# Agile Security for Web Applications

Xiaocheng Ge

Submitted for the degree of Doctor of Philosophy

University of York
Department of Computer Science

June 2007

*To my parents,*
*wife,*
*and children*

献给我亲爱的
爸爸，妈妈，
妻子
和孩子们

# Abstract

Web-based applications (or more concisely, Web applications) are a kind of information system with a particular architecture. They have progressively evolved from Internet browser-based, read-only information repositories to Web-based distributed systems. Today, increasing numbers of businesses rely on their Web applications. At the same time, Web applications are facing many security challenges and, as a result, are exposing businesses to many risks.

Security is a system property. To build a secure system is a difficult task. It has been suggested that a security development method should be integrated into more general information system development methods [1]. A suitable and promising class of methods for building Web applications is agile software development methods. These methods provide the ability to manage requirements change, which is desirable for the development of Web applications. However their use introduces substantial tension between the need to deliver a secure system and the desire to "embrace change".

This thesis proposes a novel approach to building secure Web applications using agile software development methods. The approach provides early analysis of threats, design for security, and subjects the design to thorough objective risk assessment, in an iterative, incremental, agile style. This approach addresses the tension mentioned previously.

The key contribution of this thesis is to provide a concrete example of agile security software development based on existing development practices, rather than to invent a brand new software development methodology or theory to address security problems. This is the *"right first"* step in the direction of agile security engineering.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr. Fiona A.C. Polack and Dr. Richard F. Paige, for their encouragement, patience, and guidance on my way to become a researcher. I also want to especially thank Professor Howard Chivers for his encouragement and influence. Without them, the work would not been possible. Thanks also to my internal assessor, Professor John A. Clark, and external assessor, Dr. Phillip J. Brooke, for their insightful comments.

Finally, very special thanks to my parents,葛惠来、辛淑珍, and my sister,葛笑梅, for their continuing love and support; to my wife,张莉, for her understanding and selfless love; and to my children,葛唯清、葛溥源, for giving me inspiration.

Xiaocheng Ge

葛晓程

# Declaration

This thesis is the work of Xiaocheng Ge and was carried out at the University of York. Work appearing here (or closely relating to it) has appeared in print as follows:

- X. Ge, R.F. Paige, F.A.C. Polack, H. Chivers and P.J. Brooke. "Agile Development of Secure Web Applications", in Proceedings of 6th International Conference on Web Engineering (ICWE2006), July 2006 [12].
- X. Ge, H. Chivers, F.A.C. Polack and R.F. Paige. "Adapting Security Risk Analysis to the Design of Database-centric Web-based Information System", in Proceedings of 18th International Conference on Software Engineering and its Applications (ICSSEA2005), November 2005 [13].

The discussion of database security protection used in Chapter 7 comes from the following papers:

- X. Ge, F.A.C. Polack and R. Laleau. "Secure Databases: an Analysis of Clark-Wilson Model in a Database Environment", in Proceedings of 16th International Conference on Advanced Information Systems Engineering (CAiSE04), LNCS 3084, Springer-Verlag, page 234-247, June 2004 [14].
- X. Ge, F.A.C. Polack and R Laleau, "Secure Database Development and the Clark-Wilson Security Model", in Proceedings of Atelier Securite des Systemes d'Information (SSI'04), May 2004 [15].

# 1 Introduction

*This chapter states the motivation and hypothesis of the research presented in this thesis. It also lists several statements of information security which are foundation of the research. At the end of this chapter, an outline structure of this thesis is given.*

## 1.1 Challenges of the Internet

Security has been a major concern in the development [16] and use of all types of information system for a long time. After decades of efforts, many contributions have been made by the security engineering community. One key recommendation is that security should be taken into account at every stage of the system development life cycle [17, 18]. As well, concrete security practices have been produced [19, 20, 21], and many security development engineering methods (i.e, systematic and with well defined procedures) [1, 22] have been introduced and applied.

Security is a system issue that takes into account both security mechanisms (such as access control) and the engineering of security (such as a robust design that makes it difficult for software attacks to succeed). Sometimes these overlap, but often they do not. Security engineering is concerned with building secure software. It relies heavily on the discipline of software engineering, liberally borrowing methods that work and making use of critical engineering artifacts. A sound software engineering method is a prerequisite to sound software security.

The Internet is driving a *quiet revolution*[1]: it is able to deliver information to anyone, anywhere, quickly and inexpensively. A significant impact of this power will be increasing the inter-connectivity of people, regardless of geography and time zone. In fact, this impact has already been discernible with Internet paradigms of work, for example with IBM's Jazz project[2].

The Internet brings many new security challenges, particular to businesses, mainly because the operational environment of Internet applications is wide open (and sometimes unpredictable). For example, access control is normally designed and implemented in

[1] details at http://news.bbc.co.uk/1/hi/business/5235332.stm
[2] http://jazz.net

Web applications but once applications are made available on the Internet, they are sometimes difficult to manage correctly.

Another impact comes from the fact that changes during the life cycle of Web applications are unavoidable. These changes include modified business processes due to competition pressure, adoption of new development techniques, etc. As a result, Web applications have the **ability to change** in response to the changes in business needs [23]. The phrase **ability to change** refers to the definition of agility, *"a comprehensive response to the business challenges of profiting from rapidly changing, continually fragmenting, global markets for high-quality, high-performance, customer-configured goods and services"* [24]. This definition of agility was introduced more than ten years ago initially for manufacturing, but it remains valid today for software development. This, in particular, has led to the development of so-called *agile software development.*

## 1.2 Motivation and Proposition

The ideas of agile software development emerged in the early 1990s as part of a reaction against plan-driven methods, as typified by a heavily regulated, regimented, micromanaged use of the waterfall model of development. Agile methods are evolving, and are becoming more mature after the proposal of the Agile Manifesto [25]. Today, Agile methods have become a family of development processes seeing increased use. However, their application in building secure Web applications remains an underdeveloped research area.

Agile methods grew out of the real experiences of leading software professionals, such as Kent Beck (the founder of Extreme Programming), who had experienced the challenges and limitations of traditional waterfall development on project after project. The approach promoted by agile development is in direct response to those challenges.

In its simplest form, agile software development offers a lightweight, iterative and incremental framework for helping teams, given a constantly evolving functional and technical landscape, and maintains a focus on the rapid delivery of business value.

Particularly, agile software development can accelerate the delivery of initial business value through a process of continuous planning and feedback. As a result of this iterative loop of planning and feedback, teams are able to easily **adapt** to changing requirements throughout the process. By measuring and evaluating status based on working software, there is much greater **visibility** into the actual progress of projects. As a result of adopting an agile development process, it is suggested that a software system is finally produced that better addresses the business and customer needs.

In 2006, over 700 developers and managers responded to a survey conducted by VersionOne and sponsored by the Agile Alliance. According to this survey [6], most respondents said that they have been practising agile development methods for 2 or 3 years. When asked to estimate their successes and quantify the benefits they have achieved, 40 percent claimed that they had significantly improved their ability to manage changing priorities and 24 percent observed enhanced software quality (see Table 1.1).

| What value have you **actually realised** from implementing Agile practices? | Significantly Improved | Improved |
|---|---|---|
| Enhanced ability to managing changing priorities | 40% | 52% |
| Enhanced software quality | 24% | 50% |
| Alignment between IT and business goals | 22% | 44% |
| Improved team morale | 20% | 54% |
| Accelerated time-to-market | 20% | 51% |
| Increased productivity | 17% | 58% |
| Reduced project risk | 17% | 55% |

**Table 1.1:** Benefits of Agile Processes (derived from [6])

As the survey [6] suggests, Agile methods can potentially bring much benefit. But when considering development methods that focus on guaranteeing system-wide properties, such as security or safety, it is found that they are overwhelmingly plan-driven; in particular, established security development methods are all plan-driven [1,22]. Thus a general question of feasibility arises: can an Agile method successfully be used to establish system-wide properties?

To attempt to answer this question, it is useful to first step back and compare plan-driven and Agile methods. Such a comparison was given in [7]; the differences are summarised in Table 1.2.

| **Agile Methods** | **Plan-driven Methods** |
|---|---|
| Low criticality | High criticality |
| Senior developers | Junior developers |
| Requirements change very often | Requirements do not change too often |
| Small number of developers | Large number of developers |
| Culture that thrives on chaos | Culture that demands order |

**Table 1.2:** Agile *vs.* Plan-driven method (derived from [7])

[7] concluded that Agile methods are a more appropriate choice when a small number of senior developers are working on a project of low criticality. Most projects of Web applications are the typical example of this. However, criticality concerns cannot easily

be avoided, particularly for businesses that interact with, or run over, the Internet. In such situations, security is a key concern; however, most businesses do not do enough to ensure that their Web applications are acceptably secure [26].

Research on agile secure software development, such as [27,28,29,30], has already addressed the importance and key difficulties of rigorously and fully considering security concerns when applying an Agile method. The two main challenges are as follows:

- Security is an emergent property, and cannot be fully considered simply by ensuring the security of individual components. Plan-driven security engineering processes deal with this by developing an understanding of the overall system architecture, which allows components and security mechanisms to be modelled, and their relationships understood. However, most Agile methods do not build, nor do they encourage, a description of the software architecture.
- Arguing that a software system is secure relies on evidence; if the argument requires evaluation against standard security criteria, evidence must be produced. Evidence is normally in the form of *documentation*, for example fault tree analysis report, system models, test plans and results (see the next section). Agile software development de-emphasises documentation and concentrates on the production of working software quickly. Thus, the question is: how can we record the evidence requirements for security in a development method that emphasises the production of code?

Therefore, there is a gap between ambition and reality. The ambition of business is to build software systems using Agile methods, so as to better manage different kinds of change. The reality of software system development is that there are properties — such as security — that are difficult to establish using Agile methods, particularly due to their iterative and incremental nature, and because of their deprecation of documentation. What could fill this specific gap is *Agile security engineering methods*.

The aim of the research presented in this thesis is to investigate the feasibility of developing an Agile security engineering method. This is done by producing such a method and demonstrating its use on a concrete and relevant case study: a Web application. Such applications have important security requirements. However, it is believed that the results may be more generally applicable to any kind of application that has security requirements. The fundamental requirement for assessing whether it is a suitable method is that **it is adaptive to changing requirements and can deliver an acceptably secure (or secure enough) software system** (*acceptably secure* is defined in the sequel).

The hypothesis of this thesis is: **it is possible to develop a secure enough Web application by adopting an agile method.**

There are several dimensions to this problem, in particular:

- What kind of systems are the focus of agile software development? Agile methods generally are best suited to developing low criticality, small-to-medium size systems, for example a Web application. The research in this thesis focuses on the development of Web applications.

- How should security be considered in an agile development process, and what are the limitations encountered when attempting to build a secure enough system using an Agile method?

This thesis will contribute towards answering these questions.

In the hypothesis, the notion of *"secure enough"* refers to the fact that the predicted security risks of the target application are mitigated to an acceptable level (i.e, the level required by the customer at a specific point in time). It is used because there is no absolute definition of security.

## 1.3 Assertions of Information Security

The scope of the research presented in this thesis will cover multiple aspects of software engineering, including security engineering, agile software development, and Web engineering. There are several questions to be answered:

1. What is the state-of-art in security development and agile software development?
2. What are the hurdles to adopting an Agile method to build a secure Web application?
3. What will be the concrete deliverables of the research and how will they be evaluated?

The first question will be answered in Part I of this thesis. The second question will be addressed throughout the thesis as a whole. The third question will be discussed in the remainder of this chapter. As a prelude to doing so, several fundamental assertions are made firstly about security development, in order to properly set the context for answering, in particular, the second question and the overall research hypothesis.

### 1.3.1 Security development is an exercise in risk management

Security may mean different things to different people. Sometimes it even means different things to the same person, depending on the context. For example, security to a college porter is whether the doors to the college are locked; but to a manager of a company it is whether the sensitive assets of the company are protected. In order to evaluate security in a standardised way, there are several published international security standards; for example, there is the Trusted Computer System Evaluation Criteria

(TCSEC, also known as the Orange Book) [8], Information Technology Security Evaluation Criteria (ITSEC) [10], and Common Criteria for Information Technology Security Evaluation (Common Criteria, or CC) [9]. These security evaluation criteria promote a standard way of evaluating systems, and also provide sets of standard *assessable* security requirements.

Sometimes, security requirements may be inconsistent with the functional requirements of the system. The means to addressing this type of inconsistency is to understand the trade-offs that must be made from a business perspective. In other words, when we need to reconcile functional and security requirements, we must determine how to avoid security problems in a way that is acceptable to the stakeholders. A technical means to achieve this is through *software risk management*, which can help balance security and functionality, and bridge the gap between security and development staff.

In mature engineering disciplines, more general risk management has been *de rigeur* for a very long time [31]. It has been practised in many areas: in planning financial strategy, in construction engineering, and even in life science. Software security risk management is also a mature discipline (see [32, 33, 34, 35, 36]). Risk management is widely regarded as the only viable method of providing a cost benefit justification for security controls, or alternatively, of judging which controls provide most benefit. As a consequence, security risk management is the basis of most national standards for information security management, such as [32, 37, 38, 39], and best practice, for example [32,40].

### 1.3.2 Applying agile development to build a secure system is challenging

Agile development is, by definition, iterative and incremental; but building a secure enough system requires knowledge of the *whole* system. The challenge is to reconcile these two aspects.

This challenge is considered in more detail. Security concerns are applied to all layers of a software system architecture: from the bottom (the infrastructure) to the top (the data). Application security is only one layer in that architecture. Application security is challenging because it is not easy to isolate security related concerns in the architecture of a software system. The primary reason for this difficulty is that security is a pervasive concept. For example, the security of a Web application depends not only on the absence of vulnerabilities in the application logic, but also on the security of the network and the operating system.

Therefore, an overall architecture and plan (i.e., where and when to implement security mechanisms) are essential. This will be feasible and, in some cases, straightforward in a plan-driven development, because the method itself may require a thorough analysis at the beginning of the development. But with Agile methods, developing an overall architecture is generally not mandatory. So the first hurdle in adopting an Agile method to build secure software is the trade-off between understanding the overall architecture against a loss of agility.

Another obvious issue with developing a secure system using an Agile method is the production of documentation. For the needs of security development and security evaluation, a large amount of documentation is useful, and sometimes required. For example, the Common Criteria addresses a series of documents covering every phase of the software development life cycle (SDLC), including requirement specification, design documents (i.e., functional specification, design, source code), testing documents, and operation documents (i.e., user manuals, administrator manuals). Agile methods in general attempt to minimise documentation. An extreme example is that of Extreme Programming (XP), which claims that source code is the best documentation a software needs. Thus, when using an Agile method for building a secure system, it must somehow ensure that documentation needs are met without unduly sacrificing agility. Providing the means for qualitatively assessing the value and importance of documentation in an Agile method may prove to be a useful way forward.

### 1.3.3 A good process is a product too

The research in this thesis aims to deliver an Agile method that can help to produce a secure Web application. It is necessary for us to be able to evaluate the quality of the method that we produce. [41] explained the factors that determine the quality of a software product. The method engineering community takes the view that a process, one of the constituent parts of a development method, is a product too. The quality requirements of a software development process are [42]:

- Effectiveness. An effective process must help produce the right product. It does not matter how elegant and well-written the software, nor how quickly it has been produced. If it is not what the customer wanted, or required, it is of no value. The process should therefore help to determine what the customer needs, produce what the customer needs, and, crucially, verify that what has been produced is what the customer needs.
- Maintainability. However good the developers, things will still go wrong with the software. One of the goals of a good process is to expose the developers' thought processes in such a way that their intention is clear. This allows faults to be discovered and remedied, and desired changes to be made more easily.
- Predictability. Any new product development needs to be planned, and those plans are used as the basis for allocating resources including time and people. It is important to predict accurately how long it will take to develop the product. A good process will help lay out the steps of development.

- Repeatability. If a process is discovered to work, it should be replicated in future projects.
- Quality. Quality is defined as the product's fitness for its purpose. In this thesis, security is the major criterion of quality. The goal of the proposed process is to enable software engineers to ensure a secure Web application. The process should provide a clear link between a customer's desires and a developer's product.
- Improvement. A process must be able to identify and prototype possibilities for improvement in the process itself.
- Tracking. A defined process should allow management, developers, and customer to follow the status of a project. It continuously monitors how good the predictions are, and hence how to improve them.

These requirements for a software process help to form the criteria used for evaluating the research. Evaluation will consist of two parts: evaluation of the software deliverable and evaluation of the process itself. The evaluation of the software deliverable is given in Chapter 7. It discusses the quality requirements of the product (i.e. security in this thesis). The evaluation of the process itself is in Chapter 9, which discusses the rest of the requirements listed above.

### 1.3.4 Unaddressed but important issues

There are several identified issues that, while important, are outside of the scope of the research presented in this thesis.

#### Management issues

Building software is usually not just a technical process; it also requires management, including tasks such as constructing a development team, organising the members in the team, managing the deadlines and the cost of the project, etc. All of these are mentioned in the Agile Manifesto [25]. Hence, applying an agile method is a mixture of **people** and **technology** under the umbrella of agile principles and practices. This thesis focuses on technical issues of building secure system; while aspects of management issues will be touched on throughout the thesis, they will not be the main focus.

#### Size of team and project

So far, most methods in the class of agile software development claim they are lightweight methodologies for small-to-medium-sized teams, and for developing software in the face of vague or rapidly changing requirements. In a large team, it is not realistic, or at least very difficult, to achieve good communication and quick feedback between customers and developers, which are key aspects of Agile methods. It is worth noting that a few Agile practitioners have looked at the problems of agile methods for large projects or organisations, for example [43,44,45]. But results so far are inconclusive.

The methods proposed in this thesis are targeted at small projects and focus on technical aspects of development.

## 1.4 Framework of Research

The approach taken in the thesis to incorporating security into agile methods is based on a comprehensive understanding of software security and security practices, particularly Web application security. Moreover, the solutions developed and proposed in this thesis are also based on a good understanding of existing agile practices. Many project managers and software engineers have already applied agile methods in various types of projects and have noticed the importance of software security at the same time. Their experience is accumulated and shared via many means, such as articles, personal blogs, forums etc. Initial ideas and some background knowledge come from these sources, which are discussed in Part I of the thesis.

This research has a simple and clear thread which has three stages:

1. Literature review, and determination of a suitable Agile method with which security practices can be integrated;
2. Proposal of an integrated version of the Agile method that includes security practices; and
3. Demonstration that the process satisfies its requirements in a case study.

