | ☐ |
| If yes, please indicate which concepts. | | | | |
| 7. Does the generalisation of secure concepts degrade their value in the model? | No | ☐ | Yes | ☐ |
| If yes, please indicate why. | | | | |
| | 1 | 2 | 3 | 4 |
| 8. The model is of adequate quality with respect to security. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |

| | | | | |
|---|---|---|---|---|
| **Efficacy of the SecMVC Model** | 1 | 2 | 3 | 4 |
| 9. A secure implementation can be derived from the model's generalisations. | ☐ | ☐ | ☐ | ☐ |
| 10. The model would improve the likelihood of designing a secure solution. | ☐ | ☐ | ☐ | ☐ |
| 11. The model would improve the likelihood of implementing a secure solution. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on statements: 9; 10; and 11. | | | | |
| | 1 | 2 | 3 | 4 |
| 12. The model is effective in a security context. | ☐ | ☐ | ☐ | ☐ |
| Please provide any further comment you have on this. | | | | |
| 13. Familiarity with this model will make a developer more likely to consider security. | No | ☐ | Yes | ☐ |
| If no, please indicate why. | | | | |

| | |
|---|---|
| **Overall Impression**<br>14. What is your overall impression of the SecMVC model?<br><br>15. Please provide any suggested improvements to the SecMVC model.<br><br>16. Please provide any further comments, criticism or suggestions. | |

Thank you for your input regarding the SecMVC model. Your contribution will provide valuable insight into its value as an output for this research, as well as allow the model to be further improved upon. After completion of this questionnaire, kindly email it through as an attachment.

This feedback is required by the 20th of September.

Emails may be directed to: michael.colesky@nmmu.ac.za.

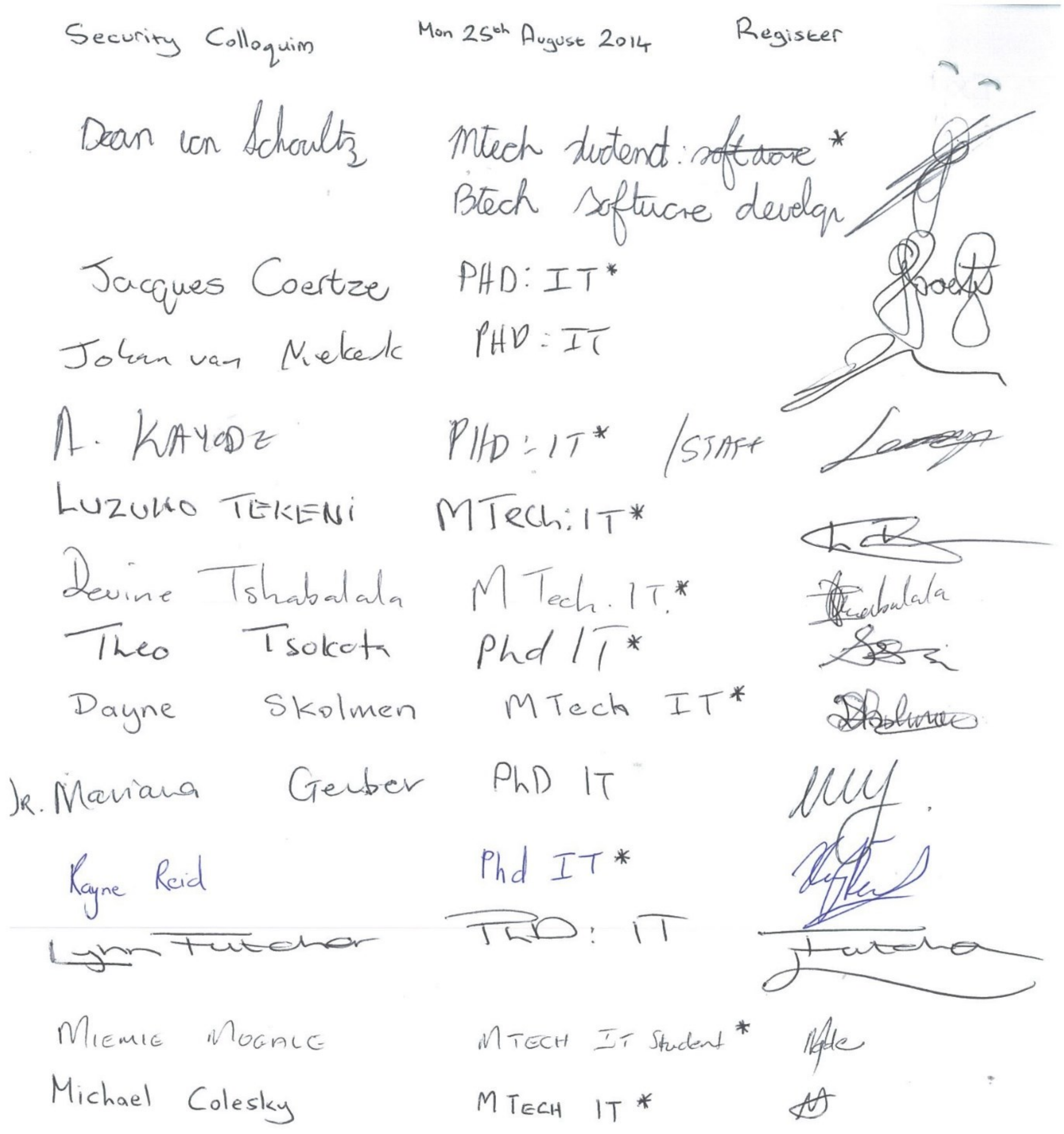
# Appendix B8: Original Expert Review Questionnaire

This section of the document is provided in a questionnaire format in order to ascertain expert perspectives and to acquire feedback on various aspects of the model. If for any question a definite answer cannot be reached, or if an answer is not applicable, kindly leave the checkboxes both unchecked, or the textboxes empty.

| | Yes | No |
|---|---|---|
| Has enough content been provided to sufficiently understand the proposed model? | ☐ | ☐ |
| Can the model be interpreted subsequently without the supporting content? | ☐ | ☐ |
| If so, can the same be said after a substantial duration? | ☐ | ☐ |
| Can the model be recalled to memory in sufficient detail after establishing familiarity? | ☐ | ☐ |
| Does the model accurately represent the secure principles and concepts mentioned? | ☐ | ☐ |
| Are there secure concepts which have been left out, despite sufficient relevance? | ☐ | ☐ |
| Does the generalisation of secure concepts degrade their value in this instance? | ☐ | ☐ |
| If so, please indicate which concepts are affected. | | |
| Can specific secure implementations be derived from these generalisations? | ☐ | ☐ |
| Can the model directly or indirectly contribute to a developer's concern for security? | ☐ | ☐ |
| Would the model positively affect a developer's ability to design a solution securely? | ☐ | ☐ |
| Would the model positively affect a developer's ability to implement a solution securely? | ☐ | ☐ |
| With the above considerations taken into account, should this model be deemed useful? | ☐ | ☐ |
| Please indicate an overall impression of the model. | | |
| Please provide any further comments, criticism or suggestions. | | |

## Appendix B9: Colloquium Peer Review Participants

A colloquium was held by the information security post graduate group in the School of Information Communication Technology in the Nelson Mandela Metropolitan University to review the proposed expert review. The participants of this review signed a register displayed below. The original expert review questionnaire is depicted in Appendix B8.



Security Colloquium Mon 25th August 2014 Register

| | | |
|---|---|---|
| Dean von Schoultz | Mtech student: ~~software~~ * Btech Software develop | [illegible] |
| Jacques Coertze | PHD: IT* | [illegible] |
| Johan van Niekerk | PHD: IT | [illegible] |
| A. KAYODE | PHD: IT* /STAFF | [illegible] |
| LUZUKO TEKENI | MTech: IT* | [illegible] |
| Devine Tshabalala | M Tech. IT.* | [illegible] |
| Theo Tsokota | Phd IT* | [illegible] |
| Dayne Skolmen | MTech IT* | [illegible] |
| Dr. Mariana Gerber | PhD IT | [illegible] |
| Kayne Reid | Phd IT* | [illegible] |
| Lynn Futcher | PhD: IT | [illegible] |
| Miemie Mogale | MTECH IT Student* | [illegible] |
| Michael Colesky | M TECH IT * | [illegible] |