Consequently, the results of the research at each stage produce the following:

- Understanding Information Security and Security of Web Applications

  After reviewing the literature of security of the Web and Web applications, key security requirements for Web applications are categorised. This will help developers simplify their understanding of their security problems. This is presented in Chapter 2 - Security and Security Practices, and Chapter 3 - Web Application.
- Exploring Agile Software Development Methods
  Many Agile methods have been proposed and applied in different projects. This is presented in **Chapter 4 - Agile Software Development.**
- Toward Agile Security Engineering
  The Feature Driven Development (FDD) method is reviewed in **Chapter 4 - Agile**

**Software Development**. Consideration is given to the theory of agile security, the models and their relations involved in the development process are analysed. Each Agile method has its own special characteristics; adding explicit consideration of security should keep the agility of the process as much as possible. The author proposes an integration of FDD and security practices to the combination of Agile methods and security concerns. These are presented in Chapter 5 - Security in Agile Development, and Chapter 6 - A Solution of Security Integration: secure FDD.

- Evaluation of Proposed Process
  Finally, the integrated method is demonstrated on a case study. The method and the case study are evaluated. The most important lessons learned from the case study are stepping-stones to future research. These are presented in Chapter 7 - Case Study and Chapter 8 - Limitations of Case Study and Research.

## 1.5 Structural Overview

The structure of this thesis is shown below.

**Part I Background Knowledge and Analysis** provides a general background to the research and discusses several issues related to the research.

**Chapter 2 - Security and Security Practices** reviews the several commonly used security artifacts.

**Chapter 3 - Web Application** explains *"what is a Web application?*, and *"what are the security concerns of Web applications?"* The answer to these questions provides a general background of the research. At the end of this chapter, a list of common security vulnerabilities of Web applications is summarised, which will provide security criteria for the case study discussed in later chapters. The integrated method will therefore target these security criteria using specially developed Agile security practices.

**Chapter 4 - Agile Software Development** introduces a suitable Agile method, and discusses in depth several issues about feature driven development (FDD). These discussions explain why this Agile software development method is suitable.

**Part II Integration of Agile and Security Development** is the body of the research, including chapters of presenting the arguments of the integration, the details of the example of security integration, and the demonstration of how it works in a Web application project.

**Chapter 5 - Security in Agile Development** gives an analysis of the problems encountered when attempting to integrate risk assessment and Agile software development. Based on the analysis, a foundation for security integration is drawn in this chapter.

**Chapter 6 - A Solution of Security Integration: secure FDD** demonstrates a concrete example of Agile security integration. The principles of security integration are: 1) the integration is an Agile software development method, and 2) the integration improves the security of the target software system. In this chapter, the security artifacts in secure FDD are described in detail.

**Chapter 7 - Case Study** demonstrates an example of the integration in a Web-based application project. At the end, an analysis of the case study helps to evaluate the proposed agile security engineering approach.

**Chapter 8 - Limitations of Case Study and Research** identifies and discusses the limitations of the case study and the research presented in this thesis. It helps the readers gain a better understanding of the research.

**Part III Conclusion and Future Work** summarises the conclusions in the thesis and lists some interesting directions for possible future research.

**Chapter 9 - Evaluation of Research** summarises the contributions of the research and evaluates the quality of the proposed process. This chapter also draws an overall conclusion from the research, and indicates several possible directions for the future research work.

# Part I

# Background Knowledge and Analysis

# 2 Security and Security Practices

*Information security is a broad domain, and it is the first engineering field reviewed in the research. The purpose of the review is to gather enough knowledge so that a set of requirements for a security development process can be proposed which can be used as criteria of Agile process selection. The concepts related to security development will also be reviewed, including a general view of information security, and some security development artefacts. At the end of this chapter, a set of requirements for a security development process is proposed.*

## 2.1 Information Security

Information security means protecting information and information systems from unauthorised access, use, disclosure, disruption, modification, or destruction in order to provide security goals, which are often summarised as confidentiality, integrity and availability [19,21,46,47]:

- *confidentiality* means preserving authorised restrictions on read access and disclosure, including ways of protecting personal privacy and proprietary information;
- *integrity* means guarding against improper information creation, modification or destruction, and includes ensuring information non-repudiation and authenticity. Integrity always includes the following overlapping sub-goals [48]:
    1. Preventing unauthorised users from making modifications.
    2. Preventing authorised users from making improper modifications.
    3. Maintaining internal data consistency (self-consistency of interdependent data) and external data consistency (consistency of data with the real-world environment that the data represents).
- *availability* means ensuring timely and reliable access to, and use of, information. Goals of availability include timely response, fair allocation, utility or usability, and controlled concurrency [49].

Information security must protect information throughout its life span, from initial creation to final disposal. Figure 2.1 is a simple illustration of the life span; the information must be protected in storage, in transit, and in use. There are many different ways that information and information systems can be threatened, and their protection must take a holistic approach.

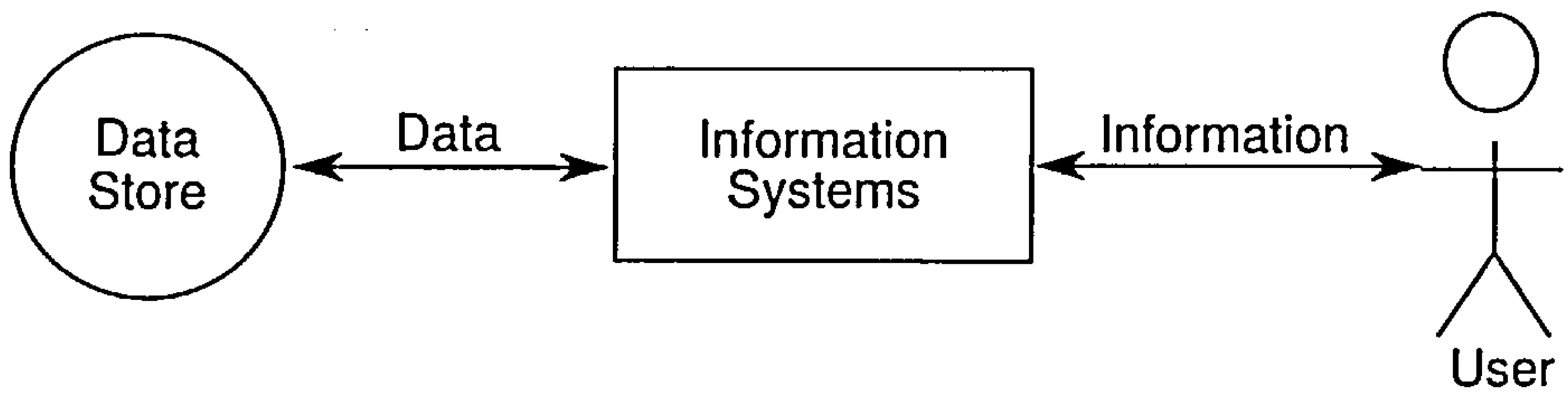



**Figure 2.1:** Information Life Span

A thorough technical analysis of information security and security goals is presented in [46]. The security of an information system can also be achieved through non-technical means, such as organisational, personnel, physical, and administrative controls.

The rest of this section considers, firstly, broad issues in security; and secondly, security as a process. This illustrates the interplay of factors involved in information systems security.

### 2.1.1 Security is a broad issue

Security is a broad concern, not a pure technical problem. At a high level of abstraction, information security contains three elements: policy, people, and technology, as shown in figure 2.2. Each element depends, in some manner, on the others.

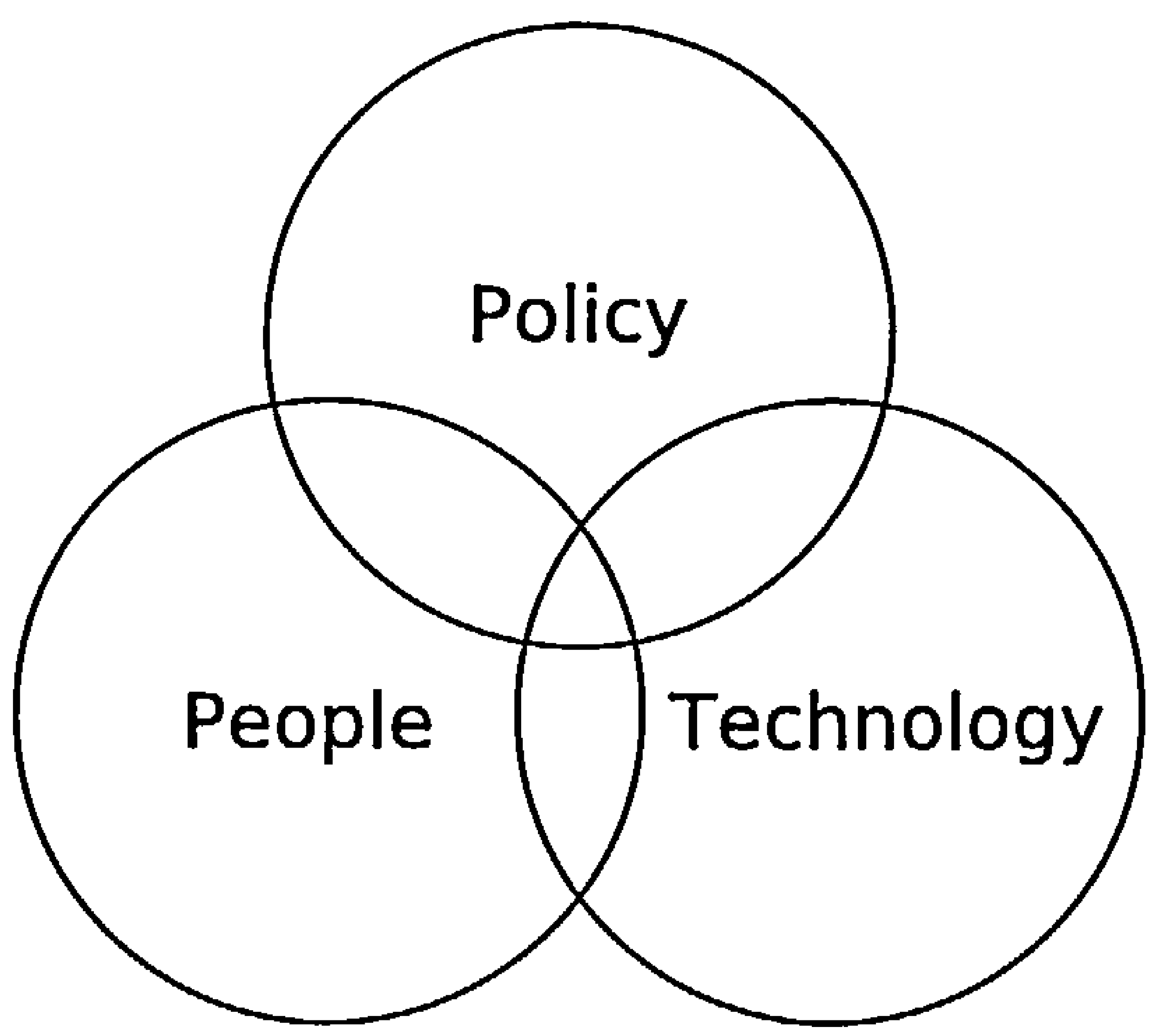



**Figure 2.2:** Three Elements of Security

### Policy

Policy is the definition of what it means for the information system to be secure. It captures the security qualities required by an organisation, as well as wider issues such as legal requirements. Typically, policy comprises a set of security statements, relevant standards, and control documentation. This is basically the written security environment - the bible that software developers and operators refer to for direction and guidance.

Security policy is "a wide ranging document which is about managing the business as a whole, managing it securely and protecting a company's key asset - its information" [50]. It is a set of rules and practices that specify or regulate how a system or organisation provides security services to protect sensitive and critical resources. In information systems development, security boils down to devising and enforcing policies that describe rules for access resources.

In many information-oriented organisations, there is no explicit security policy, rather an implicit policy that is assumed to be fairly obvious and widely shared. Without a well-defined policy, it can be difficult to identify whether events have the potential to cause a security breach.

### People

The people element comprises the various human roles and their responsibilities. Within the development team, for instance, key roles include senior project manager, security administrators, and system architects. Once a system is delivered, security-related roles include system administrators, end users and auditors.

### Technology

Technology is a wide-ranging element of security, and includes the tools, methods, and mechanisms put in place in building the software. Technical issues include the method of delivering security, architectural issues, and technical mechanisms of implementing security (e.g., protocols, access control mechanism).

Issues arising from the architectural complexity of systems can be simplified by a layered approach. For information security, four architectural layers are used in this thesis[1]:

1. **Sensitive Data Layer (SDL)** contains a variety of sensitive data.

[1] These (or very similar) layers are common in the literature

2. **Application Layer (AL)** contains the application, the supporting software and middleware, and the underlying operating system.
3. **Infrastructure Layer (IL)** includes the hardware and networking that support the AL.
4. **Social Environment Layer (SEL)** is the social environment of the information system.

In this layered security architecture, technical issues are primarily in layers SDL, AL, and IL. SEL concerns human issues. Policies may relate to any or all of the four layers, and is the first thing considered in development of security for each layer. A holistic approach must consider every component in the layered architecture, and the ways in which security is built up. The layering on and overlapping of security measures is the principle of *defence in depth*. The overall security is a chain whose strength is no greater than its weakest link.

In terms of security mechanisms, three categories can be distinguished:

- **Logical controls** are part of the software designed to monitor and control access to information and systems. Measures such as authentication mechanisms are typical protections applied to components from SDL and AL, whilst examples of software protection of components in the IL include network firewalls and intrusion detection systems. The IL protection may be outside the control of the information system and its developers.
- **Physical controls** can be used to monitor and control the environment of the work place and access to and from computing facilities. Examples include locked doors, alarms, CCTV, and security guards. Physical separation of the network or the work place into functional areas may also be used.
- **Administrative controls** seek to enforce policies, for instance using approved written policies, procedures, standards and guidelines. Administrative controls form the framework for running the business and managing people. They inform people how day-to-day operations are to be conducted. Administrative controls might encompass corporate security policy, password policy, hiring policy, and disciplinary policy.

Just as policies are the key to the development of human and technical security, the administrative controls form the basis for the selection and implementation of logical and physical controls.

Figure 2.3 shows how policy, people and technology are interrelated with the architectural layers and the control concepts. Controls are identified as policies — policies cover all four layers — and are used to determine what control mechanisms are appropriate. Logical control applies to the SDL, AL, and those parts of the infrastructure that concern software. Physical controls apply to physical aspects of the infrastructure and to parts of the SEL. Administrative control operates on people, and is thus confined to the SEL.

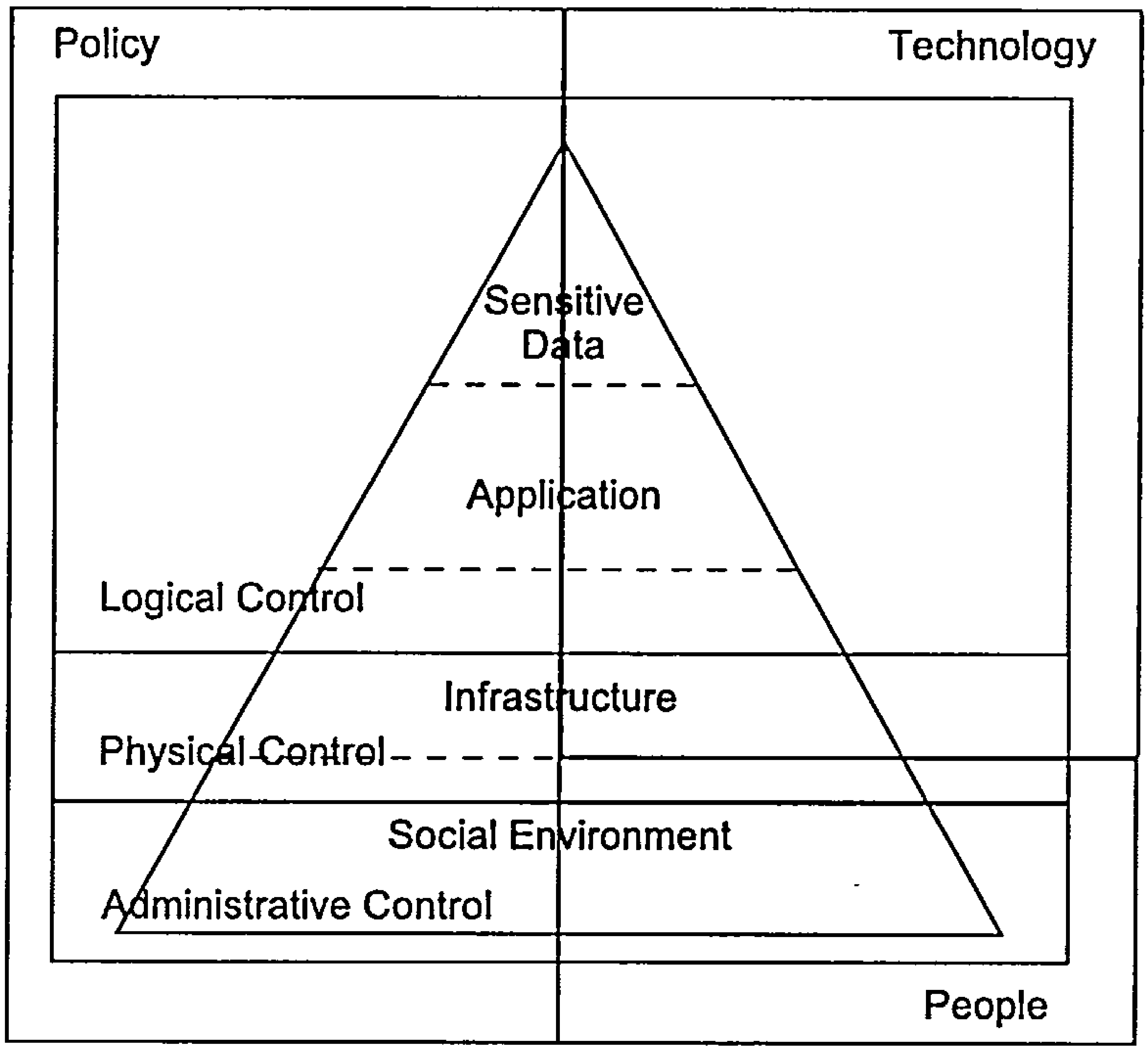



**Figure 2.3:** Overview of Security

Policies are presented in different forms according to whether they relate to the whole abstract system or to a particular layer or layers of the architecture. For example, *only database operators can an access the data in X* is an access control policy on the SDL, whereas *Room 101 is forbidden area for all staff except chief security commander* is a policy on the SEL.