# Appendix C1: First Expert Review of the SecMVC Model

**Expert Reviewer Information**

1. Please provide the following details

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Confidence level in Software Design Patterns: (1=low; 4=high) | ☐ | ☐ | ☐ | ☒ |
| Confidence level in Secure Software Development: (1=low; 4=high) | ☒ | ☐ | ☐ | ☐ |

Other relevant Fields of Expertise:

Patterns and guidelines as design aids; Multimedia Computing, Advanced Algorithms and Data Structures, Computer Archtecture, etc.

Any further personal information you feel is relevant to this review process:

Although I am not an InfoSec expert, I am a very fast learner and do my homework thoroughly.

**Utility of the SecMVC Model**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 2. The background content is sufficient for understanding the proposed model. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on this.

I feel that you need to dig deeper than Ampatzoglou et al. for a definition of software design patterns. Software design patterns are a formal way of documenting general reusable solutions to commonly occurring software design problems. So the MVC pattern is an architectural software design pattern intended to promote modularity in order to limit undesirable interactions. So in a sense, repurposing MVC as an aide-memoire for software security is at varience with the software pattern concept. I would also ask why you are using the MVC, and not another common software architectural pattern? (I suppose that one could argue that MVC is a very commonly used pattern.)

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 3. The model can be recalled to memory in sufficient detail after establishing familiarity. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on this.

It is quite easy to recall the top level issues, but not the details. (And the details are important!)

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 4. The model can easily be understood and utilised. | ☐ | ☒ | ☐ | ☐ |

Please provide any further comment you have on this.

The devil is in the details, so the model by itself is too abstract and limited. It needs all the detailed sub-models. I would suggest that you need a software security pattern language (not just one pattern), that incorporates sub-patterns for the detailed actions required to promote software security. This pattern language has to solve a software security problem (or problems) and should be structured in a way that solves this problem, and not the MVC problem of separation of concerns through modular design.

| | |
|---|---|
| **Quality of the SecMVC Model** | 1 2 3 4 |
| 5. The principles and concepts from Appendix B are represented accurately in the model. | ☐ ☐ ☒ ☐ |
| 6. Are there secure concepts which have been left out, despite their importance? | No ☒ Yes ☐ |
| If yes, please indicate which concepts. | |
| The principles and concepts are not really fully represented in the SecMVC model, as it is an abstraction. | |
| 7. Does the generalisation of secure concepts degrade their value in the model? | No ☒ Yes ☐ |
| If yes, please indicate why. | |
| Not applicable. | 1 2 3 4 |
| 8. The model is of adequate quality with respect to security. | ☐ ☐ ☒ ☐ |
| Please provide any further comment you have on this. | |
| The SecMVC model, together with the detailsed explanatory information, is of adequate quality with respect to security. | |
| **Efficacy of the SecMVC Model** | 1 2 3 4 |
| 9. A secure implementation can be derived from the model's generalisations. | ☐ ☐ ☒ ☐ |
| 10. The model would improve the likelihood of designing a secure solution. | ☐ ☐ ☒ ☐ |
| 11. The model would improve the likelihood of implementing a secure solution. | ☐ ☐ ☒ ☐ |
| Please provide any further comment you have on statements: 9; 10; and 11. | |
| The model must be taken together with the detailed explanatory information. | 1 2 3 4 |
| 12. The model is effective in a security context. | ☐ ☐ ☒ ☐ |
| Please provide any further comment you have on this. | |
| The model must be taken together with the detailed explanatory information. | |
| 13. Familiarity with this model will make a developer more likely to consider security. | No ☐ Yes ☒ |
| If no, please indicate why. | |
| Not applicable. | |
| **Overall Impression** | |
| 14. What is your overall impression of the SecMVC model? | |
| I think that it is a good start in pulling together a collection of software security issues and associated solutions in a way that provides a structured "one-stop-shop". | |
| 15. Please provide any suggested improvements to the SecMVC model. | |
| I think that the SecMVC model needs to be expanded into a software security pattern language for a particular software genre (websites?). | |
| 16. Please provide any further comments, criticism or suggestions. | |
| I am very happy to provide more asssitance if required by and acceptable to the parties concerned. | |

# Appendix C2: Second Expert Review of the SecMVC Model

**Expert Reviewer Information**

1. Please provide the following details — 1 2 3 4

   Confidence level in Software Design Patterns: (1=low; 4=high) — 1 [✓ 2] 3 4

   Confidence level in Secure Software Development: (1=low; 4=high) — 1 [✓ 2] 3 4

   Other relevant Fields of Expertise:

   Security protocols, privacy enhancing technologies, privacy by design

   Any further personal information you feel is relevant to this review process:

**Utility of the SecMVC Model** — 1 2 3 4

2. The background content is sufficient for understanding the proposed model. — 1 [✓ 2] 3 4

   Please provide any further comment you have on this.

   The _purpose_ of the model is not entirely clear; is it for educational purposes, or for actual design purposes

3. The model can be recalled to memory in sufficient detail after establishing familiarity. — 1 2 [✓ 3] 4

   Please provide any further comment you have on this.

4. The model can easily be understood and utilised. — 1 2 [✓ 3] 4

   Please provide any further comment you have on this.

   I'm not sure about utilisation. If the purpose is applicatiopn in actual software design, the utility is restricted to those cases where the model-view-controller paradigm is applicable

**Quality of the SecMVC Model** — 1 2 3 4

5. The principles and concepts from Appendix B are represented accurately in the model. — 1 2 [✓ 3] 4
6. Are there secure concepts which have been left out, despite their importance? — [✓ No] Yes

   If yes, please indicate which concepts.

7. Does the generalisation of secure concepts degrade their value in the model? — [✓ No] Yes

   If yes, please indicate why.

8. The model is of adequate quality with respect to security. — 1 2 [✓ 3] 4

   Please provide any further comment you have on this.

   I curious about the source of the specific secuir concepts you mention per general principle in appendix B. I'm not sure I agree with all of them. For eample: "Do not leave code open to reverse engineerign" (Least Privilege) goes agints the principle of open source (many eyeballs keep bugs shallow). Validation against several sets of best practices (eg ISO 27001) would be a good diea.

| **Efficacy of the SecMVC Model** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 9. A secure implementation can be derived from the model’s generalisations. | ☐ | ☐ | ☒ | ☐ |
| 10. The model would improve the likelihood of designing a secure solution. | ☐ | ☐ | ☒ | ☐ |
| 11. The model would improve the likelihood of implementing a secure solution. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on statements: 9; 10; and 11.

This is really hard to assess without trying this out in practice. But I like the general idea and believe it will work.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 12. The model is effective in a security context. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on this.

| | | | | |
|---|---|---|---|---|
| 13. Familiarity with this model will make a developer more likely to consider security. | No | ☐ | Yes | ☒ |

If no, please indicate why.

**Overall Impression**

14. What is your overall impression of the SecMVC model?

An interesting approach, that needs to be extended to other patterns

15. Please provide any suggested improvements to the SecMVC model.

MVC is only one of a set of possible patterns. For other applications other "security augmented" patterns would be necessary. I suspect that in that case the general principles will change (*you may have only 3), while the specific set of secure concepts remain relatively stable (but are assigned to different principles)

16. Please provide any further comments, criticism or suggestions.

# Appendix C3: Third Expert Review of the SecMVC Model

**Expert Reviewer Information**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1. Please provide the following details | | | | |
| Confidence level in Software Design Patterns: (1=low; 4=high) | ☐ | ☐ | ☒ | ☐ |
| Confidence level in Secure Software Development: (1=low; 4=high) | ☐ | ☐ | ☐ | ☒ |

Other relevant Fields of Expertise:

Infomation Security

Any further personal information you feel is relevant to this review process:

**Utility of the SecMVC Model**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 2. The background content is sufficient for understanding the proposed model. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on this.

More information could have been provided on the MVC pattern – only individuals whom are familiar with the pattern would be able to understand the diagram. Further information should be provided about how using this logically separates code into specific sections, and how this allows one to place and reuse security checks.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 3. The model can be recalled to memory in sufficient detail after establishing familiarity. | ☐ | ☒ | ☐ | ☐ |

Please provide any further comment you have on this.