### 2.1.2 Security is a process

The previous section considers abstract security concepts such as policies, architecture and controls. Information security cannot rely solely on the provision of security mechanisms; it also depends on the quality of engineering; for example a robust design can reduce the potential for software attacks on the implemented system. Sound software engineering is a prerequisite for software security, and development of secure software draws on development artefacts devised for critical systems engineering. In this section, the development of secure software is addressed.

Security can be thought of as a system quality that is delivered through the SDLC. It is useful to distinguish qualities of the entire system (*operational* or *run-time* [51] qualities) and *developmental* qualities (also called *development-time qualities* [51]and *build-time requirements* [52]), of development artifacts (e.g. architecture, design and code).

Operational qualities are relative to customer goals, whereas developmental qualities relate to the goals of developer (the person or organisation that develops an information system).

- *Operational Qualities*, such as usability, reliability, quality of service, safety, security and scalability, relate to how well an operational system meets its functional requirements — where *how well* is judged using **externally**-observable or measurable properties of the system behaviour. For example, the customer may assess security in relation to the customer's values or concerns, or may use standard international security evaluation criteria.
- *Developmental Qualities*, such as maintainability and reusability, reflect the developer's vested interest in the development process and its artefacts (such as architecture, design, and code). Developmental qualities of artefacts influence the effort and cost associated with the current development as well as support for future changes and users.

In relation to secure software, acceptable operational security is predicated on developmental quality. An important principle is to address security issues early in the software development life-cycle (SDLC) — knowing and understanding common threats, designing for security, and subjecting all software artefacts to thorough objective risk analyses and testing. It is accepted that considering security early is less expensive and more effective that treating security as an add-on to an operational system.

This thesis focuses on security at the development stages of a SDLC, and thus is primarily concerned with developmental qualities that are necessary to achieve operational quality.

The ultimate goal of security development is secure software, but this aspect of security is an operational quality that is judged by a third party. Knowing how security might be assessed will influence the way that software is developed. The international community has set up security standards (see section 2.2.2, below) that improve the judgement and comparability of the security quality of information systems. The standards help developers to address the question *what is the most effective way to protect information system?* In the developmental context, however, security is only one desired quality: functionality, time-to-market, cost, flexibility, re-usability, and ease of use are also important. The security concern may directly clash with other project quality goals. Balancing quality goals in a software project is nontrivial. Risk assessment can help balance security and other goals, and may be the only rational decision tool that allows the security quality to be judged in context [53].

Figure 2.4 shows the author's life cycle view of security, and shows the interrelationships of some of the concepts considered in this section. In the centre is a simple standard SDLC, from initiation to decommissioning. The development stage, and associated scope of developmental qualities, relate to the first three components of the SDLC; the operation stage and operational qualities relate to the post-implementation SDLC. International evaluation standards are typically focused at the start of development and the end of implementation (before operation), whilst risk assessment should be an ongoing process throughout the SDLC.

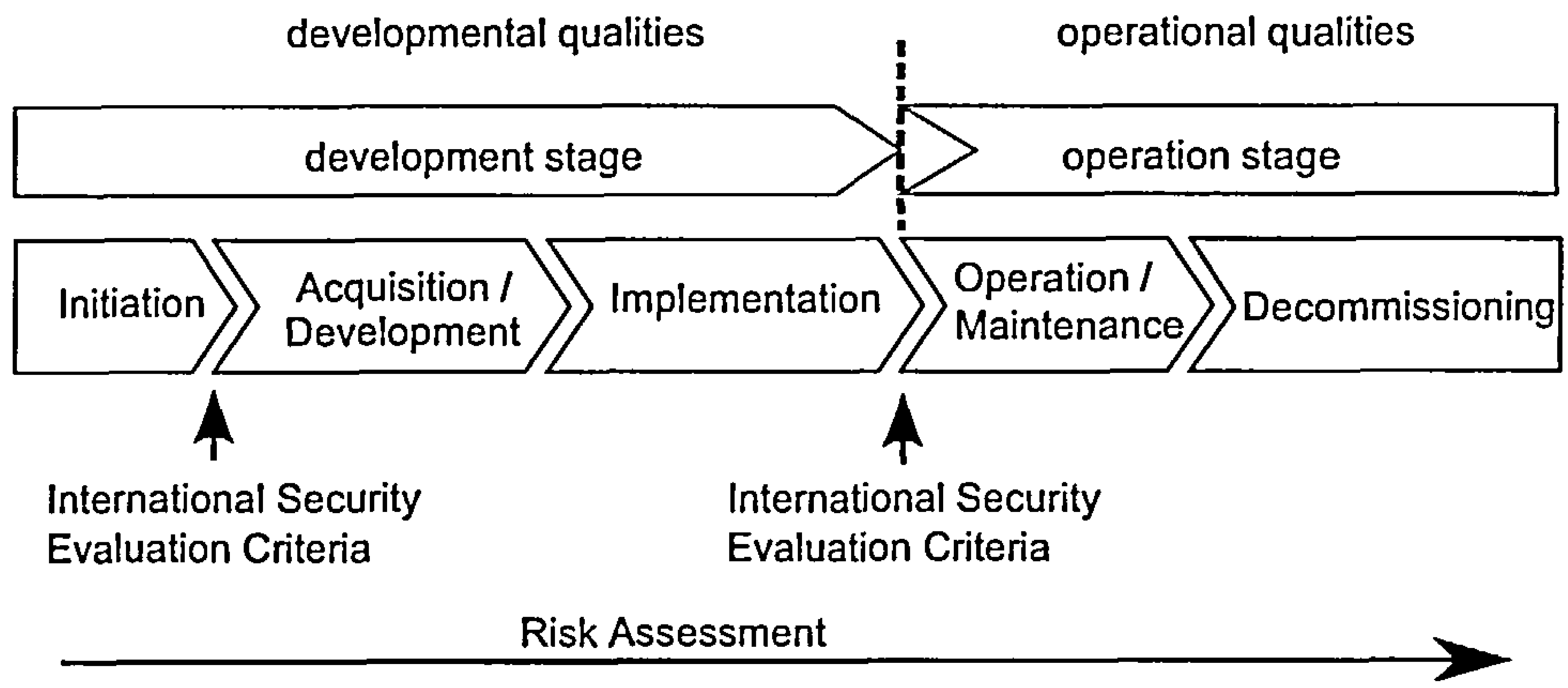



**Figure 2.4:** System Qualities, Artefacts, and the Software Development Life Cycle

The discussion identifies a number of *security artefacts*, or *security improvement artefacts*. In this thesis, security artefact refers to any approach, method, or technique that helps to improve information security. Because of the importance of security criteria, applying these criteria is a security artefact. Risk assessment is also an important artefact, not only because of its maturity, but also because of its value — it provides decision criteria for security in the business and social context. These and other security artefacts are covered further in the next section.

## 2.2 Security Improvement Artefacts

In the previous section, the development of security is seen as a progressive process. Software security engineering can and should borrow from other disciplines in computer science and software engineering when developing and evolving best practice.

McGraw [54] identifies the following influences in the development of security engineering best practice:

- Security requirements engineering;
- Design for security;
- Risk assessment, security testing, and use of international security criteria;
- Guiding principles for security;
- Auditing software for implementation risks, architectural risks, automated tools, and technology developments (code reviewing, information flow and so on), and
- Common security checklists.

An alternative view (of the author) is shown in Figure 2.5, which presents the secure development process (in slightly more detail than in figure 2.4) and a range of associated security improvement artefacts. The list of artefacts is not complete, but these are perhaps the most commonly used in the development of information systems.

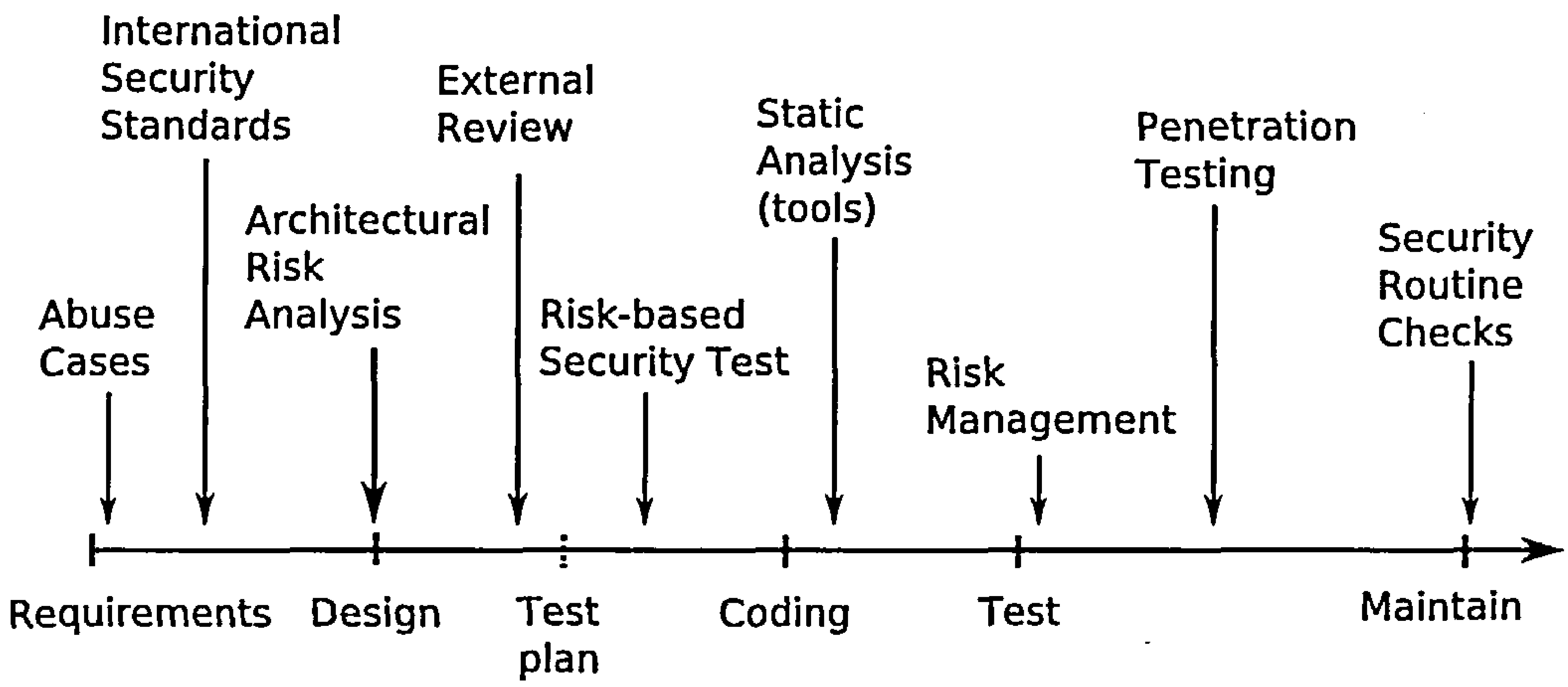



**Figure 2.5:** Security Artefacts and their relation to the SDLC

These security artefacts represent relatively independent activities. A more coherent approach to a secure SDLC is provided by systematic security methods. In the remainder of this section, existing systematic security development are reviewed briefly. Next, the two artefacts identified in the previous section, security evaluation standards and risk assessment approaches, are discussed. Finally, a range of other security artefacts is briefly considered.

### 2.2.1 Security Development Methods

Table 2.1 presents Baskerville's [1] succinct summary of methodical software development. Three generations are identified. The first is a mechanistic use of check-lists; the second provides heavyweight but relatively prescriptive engineering methods, whilst the last generation focuses on logical abstraction and analysis. The second and third generations saw the introduction of influential *plan-driven* methods, where the developer follows a specified life cycle, applying specified methods and techniques to achieve the goals of each stage (as in the SSADM life cycle on the left side of Figure 2.6, below).

The final column of Table 2.1 notes some features of security development in each generation. Risk analysis has played a large role in security development since the check-list methods of the 1970s. Security considerations have already been integrated into these software engineering methods, through an evolution of existing information systems security development and management methods.

At the time of Baskerville's 1993 summary [1], software engineering process was dom-

| Generations | Primary Features | System Development Methods and Typical Tools | Security Development Methods and Typical Tools |
|---|---|---|---|
| 1st Generation: Checklist Methods (1972-) | Map of limited solutions on to the information problem | Vendor's technical procedures and literature | Security checklists and risk analysis |
| 2nd Generation: Mechanistic Engineering Methods (1981-) | A partitioned solution that matches functional requirements | Top-down engineering, rapid prototyping, system and logic flowcharts | CRAMM, control point and exposure analysis matrices. |
| 3rd Generation: Logical Transformational Methods (1988-) | Highly abstracted design expressing problem and solution spaces | Structured analysis, data modelling, information engineering, software systems, data flow and entity relationship diagrams | Design of control logic, etc. |

**Table 2.1:** Summary of Baskerville's Generations of Methods [1]

inated by plan-driven methods.

A typical example of the third-generation security development method is represented by the SSADM[2]-CRAMM[3] Interface. SSADM and CRAMM started as second-generation methods, but evolved into logical transformation approaches that could be integrated to address operational qualities of information systems as well as functional requirements. In Figure 2.6, the left side is the SDLC of SSADMv4, the preferred approach to large-scale information systems development in the UK in 1990s. The right side of the diagram shows how the CRAMM method is integrated and interacted with SSADM, adding security risk assessment. There are two review (inspection) loops alongside the software development process. The first loop occurs at the early stage of SDLC. During the review, the inspector will take initial requirements and business options, and his reviews will be considered in the stage of technical analysis and design. The second loop is when the system is designed. The inspector will review the technical design and set countermeasures which will be considered in both conceptual and physical design stage. Since the third-generation of security development method, software practitioners have emphasised the importance of security analysis at the stage of design.

### Observation

Baskerville provided not only a taxonomy to classify methods developed to date, but also foresaw the subsequent emergence of methods that embraced increased level of

[2]Structured Systems Analysis and Design Method
[3]CCTA's Risk Analysis and Management Methodology

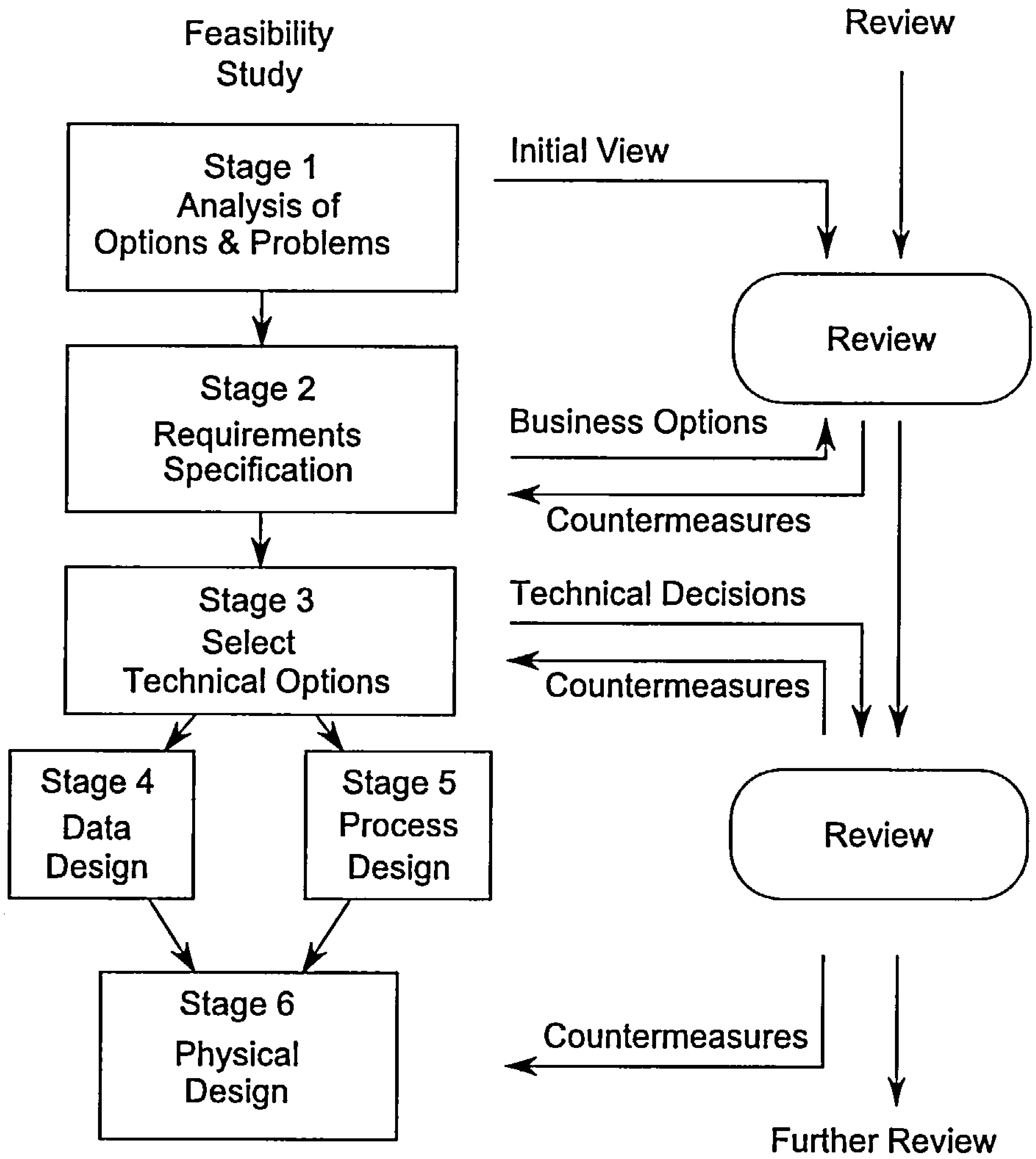



**Figure 2.6:** Overview of SSADM-CRAMM Interface [1]

abstraction. In this sense, agile security approaches are closely related to Baskerville's third generation, because the major practices in these approaches are concerned with designing an abstract model and modelling essential security attributes.