The model is complex, with graphical notation that adds little value (e.g. gears). Logical application flow cannot be determined from the model. e.g. does the flow go via the IPS then SSL and then CAS?

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 4. The model can easily be understood and utilised. | ☒ | ☐ | ☐ | ☐ |

Please provide any further comment you have on this.

While the core concepts can be understood if one has a MVC and security background, it does not give concrete examples of implementation. The model seems to be .NET focused - which limits its usefulness unnecessarily -the concepts can be applied generically to programming languages as a whole.

| Quality of the SecMVC Model | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 5. The principles and concepts from Appendix B are represented accurately in the model. | ☐ | ☐ | ☒ | ☐ |
| 6. Are there secure concepts which have been left out, despite their importance? | No | ☐ | Yes | ☒ |

If yes, please indicate which concepts.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 7. Does the generalisation of secure concepts degrade their value in the model? | No | ☐ | Yes | ☒ |

If yes, please indicate why.

Most concepts are repesented correctly by their generalizations. By only viewing the model however, a normal developer may not fully grasp all concepts contained under the generalization.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 8. The model is of adequate quality with respect to security. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on this.

Output filtering should prevent confidential infomation for leaving the site as well - not just error messages containg confidential infomation. Remove the VB example - keep it generic. Layer your defence/defence in depth should be about using multiple security feutures, such that if one fails, the other compensate for it. The grouping of the "Layer your defence " does not seem correct.Futhermore, remove comments about Windows registry-far to spesific.

**Efficacy of the SecMVC Model**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 9. A secure implementation can be derived from the model's generalisations. | ☐ | ☐ | ☒ | ☐ |
| 10. The model would improve the likelihood of designing a secure solution. | ☐ | ☐ | ☒ | ☐ |
| 11. The model would improve the likelihood of implementing a secure solution. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on statements: 9; 10; and 11.

If understood in detail, the model would allow inspire a developer to think more about securing coding and design. Using MVC, the resulting application would be easier to secure. Full understanding of the model would however require the developer to understand the backgound security princables.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 12. The model is effective in a security context. | ☐ | ☐ | ☒ | ☐ |

Please provide any further comment you have on this.

| | | | | |
|---|---|---|---|---|
| 13. Familiarity with this model will make a developer more likely to consider security. | No | ☐ | Yes | ☒ |

If no, please indicate why.