Plan-driven development methods cannot be easily adapted to be agile, because they are inflexibly-based, on a sequential, staged development akin to Royce's waterfall life cycle. However, plan-driven methods show how security is traditionally integrated into the SDLC, which is also required in agile security approaches.

### 2.2.2 International Security Evaluation Criteria

The mechanistic approach of the plan-driven methods considered in the previous section gives a verifiable development process for operational qualities such as security — in the sense that it can be shown that the development process has been correctly followed. However, these methods rely on review rather than more systematic testing and evaluation. Security evaluation criteria add to a verifiable plan-driven process the capability to identify the required level of security assurance and verify that the development process has met the security requirements.

Security evaluation criteria focus on security at the *technology* level, and derive from the need for evaluation of information system security in military and intelligence organisations. Various evaluation criteria and security guideline documents have been developed by governments and international organisation, in co-operation with industry. Figure 2.7 gives a chronological view of European and North American criteria.

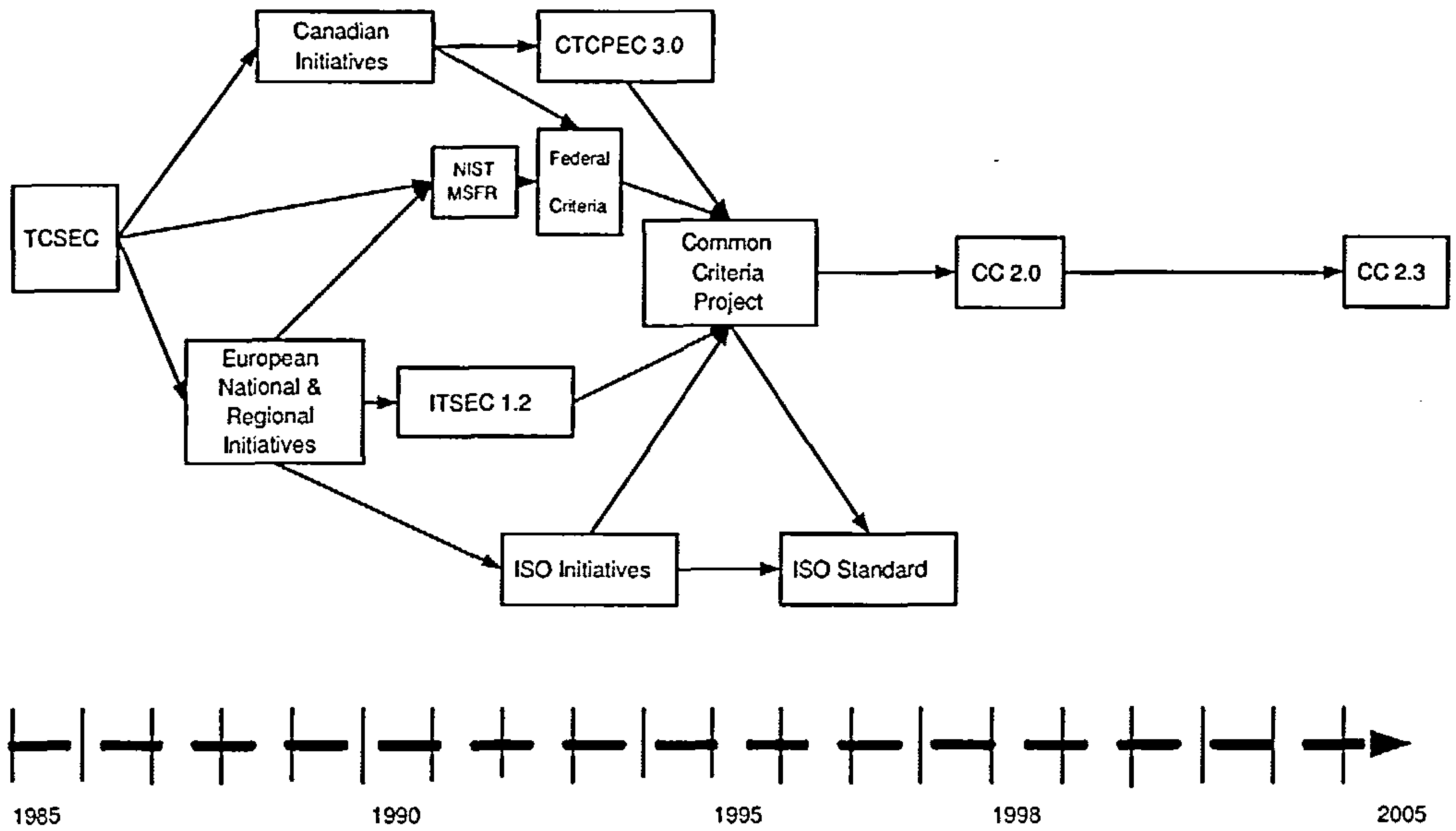



**Figure 2.7:** History of Security Evaluation Criteria

The following will review three of the most important evaluation standards: Trusted Computer System Evaluation Criteria (TCSEC, 1983-) [8], Information Technology Security Evaluation Criteria (ITSEC, 1990-) [10], and Common Criteria for Information Technology Security Evaluation (Common Criteria or CC, 1999-) [9,55].

### Trusted Computer System Evaluation Criteria (TCSEC)

TCSEC sets basic requirements for assessing the effectiveness of the computer security controls built into an information system. TCSEC was devised for the U.S. Department of Defense (DoD), for evaluation of information systems being considered for the processing, storage and retrieval of sensitive or classified information.

TCSEC, or the *Orange Book*, is the centrepiece of the DoD Rainbow Series. It was issued by the National Computer Security Center (NCSC) (an arm of the National Security Agency) in 1983, and updated in 1985, before being replaced by the Common Criteria (below).

TCSEC defines four fundamental *objectives* (requirements): Policy, Accountability, Assurance, and Documentation. TCSEC also defines four security levels or *divisions*: D, C, B and A. Division A is the highest security level; division D represents the situation where there are no security controls. Each division represents a significant difference in the trust that an individual or organization could place on the evaluated system. Divisions are broken into *classes*: C1, C2, B1, B2, B3 and A1. Each division and class expands or modifies the requirements of the immediately prior division or class. Table 2.2 summarises the TCSEC requirements of security classes C, B and A.

In Table 2.2, requirements are listed under the four objectives (Policy, Accountability, Assurance, and Documentation). The cells, read right to left, show the increasingly-stringent requirements of each class, using the following notation:

- / indicates that the requirement does not apply to a class.
- + indicates that a class requires something more than the next lower class, that there is something **new** under this requirement.
- Δ indicates that something is changed in this class. Something is **different**.
- - indicates that there is nothing extra under this requirement in this class, relative to the next lower class. The requirements are as **same** as the next lower class.

Applying TCSEC is reasonably easy because it explicitly sets out the requirements for each security class.

### Information Technology Security Evaluation Criteria (ITSEC)

ITSEC can be thought of as the European counterpart to TCSEC. It is based on earlier security criteria from France, Germany, the Netherlands and the United Kingdom. The first version of ITSEC, published in 1990, was subject to extensive international review, and replaced in 1991 by ITSECv1.2. The CEC (Commission of the European Communities) promoted it for operational use in evaluation and certification schemes.

ITSEC [10] is a structured set of criteria for evaluating computer security within products and systems. The product or system being evaluated, called the *target of evaluation (TOE)*, is subjected to a detailed examination of its security features, after which it is subjected to functional and penetration testing.

Unlike TCSEC, ITSEC assurance levels do not require evaluated targets to contain specific technical features. For example, an ITSEC TOE might provide authentication or

| | | | A1 | B3 | B2 | B1 | C2 | C1 |
|---|---|---|---|---|---|---|---|---|
| Security Policy | Discretionary Access Control | | - | Δ+ | - | - | Δ+ | + |
| | Object Reuse | | - | - | - | - | + | / |
| | Mandatory Access Control | | - | - | Δ | + | / | / |
| | Labels | | - | - | Δ | + | / | / |
| | | Device Labels | - | - | + | / | / | / |
| | | Label Integrity | - | - | - | + | / | / |
| | | Exportation of Labeled information | - | - | - | + | / | / |
| | | Exportation of Multilevel devices | - | - | - | + | / | / |
| | | Exportation of single-level devices | - | - | - | + | / | / |
| | | Labeling Human-readable output | - | - | - | + | / | / |
| | | Subject Sensitivity Labels | - | - | + | / | / | / |
| Acc-ability | Identification and Authentication | | - | - | - | Δ | + | + |
| | Audit | | - | + | + | Δ+ | + | / |
| | Trusted Path | | - | Δ | + | / | / | / |
| Assurance | Operational | Identification and Authentication | - | + | + | + | + | + |
| | | System Integrity | - | - | - | - | - | + |
| | | Covert Channel Analysis | + | Δ | + | / | / | / |
| | | Trusted Facility Management | - | + | + | / | / | / |
| | | Trusted Recovery | - | + | / | / | / | / |
| | Life-cycle | Security Testing | Δ+ | Δ+ | Δ+ | + | + | + |
| | | Design Specification and Verification | Δ+ | + | Δ+ | + | / | / |
| | | Configuration Management | Δ+ | - | + | / | / | / |
| | | Trusted Distribution | + | / | / | / | / | / |
| Documentation | Security Features User's Guide | | - | - | - | - | - | + |
| | Trusted Facility Manual | | - | + | + | + | + | + |
| | Test Documentation | | + | - | + | - | - | + |
| | Design Documentation | | Δ+ | + | Δ+ | + | - | + |

**Table 2.2:** Security Requirements in TCSEC (derived from [8, 9])

integrity features without providing confidentiality or availability. A TOE's security features are documented in a Security Target document, whose contents must be evaluated and approved before the target itself is evaluated. Each ITSEC evaluation is based exclusively on verifying the security features identified in the Security Target.

The Security Target document describes the security functionality offered by the TOE, along with a description of the environment that the TOE is intended to operate in. In the case of a system, the Security Target contains a System Security Policy (rules of operation tailored to a specific operating environment).

ITSEC separates functionality and assurance. There are six assurance evaluation levels, E1 to E6. (E0 can describe a system with no security assurance.) The higher the level, the more detail and rigour are required in the deliverable. Functionality is assessed separately, under levels F-C1 to F-B3 that correspond to the TCSEC levels, and five additional levels that relate to different classes of system (e.g. high integrity; high availability).

Evaluation takes place under headings such as effectiveness and correctness, where each evaluation produces a deliverable that demonstrates the level attained by the TOE. Evaluation comprises the same requirements for any of the six evaluation levels.

The effectiveness evaluation assesses the TOE directly. *Suitability Analysis* demonstrates that the security functionality of the TOE is capable of satisfying the security claims. The *Binding Analysis* demonstrates that the security functions of the TOE are combine to satisfy the security claims of the TOE. The *Ease of Use Analysis* demonstrates that it is not possible to operate the TOE in an insecure manner whilst believing it to be operating securely. The *Construction Vulnerabilities Analysis* investigates vulnerabilities in constructing the TOE, and the *Operational Vulnerabilities Analysis* investigates vulnerabilities in operating the TOE.

The correctness evaluation relates to the process of development. *Requirements* relates to the first phase of development, here the production of a Security Target document. The *Architectural Design* is a top-level design document identifying the basic structure of the TOE, its external interfaces and its major hardware and software components. In particular, it must describe the separation between security-enforcing and other components. The *Detailed Design* is a refinement of the Architectural Design, to the level of detail that can be used as a basis for implementation, and must identify all security enforcing components. The *Implementation* deliverable tests that the security claims refined in the Detailed Design are implemented correctly. It also includes full source code and hardware drawings of the TOE.

Additional correctness deliverable include the Development Environment deliverable (including Configuration Control, Programming Languages and Compilers and Developers Security).

Whilst ITSEC assurance evaluation is rather different to the requirements of TCSEC classes, it is possible to map between classes and levels, as in Table 2.3. No mapping is possible for the five additional ITSEC levels that relate to class of system.

| ITSEC Criteria | | TCSEC Class |
|---|---|---|
| E0 | <—> | D |
| F-C1, E1 | <—> | C1 |
| F-C2, E2 | <—> | C2 |
| F-B1, E3 | <—> | B1 |
| F-B2, E4 | <—> | B2 |
| F-B3, E5 | <—> | B3 |
| F-B3, E6 | <—> | A1 |

**Table 2.3:** Relationship of ITSEC (functional and assurance levels) to TCSEC [10]

ITSEC is more thorough than TCSEC, considering functionality as well as security assurance, development process as well as product. However, this considerably adds to the complexity of the evaluation process.

Like TCSEC, ITSEC has been largely replaced by the Common Criteria, which provides similarly defined evaluation levels and implements the target of evaluation concept and the Security Target document.

### Common Criteria for Information Technology Security Evaluation (Common Criteria or CC)

The Common Criteria represent international standards for computer security. (CCv2.1 [9] is ISO/IEC 15408; CCv2.5 [55] is ISO/IEC 18405). Unlike TCSEC, but more like ITSEC, the Common Criteria describe a framework in which computer system users can specify their security requirements; enable vendors to implement, and make claims about, the security attributes of their products; and allow testing laboratories to evaluate products to determine whether they actually meet the claims. In other words, the Common Criteria provides assurance that the process of specification, implementation and evaluation of a computer security product has been conducted in a rigorous and standard manner.

Like ITSEC, evaluation of CC requires the Security Target document, which is the key to any evaluation.

A security target is the basis for agreement between all parties as to what security the product offers. The Security Target document contains:

- A description of the security functionality offered by the product, along with a description of the environment that the product is intended to operate in.

- A set of security requirements: a security target permits the expression of security requirements for a specific product that are shown by evaluation to be useful and effective in meeting the identified objectives.
- The product summary specification, together with the security requirements and objectives, and the rationale for each.

The security requirements can be stated explicitly, or directly by reference to CC functional or assurance components, or less directly by reference to a *Protection Profile.* A protection profile could be developed by user communities, IT product developers, or other parties interested in defining such a common set of requirements. A protection profile gives consumers a means of referring to a specific set of security needs and facilitates future evaluation against those needs. A protection profile is intended to be reusable and to define product requirements that are known to be useful and effective in meeting the identified objectives, both for functions and assurance.

Common Criteria defines eight levels at which information system security can be evaluated and compared. The levels represent different sets of functional security requirements and hierarchical levels of assurance. The primary goal of using these criteria is to demonstrate that the system fulfils certain requirements regarding its protection mechanisms and that the correctness of the implementation meets a certain *Evaluation Assurance Level (EAL).*

The seven EALs define the rigour that must be applied in the development and presentation of the product. In addition, EAL0 indicates failure of evaluation.

- EAL1 - functionally tested
- EAL2 - structurally tested
- EAL3 - methodically tested and checked
- EAL4 - methodically designed, tested, and reviewed
- EAL5 - semi-formally designed and tested
- EAL6 - semi-formally verified design and tested
- EAL7 - formally verified design and tested

As for other security evaluation criteria, the Common Criteria set requirements for the documentation, in categories such as configuration management, delivery and operation, development (functional specification, design, and source code), guidance documents (user manuals), life cycle support, tests, and vulnerability assessment.

Unlike ITSEC, Common Criteria does not specify any particular process or sequence of system development. Instead *"the development is a refinement process of the security requirements into a TOE summary specification expressed in the security target. Each lower level of refinement represents a design decomposition with additional design detail. The final representation is the TOE implementation itself."* [9].

In addition, there should be *sufficient* design representations presented at a *sufficient* level of granularity to demonstrate where required [9]:

**a)** *"that each refinement level is a complete instantiation of the higher levels (i.e. all TOE security functions, properties, and behaviour defined at the higher level of abstraction must be demonstrably present in the lower level);"*

**b)** *"that each refinement level is an accurate instantiation of the higher levels (i.e. there should be no TOE security functions, properties, and behaviour defined at the lower level of abstraction that are not required by the higher level)."*

### Observations

The international security criteria, like the development methods considered in section 2.2.1, show an increasing maturity over the last 20 years. However, they are also increasing in size and the difficulty of application.

Security criteria are not only a means of evaluating a system, but also of assistance in describing the target or attained security of a system in a standardised way.

In summary, these security criteria have several common features:

- System life cycle. In the criteria, the way the TOE is developed is as important as the way it is operated. The criteria thus define two major stages of an SDLC, construction and operation. In particular, ITSEC explicitly refines the construction stage in various phases: requirement, architecture design, detailed design and implementation.
- Environment. All the criteria take the development environment and the operation environment into account. The requirements of the environment are stated for each security level, and environmental issues (such as security policies or rules of operation tailored to a specific operating environment) must be documented in the security target document.
- Documentation. The criteria require a suite of documents, including evaluation documents, design documents, testing documents and operation documents. Among these documents, the Security Target document is key to any evaluation. Other documents provide support and evidence for security evaluation.

These common features are significant for the security development, and will influence how agile and security software development are integrated. Each element is now considered further.

**System Life Cycle** All the criteria regard the system life cycle (or, equivalently, SDLC) to be linear and sequential, assuming a top-down development approach. In [18], NIST specifies five basic phases of this model:

- Initiation. The need for a system is expressed and the purpose and scope of the IT system is documented.
- Development/Acquisition. The system is designed, purchased, programmed, developed, or otherwise constructed.
- Implementation. The system security features should be configured, enabled, tested, and verified.
- Operation/maintenance. The system is executed. Typically, the system is being modified on an ongoing basis through the addition of hardware and software and by changes to organisational processes, policies, and procedures.
- Disposal. This phase may involve the disposition of information, hardware, and software. Activities may including moving, archiving, discarding, or destroying information, hardware, or software.

This linear development model, another variant of the Waterfall model, is similar to that which underpins the development methods considered in section 2.2.1. The recognition in ITSEC that security development divides into construction and operation is analogous to the stages of development and operation identified in considering qualities of the security development process in section 2.1.2. However, as with the plan-driven development methods, it is not immediately obvious whether the security assurance requirements of a linear process model are compatible with agile development.

**Environment** The environment includes the development and operation environment. This is important to security, as system security may be classified differently for use in different environment. The potential environment, including development and operation, of system should be considered when the development process starts.

The demands of specifying an operational environment promotes advanced preparation of an overview (or plan), and it is easy to understand the importance of architecture design (explicitly required in ITSEC). Again, this maps closely the approach of the plan-driven methods, but poses problems of compatible with agile development.

**Documentation** The evaluation process is a documentation-oriented process, and requires the development process to produce appropriate documents. From the criteria, the higher security level the system, the more documents are required.

The demands of documentation go well with plan-driven methods such as SSADM/CRAMM, which were devised to furnish managers with evidence of project progress. However, extensive documentation makes iterative development difficult — because of the constant updating of documentation that is entailed. It is also contrary to the lightweight approach of agile development.

### 2.2.3 Risk Assessment

The previous section considered existing evaluation criteria. However, a key element in security development is finding the appropriate level of compromise between the costs and benefits of security. Risk management is a traditional approach to such analysis.

*Risk management* has been *de rigeur* in many mature engineering disciplines for at least a century [31]. In addition to construction and other engineering, it has been applied widely in planning financial strategy.

In software engineering, risk management can be seen as *"the process of managing risks to agency operations (including mission, functions, image, or reputation), agency assets, or individuals resulting from the operation of an information system. It includes risk assessment; cost-benefit analysis; the selection, implementation, and assessment of security controls; and the formal authorisation to operate the system."* [56] The risk management process considers effectiveness, efficiency, and constraints due to laws, directives, policies, or regulations. *Risk Assessment* refers to *"the process of identifying risks to agency operations (including mission, functions, image, or reputation), agency assets, or individuals by determining the probability of occurrence, the resulting impact, and additional security controls that would mitigate this impact."* [56]. Risk assessment also incorporates threat and vulnerability analyses. The value of risk assessment is that it provides decision criteria for security requirements in their business and social context.

Many documents, such as [18, 57], suggest that risk assessment is an essential activity for information systems security. Security in a software system should be commensurate with risk. However, the process of determining which security controls are appropriate and cost effective is often a complex and subjective matter. An objective of security risk assessment is to put this process onto a more objective basis. Risk assessment is widely regarded as the only viable method of providing a cost-benefit justification for security controls, and is the basis of many standards for information security management [32, 38, 39]. It is even regarded as a rational approach to broader security choices in society [58].

Security risk assessment is a family of security analysis methods, of which some are commercially-oriented (e.g, STRIDE [2, 57, 59], CRAMM [60], ACSM/SAR [61] and Cigital Framework [54]), and some are standards-based (e.g, ASSET [62], OCTAVE [39], and COBIT [63]).

Whilst some risk analysis methods attempt to calculate a nominal value for an information asset and determine risk as a function of loss and event probability, others rely on checklists of threats and vulnerabilities to determine a basic risk measurement (details in [1]). Whilst the methods have different features, they almost all share common practices, and have a common set of concepts that should be considered in any risk analysis method. These are given explicit definitions:

- **Asset**: the object of protection efforts, variously defined as a system component, data, or even a complete system.
- **Impact**: the magnitude of the loss that can be expected to result if a threat is realised. This can be monetary or tied to reputation; it may result from a breach of a law, regulation, or contract. Without quantification of impact, technical vulnerability is hard to deal with, especially when it comes to mitigation activities.
- **Probability**: the likelihood that a given event will be triggered. Whilst probability may be quantified, the calculation of probability is often extremely rough, and more suited to gross measures such as high (H), medium (M), and low (L).
- **Risk**: the level of impact on operations (including mission, functions, image, or reputation), assets, or individuals resulting from the operation of an information system given the potential impact of a threat and the likelihood of that threat occurring.
- **Threat**: any circumstance or event with the potential to adversely impact operations (including mission, functions, image, or reputation), assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.
- **Vulnerability**: for a threat to be effective, it must act against a vulnerability in the system. In general, a vulnerability is a defect or weakness in system security procedures, design, implementation, or internal controls that can result in a security breach or a violation of security policy [64]. In software, vulnerabilities come in two basic flavours: flaws are design-level problems leading to security risk, and bugs are implementation-level problems leading to security risk. Automated source code analysis tools tend to focus on bugs. Human expertise is normally required to uncover flaws.
- **Countermeasures** and **Safeguards**: the management, operational, and technical controls prescribed for an information system to adequately protect the confidentiality, integrity, and availability of the system and its information. For every risk, controls can be applied to prevent, reduce or (at a minimum) detect the risk when it triggers.

It is commonly accepted that there are common activities in risk analysis, though these come under different labels in different approaches [2, 65, 18, 32]. The major activities can be characterised as follows.

- Applications are built from individual features and every feature potentially can be attacked, so it is important to learn as much as possible about the target of analysis — its components and how they are connected:
  - Read and understand the requirements specification documents, architecture, and other design materials.
  - Discuss and brainstorm about the target.
  - Determine the system boundary and data sensitivity/criticality.
  - Use the software (if it exists in executable form).

- Study the code and other software artifacts (including using code analysis tools).
- Identify threats and agree on relevant sources of attack.

- Analysis of application structure is not done only to determine how everything works, but also to investigate the components and assets, and how data flows between the components. The components or assets are threat targets, and they are also the objectives of security protection. Discussion of security issues surrounding the application is instructive:
  - Argue about how the product works and determine areas of disagreement or ambiguity.
  - Identify possible vulnerabilities, making use of tools, lists of common vulnerabilities, etc.
  - Map out possible ways that vulnerabilities can be exploited and begin to discuss possible fixes.
  - Gain understanding of current and planned security controls.
- When the threat targets are known, it is natural to examine how likely the vulnerabilities are targeted. Determine the probability that an attack can compromise security:
  - Map out attack scenarios for exploitation of vulnerabilities.
  - Balance controls against threat capacity to determine likelihood.
- When planning the security protection, it is necessary to assess the likely impact if the attack to a threat target succeeds. Perform impact analysis:
  - Determine impacts on assets and business goals.
  - Consider impacts on the security posture.
- Once analysis is complete, the developers need to determine the most important threats, and work on first determining the risk that a threat poses. The method used to calculate risk is not important so long as it is realistic and consistent. A simple way to calculate risk is to multiply the potential impact of an attack by the likelihood of the vulnerability occurring, $risk = probability \times impact$.
- Develop a mitigation strategy. A higher-ranking risk means the threat poses a greater overall risk to the system. According to the risk ranking, a plan is developed to mitigate the risks, and countermeasures recommended.
- At the end of risk assessment process, a document about threats, risks and the mitigation plan is always produced:
  - Carefully describe the major and minor risks, with attention to impact.
  - Provide basic information regarding where to target (limited) mitigation resources.

Risk assessment can be applied in every phase of SDLC. For instance, Table 2.4 summarises security considerations (introduced in [18]) and guidance for integration of risk management with SDLC (from [32]).

| SDLC Phases | |
| --- | --- |
| **Security Consideration** | **Support from RM Activities** |
| Initiation | |
| Security Categorisation<br>Preliminary Risk Assessment | Identification of risks is used to support the development, including security requirements, and a security concept of operations (strategy) |
| Development | |
| Risk Assessment<br>Security Functional Requirement Analysis<br>Security Assurance Requirements Analysis<br>Cost Considerations and Reporting<br>Security Planning<br>Security Control Development<br>Developmental Security Test and Evaluation<br>Other Planning Components | The risks identified during this phase can be used to support the security analyses of the system that may lead to architecture and design trade-offs during system development |
| Implementation | |
| Inspection and Acceptance<br>System Integration<br>Security Certification<br>Security Accreditation | The risk management process supports the assessment of the system implementation against its requirements and within its modeled operational environment. Decisions regarding risks identified must be made prior to system operation |
| Operation/Maintenance | |
| Configuration Management and Control<br>Continuous Monitoring | Risk Management activities are performed for periodic system reauthorisation (or re-accreditation) or whenever major changes are made to the system in its operational production environment (e.g. new system interfaces) |

Continued on next page

| | Continued from previous page |
|---|---|
| Decommissioning | |
| Information Preservation<br>Media Sanitation<br>Hardware and Software Disposal | Risk management activities are performed for system components that will be disposed of or replaced to ensure that the hardware and software are properly disposed of, that residual data is appropriately handled, and that system migration is conducted in a secure and systematic manner |

**Table 2.4: Security in the SDLC (derived from [18,32])**

### Observations

Risk assessment is generally a favoured security engineering technique, to be applied to a whole system (because security is a system wide issue). However, some research suggests that a risk analysis process can and should be applied in the early stages of the development. For example, [18] recommends that risk assessment is considered when designing a system, and also that a preliminary risk assessment can be performed when analysing requirements; [66] also states that risk assessment at the requirements stage can help to improve the quality of requirements.

The interdependency between vulnerability and architecture shows that the architecture of a system or an application is important to the security development. Performance of risk analysis on the architecture is at the heart of the security development, whether a project is applying an agile method or a traditional development method. For instance, Figure 2.8 is a fragment of a typical sequence of phases in software development, with the addition of pointers to the development of the security requirements. This suggests that developers should address security concerns in parallel with the analysis of the functional requirements and architecture design — first consult the security policies and security guidelines, then produce the architectural design and the overall model (including a security architecture). Furthermore, architectural risk analysis can help develop the specification of security requirements.

Architectural risk assessment should be repeated when the design of a system is modified. When performing architectural risk analysis, the object of the analysis task is not the whole system (software, hardware, network, and environment), but the design models.

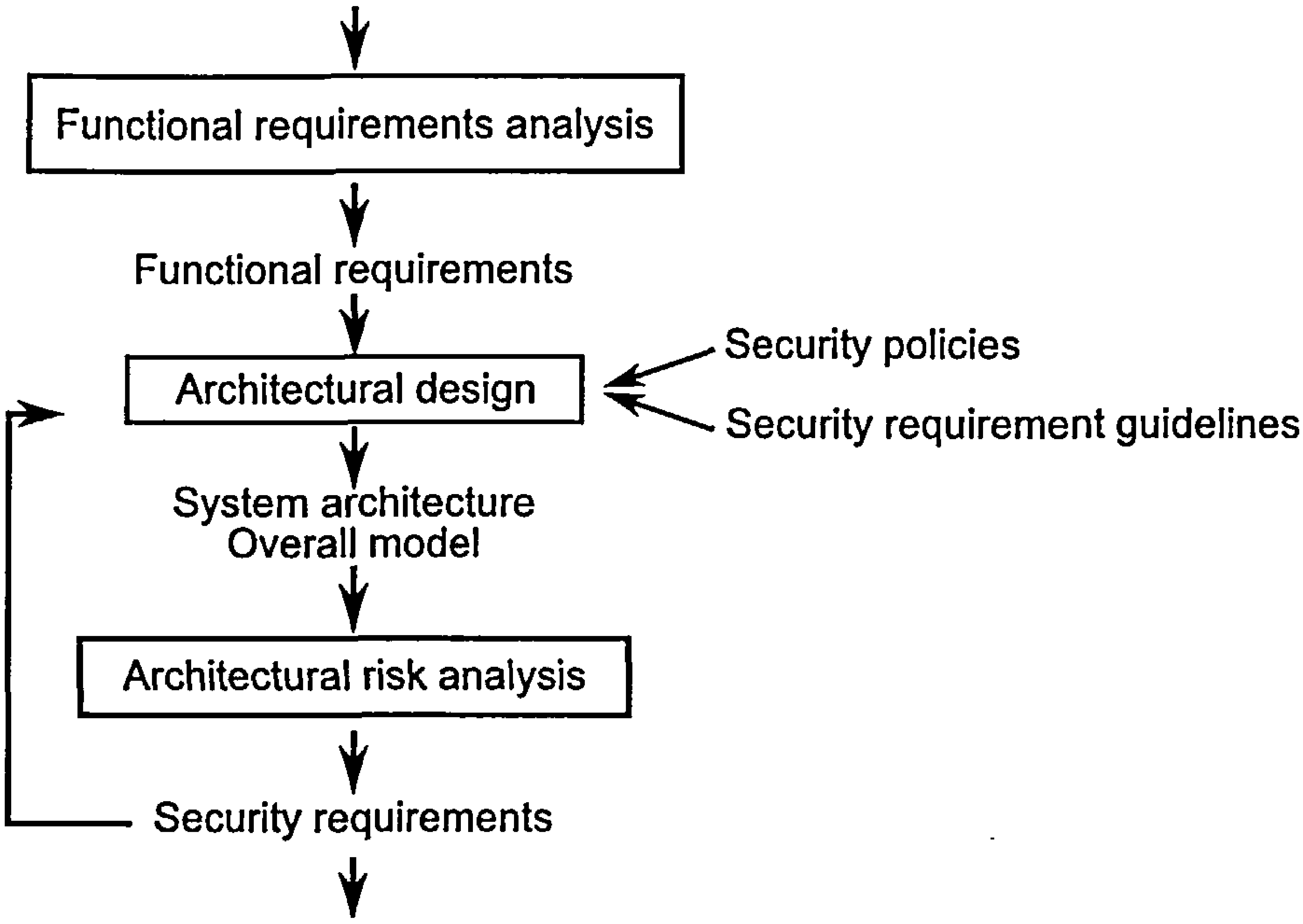



**Figure 2.8:** Security Analysis and Security Requirements

Risk assessment usually requires the customer's contributions for ranking risks. Besides changes in functionality, the analysis also has to be repeated when new threats, vulnerabilities, and attacks are made public. The adaptive nature of risk analysis does not depend on the process or methodology in which it is carried out. It always depends on the design and modelling process performed before the risk analysis.

### 2.2.4 Other security artifacts

This chapter has concentrated on existing methods of developing and evaluating secure information systems, and of analysing and managing risk in such systems. This thesis concentrates on development and assurance of security. However, there are many other security artifacts and activities, that are outside the scope of the thesis, but can contribute to security development. Two groups of artifacts, architectural modelling and best practice guidelines, are discussed briefly in this section.

#### Architectural Modelling

From a software engineering point of view, the goal of analysis is to determine the functionality of a system according to the identified requirements. From a security point of view, analysis also serves to identify threats, attacks, and vulnerabilities. Furthermore, expected risks have to be identified, and priorities have to be assigned to them. There are several security engineering approaches for modelling threats, attacks, and vulnerabilities, such as goal trees for security risk assessment. Furthermore, semi-formal and formal modelling approaches have been adapted to security analysis.

**Goal Trees** Various techniques with symbolic tree representations have been developed in software engineering (e.g. KAOS [67, 68]). Each node represents a specific goal that has to be reached. The overall goal is then decomposed into sub-goals. There are a variety of ways to achieve the high level goal. Goal trees, such as threat tree [39,69] and attack tree [70], are based on *AND/OR* trees, and admit basic quantification: values such as the cost of a particular attack can be assigned to the nodes and propagated up to the root. Another application of goal trees is to identify which attack would be the most likely. Furthermore, goal trees facilitate "what if" experiments, e.g. what would happen if you have more or less budget than expected.

**Semi-formal Modelling Techniques** Semi-formal modelling techniques (those involving diagrams with defined syntax but limited formal semantics) can be used to specify security concerns. Approaches including data, function, and aspect oriented modelling techniques. They are characterised by different graphical elements but also linguistic elements. For instance, graphic modelling techniques include Entity Relationship Method (ER) [71], and Unified Modelling Language (UML) [72, 73]. UML can be made more precise by adding constraints in the Object Constraint Language (OCL) as part of the model.

UMLsec [74, 75] and secureUML [76, 77] are two approaches that extend the security potential of UML models. UML already provides the capability to describe some kinds of security concerns informally: class diagrams offer the possibility to describe a security role, its characteristics and relations between roles; State-charts can be used to describe the behaviour of elements or roles; Activity, Sequence, and Collaboration diagrams are used to describe the cooperation of the different elements of the system. Based on this, UMLsec and secureUML define UML profiles to express particular security concerns, such as an access control model.

At the requirements stage, approaches based on use cases, such as Abuse case modelling [78], can promote a better understanding of system security among stakeholders. A typical abuse case includes a description of each instance of potential abuse and the range of privileges that might be abused and the impact that will result.

**Formal Methods** Security is a popular topic in the formal methods community. Formal specification, analysis, and proof of properties have been applied to security protocols of authentication, fair exchange, electronic commerce, and electronic actions, for example [79,80]. Experience shows that formal methods are powerful tools to verify the requirements, but that not all requirements can be specified formally [81]. During the process of development, formal methods can provide additional assurance of the trustworthiness of a system. The benefits which a formal method can provide were listed in [82]. However, formal models may give a misleading impression of security, omitting environmental aspects including operational context; furthermore, security properties are not usually preserved by formal refinement, and must be reproved at each stage of a formal development.

### Best Practices and Guidelines

Various organisations publish general guidelines that can be used when reviewing the architecture and design from a security perspective (for instance [2,65,57,83]). There are also guidelines for programming secure systems (such as [84,85]). Some organisations offer security services and advice about improving the security based on a substantial experience, such as the CERT Coordination Centre (CERT/CC) and National Institute of Standards and Technology (NIST). There are many best practices and guidelines, and it is important to find the guidelines that suit a particular project. Once the best practice guidelines have been selected, the steps are generally straightforward. Typically, a developer becomes increasingly familiar with a particular approach by using the same guidelines several times. Problems occur, however, when the developer leaves the typical narrow focus of such security improvement documents or moves developing to a different type of system or in a different environment. By nature, the more concrete documents are more time-dependent and should be updated on a regular basis.

## 2.3 Conclusions

Information security is about protecting information and the information system from unauthorised access, disclosure, modification, and destruction. Two characteristics make security itself complicated: security is a system property; and security is a process.

In this chapter, two important security artefacts (international security evaluation criteria and risk assessment) were reviewed. International security evaluation criteria are very important because they make the security of information system evaluable and comparable in a standardised way. The common characteristics of the three best-known security criteria were summarised in this chapter. In these three criteria, SDLC is regarded as a linear model; environment is a very important factor which is considered in early stage of SDLC; and documentation is an essential requirement of security evaluation. Furthermore from these security evaluation criteria, the importance of system architecture is emphasised. The architecture is important because it encapsulates a set of significant decisions about the organisation of a software system, including [86]:

- Selection of the structural elements and their interfaces that make up a system,
- Behaviour as specified in collaborations among these elements,
- Composition of the structural and behavioural elements into larger subsystems,
- An architectural style that guides this organisation.

Risk assessment is a tool to balance security goals against other software project quality goals. Risk assessment is also a family of mature engineering methods. Security development is an exercise in risk management, consisting of identifying risks early, understanding the implications in light of experience, creating an architecture that addresses the risks, and rigorously testing the system for security.

Besides security criteria and risk assessment, there are many other security artefacts which are also useful and productive, and they are briefly discussed in this chapter.