**Overall Impression**

14. What is your overall impression of the SecMVC model?

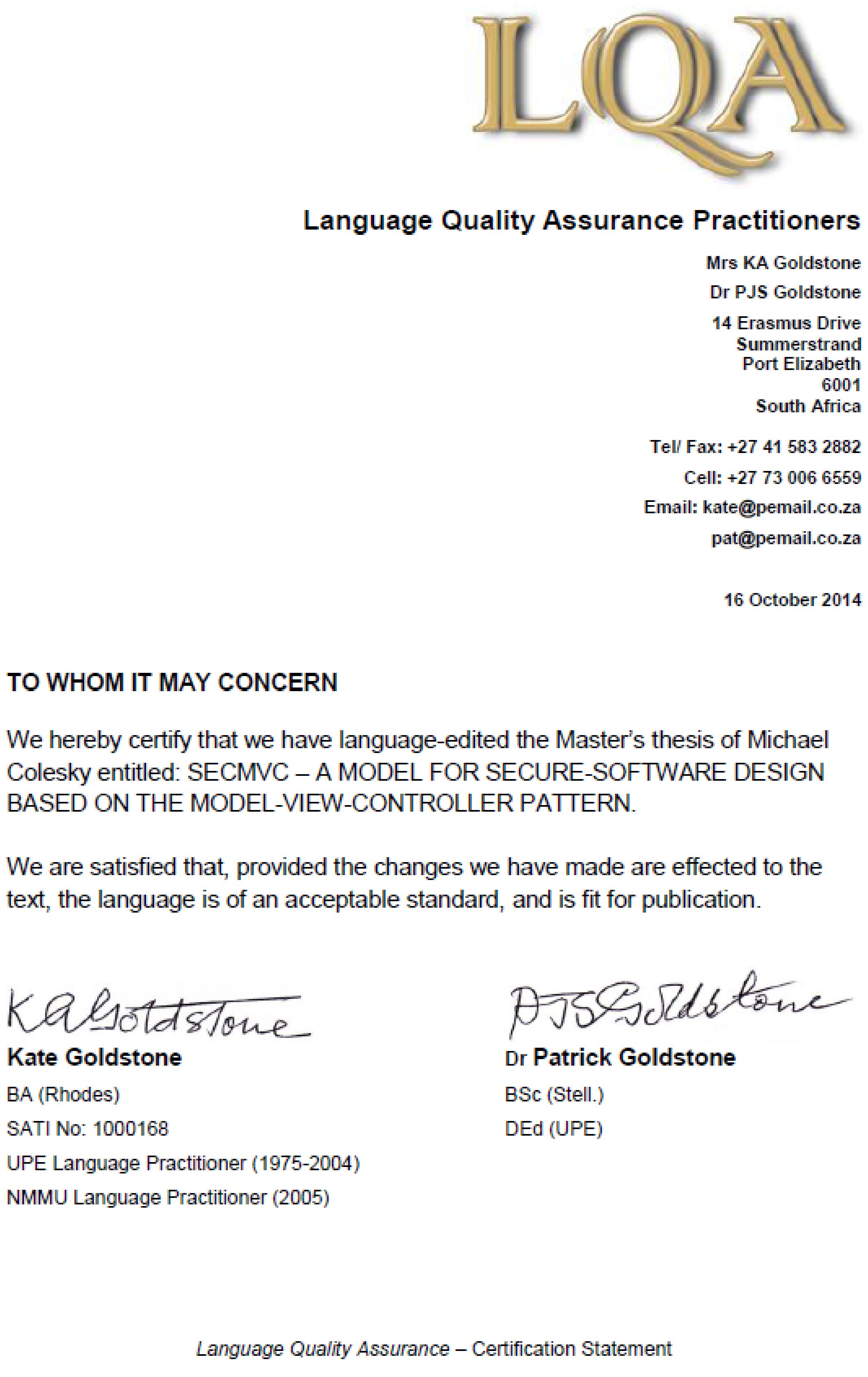
The model incorporates several tried and true principles of secure coding, and if used properly would aid developers in coding securely.

15. Please provide any suggested improvements to the SecMVC model.

A detailed explanation of each step should be provided. More background on MVC should be provided. "Layer your defense" should be reviewed to make sure that it fits the general the "Defense in Depth" principle. Output filtering also pertains to making sure that confidential information such as ID numbers does not get exposed. Logging is vital for incident response, and should not be limited- but stored securely. More details on encryption should be provided, such as not to do custom crypto.  A developer following the model may in theory be able to extrapolate the generalized principles to arrive at the correct implementation, however, some obvious specific principles may be missed. The general flow of the model should be explained or illustrated better.

16. Please provide any further comments, criticism or suggestions.

# Appendix D: Language-editing Certificate



LQA

**Language Quality Assurance Practitioners**

**Mrs KA Goldstone**

**Dr PJS Goldstone**

**14 Erasmus Drive**
**Summerstrand**
**Port Elizabeth**
**6001**
**South Africa**

**Tel/ Fax: +27 41 583 2882**

**Cell: +27 73 006 6559**

**Email: kate@pemail.co.za**

**pat@pemail.co.za**

**16 October 2014**

**TO WHOM IT MAY CONCERN**

We hereby certify that we have language-edited the Master's thesis of Michael Colesky entitled: SECMVC – A MODEL FOR SECURE-SOFTWARE DESIGN BASED ON THE MODEL-VIEW-CONTROLLER PATTERN.

We are satisfied that, provided the changes we have made are effected to the text, the language is of an acceptable standard, and is fit for publication.

[illegible]

**Kate Goldstone**

BA (Rhodes)

SATI No: 1000168

UPE Language Practitioner (1975-2004)

NMMU Language Practitioner (2005)

[illegible]

Dr **Patrick Goldstone**

BSc (Stell.)

DEd (UPE)

*Language Quality Assurance* – Certification Statement