The purpose of reviewing information security and security development is to establish a list which can help to select a candidate process from existing Agile methods. Baskerville's classification [1] concluded that security development methods should be integrated into more general software engineering methods. The agile security approach proposed in this thesis will be a third-generation method; i.e, a logic transformation method. Figure 2.9 shows a framework for a security development process, derived by abstracting from method-specific features of the SDLC of the third-generation SSADM-CRAMM interface.

Based on the analysis in this chapter, at least two criteria for an agile approach to security development can be concluded:

- the method should have a clear (explicit) development process, ideally comprising simple milestones so that the security review loops can be easily integrated (as in Figure 2.9).
- an architecture is essential, not only for the documentation requirement of security evaluation, but also for risk assessment. Therefore, it is better that a general blueprint of system can be produced at the early stage of SDLC so that the architecture can be captured.

In exploring an agile approach to security development, this thesis will focus on a particular kind of information system, namely Web applications. The next chapter provides a review of relevant material from this domain.

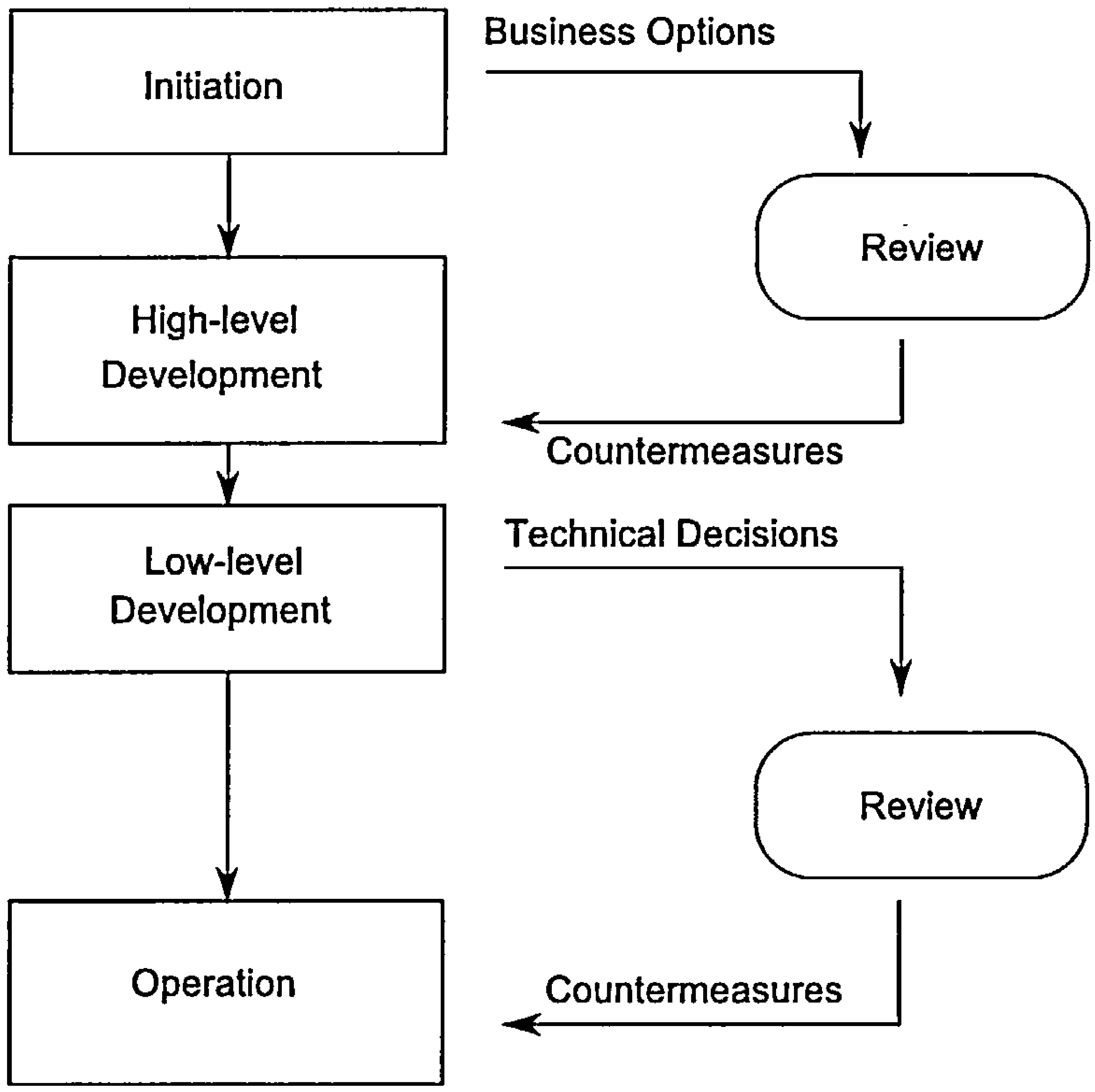



**Figure 2.9:** A Framework of Security Development Process

# 3 Web Application

*Web systems are a special kind of information system; they share a common three-tier architecture. A Web application is the kernel of a Web system. Based on knowledge of security engineering, the security issues of Web applications are reviewed in this chapter. The later sections of this chapter also explain why an Agile method is a suitable choice for building Web applications.*

## 3.1 Introduction

With the advance of the Internet, the Web is playing an increasingly important role in today's society. Modern organisations rely heavily on Web systems to facilitate their business processes, reduce costs of the process life-cycle, and manage resources of the organisation. On one hand, the systems running in an organisation need to be integrated for consolidated decision making, more accurate system information, and better performance and monitoring. On the other hand, integration *within* the organisation is not enough; systems are becoming service oriented. Relatively, perhaps the greatest advantage of Web systems is that they enable the integration of applications *across* organisation boundaries, providing fast and seamless collaboration with partners, customers, and suppliers. Web systems represent the engine allowing distributed organisations to communicate, to share information with other companies, customers and providers, and to manage production and distribution.

Computer scientists from both industry and academia have already been showing great interest in Web engineering. Publications include example general discussion [87, 88, 89], engineering development methods [90,91,92,93,94,95,96,97,98,99,100], reverse engineering [101, 102, 103, 104], and testing [105, 106]. Last but not least, there is also substantial published literature about security in Web engineering, for example [83,107,108,109,110,111,112] and some articles at [113,114].

Compared with information systems for client/server computing, the key reason for the success of Web systems in the business world is that Web systems are more open, flexible and adaptive [89]. Traditional information systems have an architecture of two tiers, but Web systems are typically constructed in a three-tiered architecture (an example is shown in Figure 3.1).

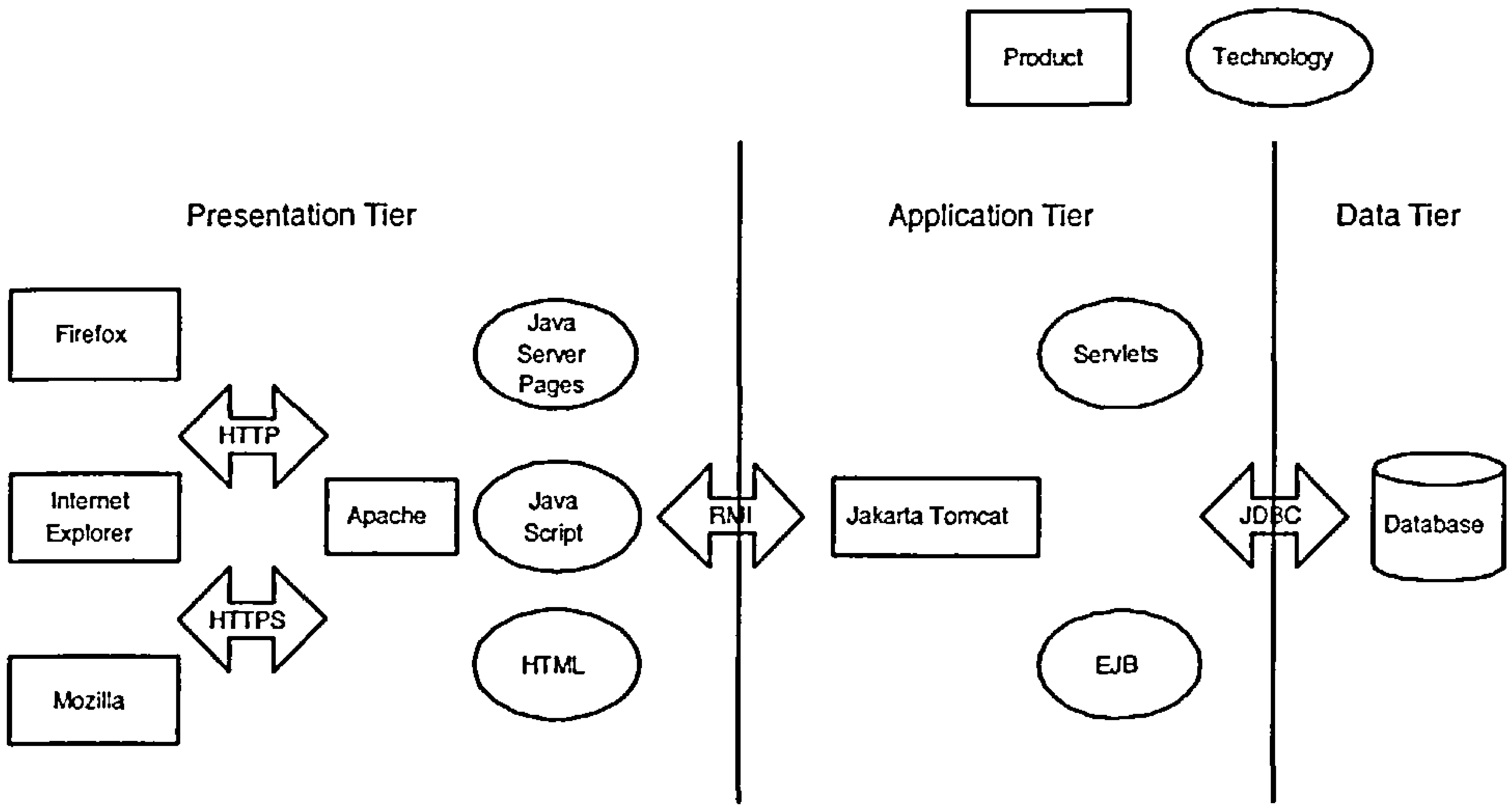



**Figure 3.1:** Typical Architecture of a Web application

Most commonly, a Web browser, e.g, Mozilla Firefox or Microsoft Internet Explorer, and a Web server, e.g, Apache, lie in the first tier, the *presentation tier*; an engine using some dynamic Web content technology, e.g, Enterprise Java Bean (EJB) and Java Servlets, is in the middle, the *application tier*; and normally a database management system (DBMS) with databases is in the third, the *data tier.*

The architecture is also vitally important for security: security requires the strategy of *Defence in Depth*. *Defence in Depth* is all about building a number of layers around the information that work together to provide a strong and (hopefully) impenetrable protection. The architecture plays an important role to allocate these layers.

## 3.2 Security of Web Systems

Security is a process: it does not make any sense to study the security of a Web application without mentioning the security of the entire Web system. This section discusses the general security issues associated a Web system.

Stories of security breaches often appear in daily newspapers, for example,

- *"The banking industry has warned customers with their on-line accounts to guard against a new wave of cyber fraud. Industry body APACS said some 2,000 British on-line account holders had been taken in by scams in the past year, losing £4.5m between them. Many were duped into revealing their account passwords by phony e-mails purporting to come from their bank. Others had their computers infected with programs which allow fraudsters to record their log-in details. ..."* — BBC News, 1 October 2004

- According to an APACS [115] report on 8 November 2005, the latest card fraud figures show Internet fraud accounts for a quarter of all losses. The figure of Internet fraud was £55.1m during the period of January to June 2004; and it became £58.0m from January to June 2005, increasing 5%.
- A survey in the Secure Software Forum [116] over 2005 stated that only 36% of interviewed enterprises have implemented a programme to educate development teams about secure coding practices, and 30% integrated a security assurance programme into their development process. It was concluded by the panel of corporate security executives, academics and professional software developers at the RSA Conference 2006 [26] that most businesses are not doing enough to ensure stakeholders security, and need to do better.

The security considerations for a Web system are shown in Figure 3.2. This diagram demonstrates that Web security is about information security of the entire system, relating to the network, hosts and system executing on the hosts. This includes Web applications and other software such as operating systems and database management systems. The application level security is at the top of Web security concerns. The security policies and procedures cross all layers of Web security.

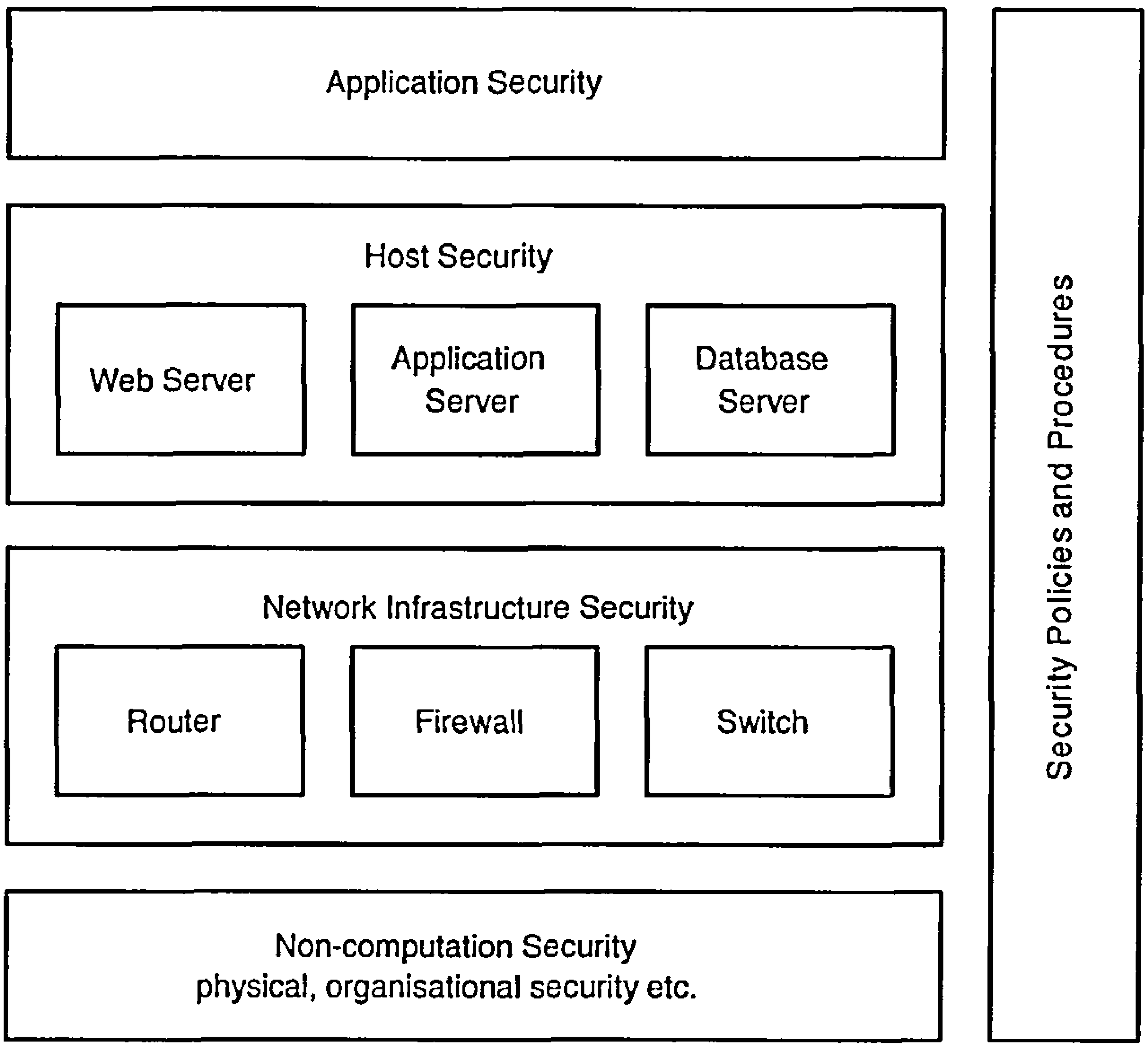



**Figure 3.2:** Security Considerations of Web Systems

The security of Web systems includes all concerns from network layer to application layer. All these concerns of Web systems are highly interdependent because of the independence of these layers. For the security of Web applications, the flaws and vulnerabilities from other layers of Web system continually impact the risks of Web applications.

For example, a security vulnerability of the network may cause an application to be insecure, so that users may bypass the application to have access to protected information.

To build a Web system that is secure against various attacks from hostile users, a holistic approach to security is required. It should be a layered and systematic approach, as illustrated in Figure 3.3.

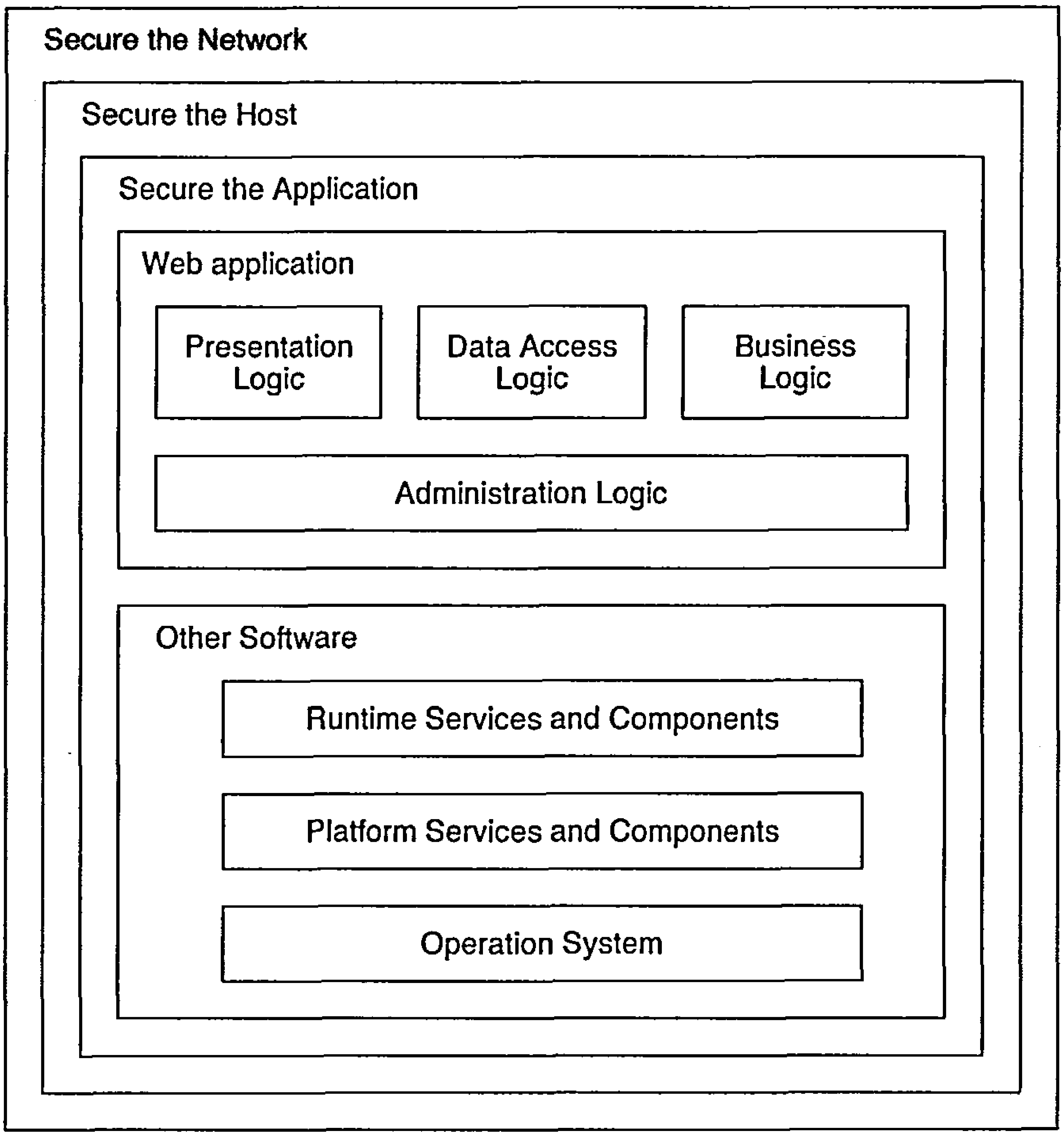



**Figure 3.3:** Holistic Approach to Web System Security

Figure 3.3 is an instance of *Defence in Depth* principle. Meanwhile, it also gives some details of Web application and other supporting software. There are four parts in the architecture of a Web application: presentation logic, data access, business logic, and administration.

There is a huge amount of literature about the security of Web systems, e.g, [83, 107, 108, 109, 110, 111, 112] and an on-line library at [113, 114]. One of the leading contributions is Guidelines on Securing Public Web Servers [83], which is a comprehensive handbook about Web server security. Inevitably, the security of Web applications has extensive coverage in all these documents.

Web application security may not be the weakest link in Web security, but improvements can be made. The security of a Web system is overall an engineering issue: assessing it needs contributions from all participants, from developers to managers; and it leans on every individual hardware and software component in the system (illustrated in Figure 3.3).

## 3.3 Security of Web Applications

Considering Figure 3.3 again, from bottom to top, there is network security, host security, and the security of supporting software. The security of a Web application is built on all of them. At the layer of supporting software, there are also three groups of software components and services: basic run-time services and components (e.g, printing, indexing), platform services and components (e.g, DBMS services), and the operating system (e.g, Windows system, Linux). This supporting software is also crucial to the security of Web applications. Such a systematic architecture of Web applications is both an advantage and a disadvantage; a systematic architecture makes it easier to establish security; but if the architecture is inadequate, a good overall security cannot be expected.

There are several comprehensive references about building secure Web applications. The Open Web Application Security Project (OWASP) and OASIS are two of the most active non-profit organisations. OWASP published a list of Top-Ten vulnerabilities of Web applications [117] and general guidance for building secure web applications [65]; OASIS proposed the Application Vulnerability Description Language (AVDL) [118] — a unified language to describe vulnerabilities. Meanwhile, other organisations have also made a contribution, such as technical reports from Microsoft [2, 57] which focus on developing secure Web applications using .NET framework; a report of security threats classification from Web Application Security Consortium (WASC) [119] which summarises the most common security threats of Web application; and a white paper from Defence Advanced Research Projects Agency (DARPA) [120] which provides security patterns of Web applications.

When a user executes a Web application, a trust relation is established between two parties both technologically and psychologically. Ideally, security should be about protecting both ends of the connection. But what is considered in this thesis is the protection of Web applications. The research in this thesis thus complements these references.

In Chapter 2, where I reviewed information security, the importance of the software architecture has been already addressed. The three-tier architecture is general and typical for Web systems; for a Web application, there are also some choices. The loose procedural architecture and model-view-controller architecture are two typical, and most widely used, architectures for Web applications. Now, these are discussed.

### 3.3.1 Typical Architecture of Web Applications

The three-tier architecture is widely adopted for building a modern Web system; largely, the Web application is located at the *application tier*. There are two popular choices of application architecture: loose procedural architecture and Model-View-Controller (MVC) architecture.

#### Loose Procedural Architecture

Web applications are made up of many loosely coupled components (i.e. Web pages and script files). In a loose procedural architecture, each component usually fulfils a complete function including interacting with users and performing a piece of business logic. The links among these components are always the URL links, presenting the logic of the business procedure. This is a logical way to develop the application following the process of functionality decomposition. This architecture is very common when the business logic of the application is simple.

Loose procedural architecture is popular because of its simplicity. But it is difficult to deliver some operational qualities, such as security and scalability, if these qualities are not considered before the programming starts. The main reason for this is that the architecture is not a layered architecture so it will be difficult to correct problems spread across all parts of the system, like security. Flaws in design or bugs in implementation may cost a lot to rectify. So it requires the developers to have substantial security knowledge to avoid errors.

#### Model-View-Controller Architecture

When the business logic of Web applications becomes complex, it is more difficult to implement and maintain. Using a scalable application architecture becomes beneficial. The Model-View-Control architecture is one such approach. It divides functionality among the objects involved in maintaining and presenting data to minimise the degree of coupling between the objects. The MVC architecture, which has its roots in Smalltalk [121], has been proposed for J2EE applications, and is well explained in most Java programming textbooks, for example [122]. Other languages, such as ASP.NET, can be considered a partial implementation of this approach. For PHP, the WACT project[1] aims to implement the MVC paradigm in a PHP friendly fashion.

[1] http://wact.sourceforge.net

The MVC architecture divides applications into three layers - model, view, and controller - and decouples their respective responsibilities. Each layer handles specific tasks and has specific responsibilities to the other layers. Figure 3.4 shows the typical architecture of Java Web application.

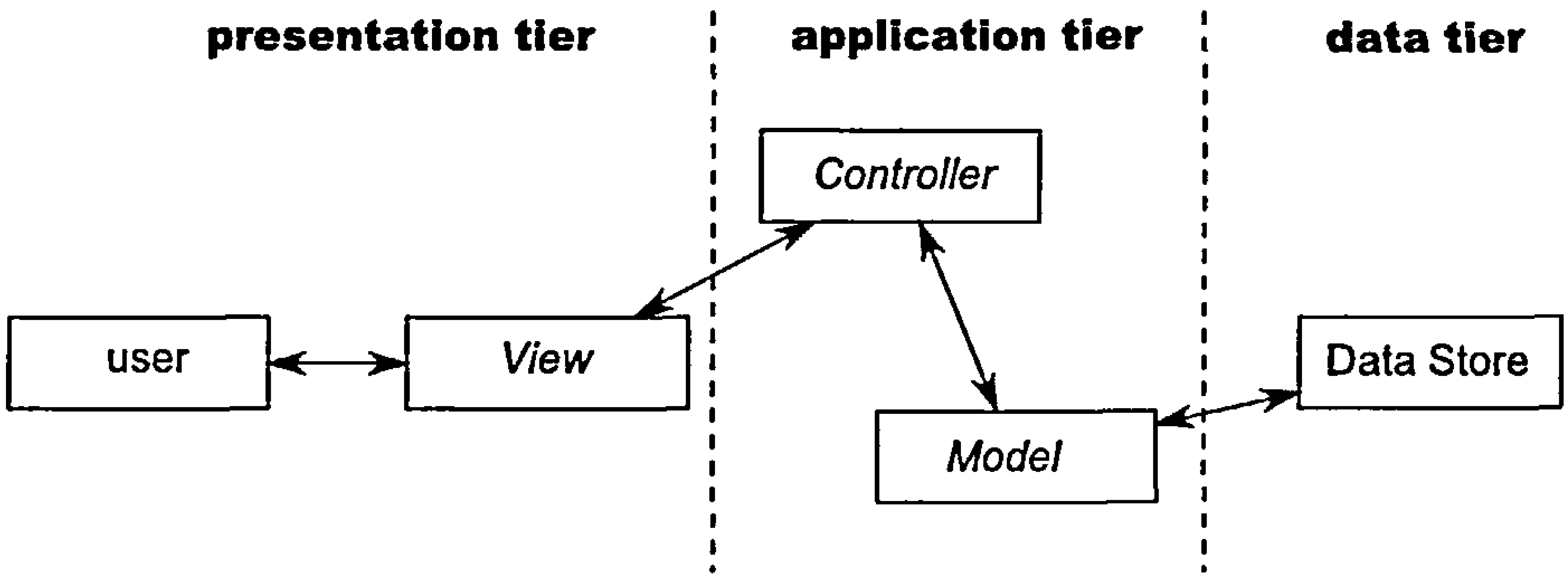



**Figure 3.4:** MVC Architecture of Web Applications

**Model** A model represents business data and business logic or operations that govern access and modification of this business data. Often the model serves as a software approximation to real-world functionality. The model notifies views when it changes and provides the ability for the view to query the model about its state. It also provides the ability for the controller to access application functionality encapsulated by the model.

Models encapsulate functionality, such as an account or user. A good model should be transparent to the controller, and provide a method to deal with high-level business processes rather than a thin connector to the data store. The idea is to encapsulate the actual "dirty" work into the model code, rather than exposing primitives. If the controller and model are on different machines, the performance difference between different design approaches will be staggering, so it is important for the model to be useful at a high level. The model is responsible for checking data against business rules, and any residual risks unique to the data store in use. For example, if a model stores data in a flat file, the code needs to check for OS (Operating System) injection commands if the flat files are named by the user. If the model stores data in an interpreted language, such as SQL, then the model is responsible for preventing SQL (Structured Query Language) injection. If it uses a message queue interface to a mainframe, the message queue data format (typically XML (eXtensible Markup Language)) needs to be well formed and compliant with a DTD (Document Type Definition). The contract between the controller and the model needs to be carefully considered to ensure that data is strongly typed, with reasonable structure (syntax), and appropriate length, whilst allowing flexibility to allow for internationalisation and future needs.

**View** A view renders the contents of a model. It accesses data from the model and specifies how that data should be presented. It updates the data presentation when the model changes. A view also forwards user input to a controller.

The code aims to produce the (HTML, XML, etc) output for the user with little to no application logic. As many applications will be internationalised (i.e, contain no localised strings or culture information in the presentation tier), they must call into the model (application logic) to obtain the data required to render useful information to the user in their preferred language and culture (such as time format, units, etc). All the user's input is directed back to controllers in the application tier.

**Controller** A controller defines application behaviour. It dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. In a stand-alone GUI client, user inputs include button clicks and menu selections. In a Web application, they are HTTP GET and POST requests to the Web tier. A controller selects the next view to display based on the user interactions and the outcome of the model operations. An application typically has one controller for each set of related functionality. Some applications use a separate controller for each client type, because view interaction and selection often vary between client types.

The controller takes inputs from the objects of view and dispatches them through various work-flows that call on the objects of the application model to retrieve, process, or store the data. A well written controller always validates inputs from clients before passing them to the *model*, and also ensure that the outputs from the *model* are safe for the *view*.

From a security point of view, MVC architecture is a better choice than loose procedural architecture because it is layered architecture which more easily supports a style where security problems of a Web application can be decomposed and mapped into layers. As a result, it can reduce the requirement of security awareness on the developers and the difficulty of maintaining the application.

### Selection of Application Architecture

There is no hard rule on selection of application architectures; the selection is purely based on the project because each architecture has its own advantages and drawbacks. In real projects, the developer chooses a suitable architecture. For very simple Web applications, a loose procedural architecture is the preferred choice because any perceived performance benefit from moving to a more scalable architecture will never be recovered during the development. For example, it perhaps takes an additional several weeks to refactor the scripts into an MVC architecture, but the end users will likely not notice the improvements in scalability. But the MVC approach typically results in a clean separation of presentation from content, leading to clear delineation of the roles and responsibilities of the developers and page designers on your programming team. In fact, the more complex the Web application, the greater the benefits of using the MVC architecture should be. The MVC architecture has been gradually adopted as a standard architecture of many JAVA enterprise applications [122].

Once an architecture is selected, identifying and addressing the security vulnerabilities in this architecture are next things to do in the developer's to-do list.

### 3.3.2 Web Application and Risk Analysis

It is very difficult to design and build a secure application without security awareness. Therefore, the security development process is concerned with risk analysis and implementing effective countermeasures to mitigate risks. Chapter 2 showed that risk assessment can be performed at any stage of development process. Performing risk assessment at a higher level (i.e, conceptual/architectural level) is an increasingly important and mature discipline for building security critical information systems (see [2, 18]).

The general process of architectural risk analysis is to analyse the application's architecture; identify potentially vulnerabilities that may allow a user, perhaps mistakenly, or an attacker with malicious intent, to compromise the application's security; and evaluate the risk of security breach; finally suggest the implementation of countermeasures. Figure 3.5 shows a simplified architectural risk assessment process.

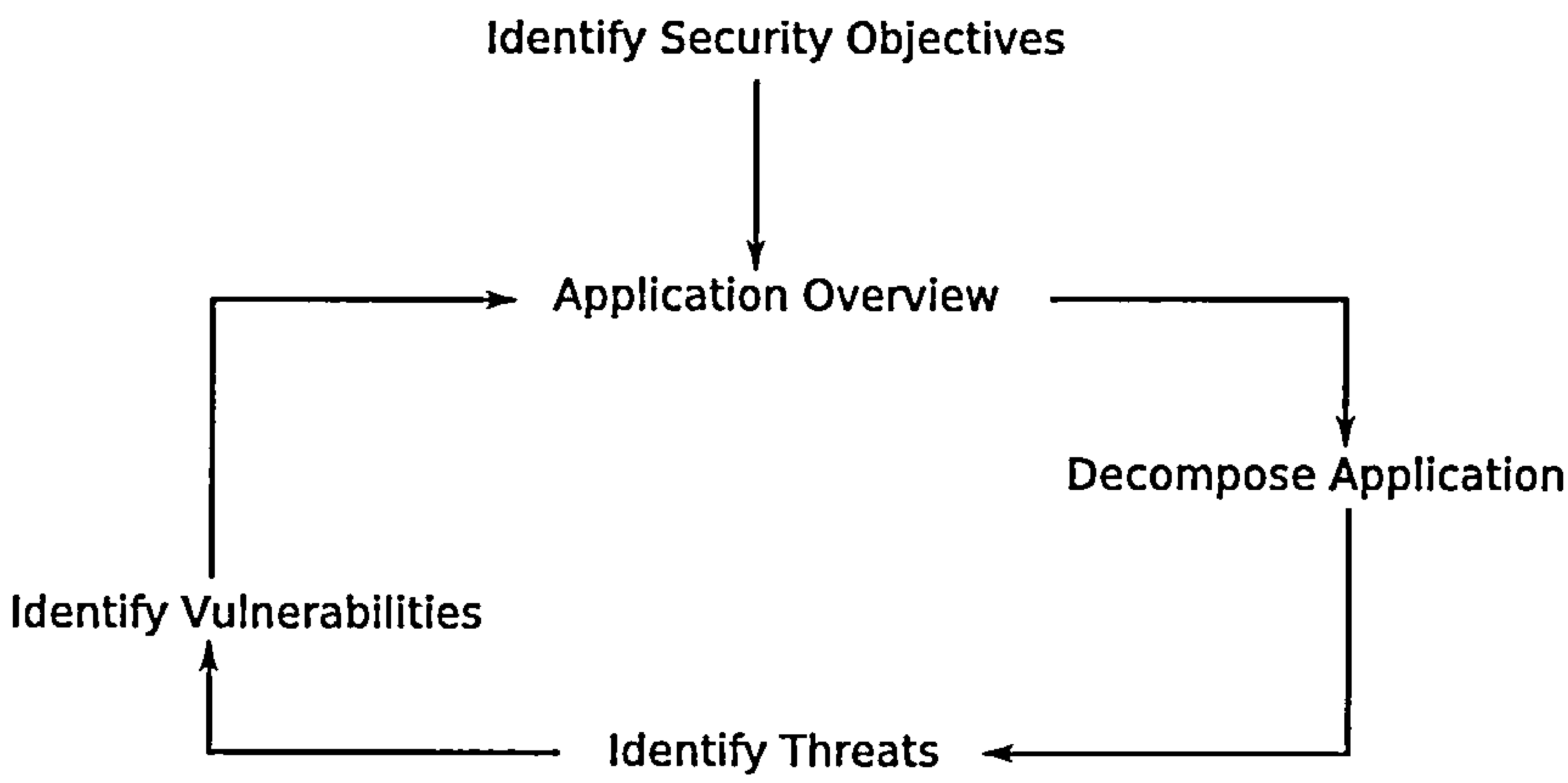



**Figure 3.5:** Iterative Risk Analysis Process [2]

The basic idea behind this process is quite straightforward: the developers have to understand the security objectives of the application firstly; then they study the application from a point of view of security, such as the security environment and architecture of the application; thirdly, they decompose the problem domain of the application in order to simplify the security problems; finally, they identify the vulnerabilities and thereby security risks of the application and analyse them; at the end they will review the application in order to ensure all vulnerabilities and risks are identified and analysed. The details of inputs and outputs of threat modelling is listed in Table 3.1.

Table 3.1 also demonstrates that security is embedded in the development of functionality: there is no definite line to distinguish the development activities of security and

| Input [a] | Output |
|---|---|
| **I Identify Security Objectives** | |
| • Business requirements<br>• Security policies<br>• Security goals | • Security objectives |
| **II Create an Application Overview** | |
| • Application infrastructure<br>• Use cases<br>• Functional specifications | • Brief diagram with end-to-end deployment scenario<br>• Key use scenarios<br>• System roles<br>• Security mechanisms |
| **III Decompose Application** | |
| • Deployment diagrams<br>• Use cases<br>• Functional specifications<br>• Interaction diagrams | • Trust boundaries<br>• Navigation routes<br>• Data flows |
| **IV Identify Threats** | |
| • Common threats | • Threats list |
| **V Identify Vulnerabilities** | |
| • Common vulnerabilities | • Vulnerabilities list |

**Table 3.1:** Risk Modelling Activities

[a] except the outputs of previous step

functionality; they are largely overlapped. Security development thus cannot be a standalone process parallel with or following functionality development, the two are interleaved. Analysis for security purposes generally require a system description in some form, including functional descriptions. For example, to identify a threat, the developers need the use cases, application architecture, and other design artifacts to conclusively identify threats.

### 3.3.3 Common Vulnerabilities of Web Applications

Figure 3.5 shows that identifying vulnerabilities is an important link in the process of risk analysis. A vulnerability is a weakness or defect of security protections. Therefore, how to treat the vulnerabilities becomes very important. Attackers do not create vulnerabilities, but exploit them. The best solution to deal with vulnerabilities is to reduce the vulnerabilities in the first place when the application is developed. This requires that the developers have been educated about possible vulnerabilities.

The vulnerabilities of an application are various because they are interdependent on the business logic and architecture of the application. Although there are two simple styles of the architecture of a Web application, MVC is one of popular choices. Web applications with MVC architecture have many similar security vulnerabilities. Therefore it is valuable to summarise a list of common vulnerabilities of Web applications. Such a list can help developers to specify the security requirements; to design and implement the application with security awareness; to plan security tests; and even to maintain the application.

Security of Web applications has been studied for a long time. There are many contributions from industry and academia about the threats, vulnerabilities and attacks of Web applications, including [2, 65, 57, 109, 117, 119]. Those documents are comprehensive, but very large: it is difficult to know where to start in terms of considering these concerns in an agile development. Based on an analysis of these documents, a compact list of common vulnerabilities is concentrated from those documents, and listed in Table 3.2.

| Category | Common Guidelines |
|---|---|
| Input/Data Validation | Do not trust input. Consider centralised input validation; not rely on client-side validation. Be careful with typical issues. Constrain, reject, and sanitise input. Validate for type, length, format and range. |
| Authentication | Use strong passwords. Support password expiration periods and account disablement. Do not store credentials. Encrypt communication channels to protect authentication tokens. |

Continued on next page

Continued from previous page

| | |
|---|---|
| Authorisation | Use least privileged accounts. Consider authorisation granularity. Enforce separation of privileges. Restrict user access to system-level resources. |
| Session Management | Authenticate users have a robust and cryptographically secure association with their session. Prevent common Web attacks, such as reply, request forging and man-in-the-middle attacks. Enforce authorisation checks. |
| Sensitive Data | Avoid storing secrets. Encrypt sensitive data over the network. Secure the communication channel. Provide strong access controls for sensitive data stores. |
| Exception Management | Use structured exception handling. Do not reveal sensitive application implementation details. Consider a centralised exception management framework. |
| Auditing | Do not record any sensitive data such as password. Identify malicious behaviour. Know what good traffic looks like. Audit through all of the application tiers. Secure access to log files. Back up and regularly analyse log files. |
| General Logic Error | Remove unwanted functional behaviour. Examine all possible data flows and block unnecessary routes. Use strong authorisation. Consider authorisation granularity. |

**Table 3.2: Common Vulnerabilities and Guidelines (derived from [2,65] )**

[2] is about protection by Microsoft products and [65] is more general for all types of Web system.

The purpose of such a list is to have materials to educate developers and to help the security development. Listing the vulnerabilities is not enough. The development process, especially the design process, is full of decision making. Developers need to identify what the problems are and how to handle them. In the thesis, the design problems are categorised into two classes: *architecture* (i.e. what protection is the fundamental or architectural?) and *mechanism* (i.e. what protection is the choice of implementation?). The problems in the *architecture* class should always have priority when designing. The analysis is shown in Table 3.3.

## 3.4 Development of Web Applications

Developing Web applications has its roots in many disciplines, such as the development of information systems and hypermedia applications. From different perspectives,

| Problem | Problem Class |
|---|---|
| Input/Data Validation | It is always a *Mechanism* problem, so that security protections are implemented wherever there are user inputs. |
| Authentication | It is an *Architecture* problem. Authentication is always a separate function of the system. |
| Authorisation | It is always a *Mechanism* problem. The components of Control often provide some parts of authorisation function. |
| Session Management | It is a *Mechanism* problem. In a Web system, a session is always managed by the server. |
| Sensitive Data | It is a complicated problem. Sometimes, it needs a separate function to protect the sensitive data, for example using an extra Java bean to cover the sensitive data. But mostly, it is a *Mechanism* problem so that the protection is implemented in the existing components, such as encryption. |
| Exception Handling | It is always a *Mechanism* problem. The protection is implemented in the part of Controller and Model, especially the database exceptions. |
| Auditing | It is an *Architecture* problem. It needs extra components to implement this function. |
| Logic Errors | It is a *Mechanism* problem. And it always needs inspection and testing. |

**Table 3.3:** Analysis of Common Vulnerabilities

methods for developing Web applications have been proposed in the last ten years, for example MIDAS [91, 92] (a methodological framework for developing Web application), OOHDM [93, 94] (an Object-Oriented design method for hypermedia applications), OO-H [95, 96] (an Object-Oriented modelling technique for Web application), UWE [97, 98] (a UML-based, model driven approach), W2000 [99] (a complete notation for modelling complex Web applications), WebML [100] (a UML-based modelling language of Web application). Despite the different techniques and languages used when developing Web applications, some similarities can be observed:

- All methods identify the necessity of a brand new model — a navigation model — to tackle the navigation route of Web applications. Inside this model, all methods distinguish between information and access structures.
- All methods clearly separate content, navigation and presentation space by means of different models.
- All methods aim at defining a precise and systematic, even in some steps automated, process for the development of Web applications. The fundamental development process is based on the Waterfall model.
- In the end, all methods are based on UML or UML-like diagrams. [123, 124] summarise the UML syntax used in these methods.

The business logic always impacts on application security. Implementing business logic is the main part of developing an information system. When building a traditional client/server information system, the application is developed first, and the data are fed into the system separately. For Web system, the application and the data must be delivered together. In [123, 125], the characteristics of a project that is building Web applications are summarised:

- Focus on understanding requirement changes of customers;
- Short development life-cycle time, typically 3 month or less;
- Delivery of bespoke solutions integrating software and data;
- Multidisciplinary development teams;
- More rigorous requirements analysis, including a clear analysis of business needs;
- Better testing and evaluation of Web deliverables;
- More focus on the issues associated with the evolution of Web technologies.

Agile methods claim that they embrace changes, have short development iterations, and construct multiple roles in the development team. Web application development needs are well suited to agile approaches and indeed many Web applications are developed using them. So far, the best-known processes within the Agile Alliance are probably

- Adaptive Software Development [126]
- Crystal Methodologies [127]
- Extreme Programming (XP) [128]
- Feature Driven Development (FDD) [129]
- Scrum [130]

These Agile methods tend to focus on specific a kind of software development activities, and encourage small teams of highly skilled developers to start with the initial agile or lightweight process and expand it to suit their organisation and project. Agile methods are people oriented and encourage and embrace changes, allowing nearly the full project development time to define the problems and proposed solutions in their entirety.

### 3.4.1 Agile Attempts for Building Web Applications

So far, there is little research that attempts to link agile methods with the development of Web applications. The AWE process [125] is a lightweight approach aimed at tackling the problems associated with the development of Web applications. The AWE process [125] is not based on any established Agile methods. It comprises seven stages: *Business Analysis*, *Requirement Analysis*, *Design*, *Implement*, *Test*, *Evaluate*, and *Deployment*. The whole process looks similar to Waterfall model in a circle or spiral model.

But it considers some agile principles, such as good communication. Strictly speaking, AWE process [125] is a good attempt to apply the ideas of Agile Manifesto [25] in the development process of Web applications. But security has not been considered in either attempt. The reality is that Agile methods always consider how to satisfy functional requirements in an iterative and incremental way; security is not the priority in any Agile method.

## 3.5 Conclusions

Security of Web applications was reviewed in this chapter. Security considerations of Web applications can be summarised as a check-list, Table 3.2. Graphically, Figure 3.6 maps the considerations into a MVC architecture. For example, the components of **View** should *validate inputs* to prevent the vulnerability of `Input/Data Validation`; *protect sensitive data* to improve the vulnerability of `Sensitive Data`; and *maintain session's integrity* to improve the vulnerability of `Session Management`.

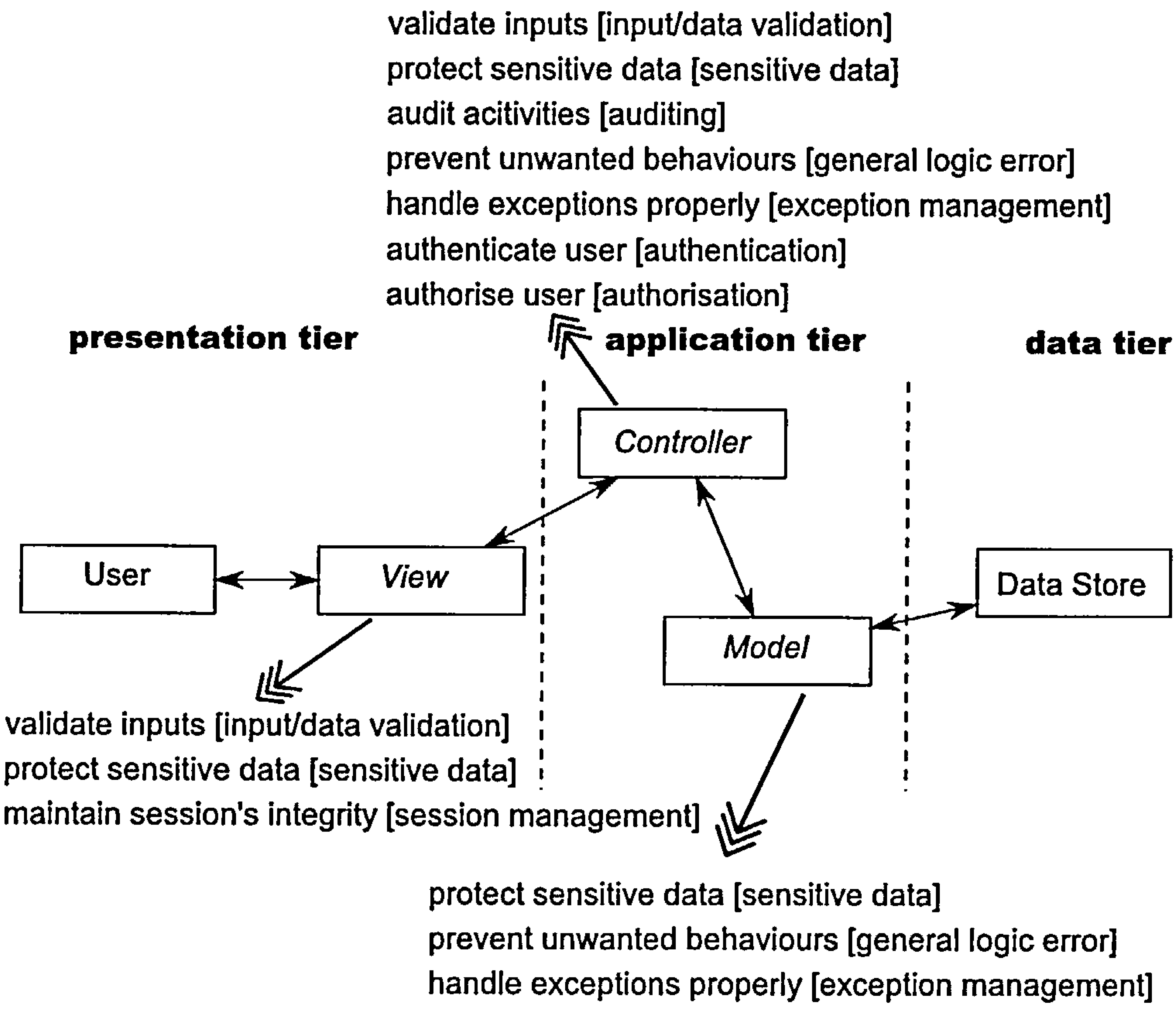



**Figure 3.6:** Web Application Security Check-list

The following can be concluded:

1. An Agile methodology is a suitable and justifiable choice to develop a Web application.
2. A Web system is best served by a layered architecture, shown in Figure 3.2; security of the Web application is the top layer.
3. The vulnerabilities of an application are interdependent on the application's business logic and architecture. Common vulnerabilities of a Web application are mapped into MVC architecture, shown in Table 3.3 and Figure 3.6.

Furthermore, implementing business logic is the main theme not only for developing a Web application, but also for building an information system. The next chapter will talk about Agile methods.

# 4 Agile Software Development

*The previous chapter argued that Agile methods are an appropriate and justifiable choice to develop Web applications. This chapter reviews several mature Agile methods, and focuses on selecting a suitable agile method with which to integrate security. The selected method is reviewed briefly in this chapter.*

## 4.1 Agile Software Development

The modern definition of agile software development evolved in the mid 1990s as part of a reaction against *"heavyweight"* engineering methods, as typified by a heavily regulated, regimented, micro-managed use of the waterfall model of development. The processes originating from this use of the waterfall model were seen as bureaucratic, slow, demeaning, and inconsistent with the ways that software engineers actually perform effective work.

In 2001, seventeen prominent figures created the Agile Manifesto [25], widely regarded as the canonical definition of agile development, and accompanying agile principles.

Some of the principles behind the Agile Manifesto [25] are:

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Even late changes in requirements are welcomed
- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organising teams
- Regular adaptation to changing circumstances

The publication of the manifesto spawned a movement in the software industry known as agile software development.

For many people the appeal of agile software development is a reaction to the bureaucracy of the traditional software development methods. These new approaches attempt a compromise between no process and too much process, attempting to provide just enough process to gain a reasonable payoff.

Compared with plan-driven software development, the values of Agile methods are [25]:

- **Individuals and interactions**
- **Working software**
- **Customer collaboration**
- **Responding to change**

The result of all of this is that agile software development have some significant changes in emphasis from traditional methods. The most immediate difference is that they are less document-oriented, usually emphasizing a smaller amount of documentation for a given task. In many ways they are code-oriented: following a route that says that the key part of documentation is source code.

In short, Fowler summarised in [131] the following views:

- Agile methods are adaptive rather than predictive. Engineering methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.
- Agile methods are people-oriented rather than process-oriented. The goal of engineering methods is to define a process that will work well whoever happens to be using it. Agile methods assert that no process will ever make up the skill of the development team, so the role of a process is to support the development team in their work.

## 4.2 Agile Methods

Agile methods are a family of development processes, not a single approach to software development; they include eXtreme Programming (XP) [128, 132, 133], Feature Driven Development (FDD) [129, 3], Adaptive Software Development [126], Crystal Methodologies [127], Scrum [130], and others. Agile methods are surveyed in [131, 11, 134]. Table 4.1 summarises the features of these Agile methods.

| Method | Key Points | Special Features | Identified Shortcomings |
|---|---|---|---|
| ASD | Adaptive culture, collaboration, mission-driven component based iterative development | Organisations are seen as adaptive systems. Creating an emergent order out of a web of interconnected individuals. | more about concepts and culture than software practice. |
| Crystal | Family of methods. Each has the same underlying core values and principles. Techniques, roles, tools and standards vary. | Method design principles. Ability to select the most suitable method based on project size and criticality. | Too early to estimate: only two of suggested methods exists. |
| XP | Customer on-site, test driven development, small teams, daily builds. | Refactoring - the ongoing redesign of the system to improve its performance and responsiveness to change. | While individual practices are suitable for many situations, overall view and management practices are given less attention. |
| FDD | Five-step process, Object-oriented component (i.e. feature) based development. Very short iterations: from hours to 2 weeks. | Method simplicity, design and implementation the system by features, object modelling. | FDD focuses only on design and implementation. Needs other supporting approaches. |
| Scrum | Independent, small, self-organising development teams, 30-day release cycles. | Enforce a paradigm shift from the "defined and repeatable" to the "new product development view of Scrum." | While Scrum details in specific how to manage the 30-day release cycle, the integration and acceptance tests are not detailed. |

**Table 4.1:** General Features of Agile Methods [11]

## 4.3 Selection of Agile Methods

As discussed in previous chapters, security is a key concern for Web systems and Web applications are ideal for development by an Agile process. It will be useful for building Web applications if there is an integration of security and agile development proposed.

The Agile method for the security integration will be selected from the most mature Agile methods, discussed in previous sections. In Chapter 2, the criteria of selecting Agile method is proposed:

- the method should have a clear (explicit) development process and it is better that the process have some simple milestones so that security review loops can be easily integrated (see Figure 2.9).

- An architecture is essential for not only the documentation requirement of security evaluation but also the requirement of risk assessment. Therefore, it is better that a general blueprint of system can be produced at the early stage of SDLC so that the architecture can be drawn.

Among the existing Agile methods, Feature Driven Development (FDD) seems to be the best choice because:

- Compared with other Agile methods, such as XP, FDD has a simple process. It will be easier to compare and integrate with security development process.
- FDD is a model-driven process. The feature is a fundamental element of FDD. FDD is a model-driven process because the features are always modelled during the development. Risk assessment is also a model-driven process so there is a high probability that risk assessment can be integrated into FDD smoothly if their models can be merged. Other Agile methods, such as XP, are not model-driven methods.
- FDD emphasises overall modelling at the beginning of development which suits the requirements of security development. Other Agile methods do not have such a stage to design an architecture of the system.

In next section, FDD will be briefly reviewed.

## 4.4 Feature Driven Development

Feature Driven Development (FDD) was first introduced in 1999, as a tailored complement to the *"Object Modelling in Colour"* technique [135]. A revised version of the methodology was published in 2002 [3]. This version was completely decoupled from *"Modelling in Colour"*, and was general enough to be considered an independent methodology.

Like other agile processes such as XP, FDD focuses on short iterations that deliver tangible units of functionality. But FDD is different in that it is model-driven; models (e.g., written in standardised languages such as UML) are created to help identify units of functionality, understand the context in which units are used, and to help derive code.

When contrasted with other agile processes, particularly XP, FDD has a straightforward process, which contains five sub-processes (shown in Figure 4.1). The first three sub-processes are concerned with requirements analysis and development planning. They are performed sequentially at the beginning of the project. The remaining two are design

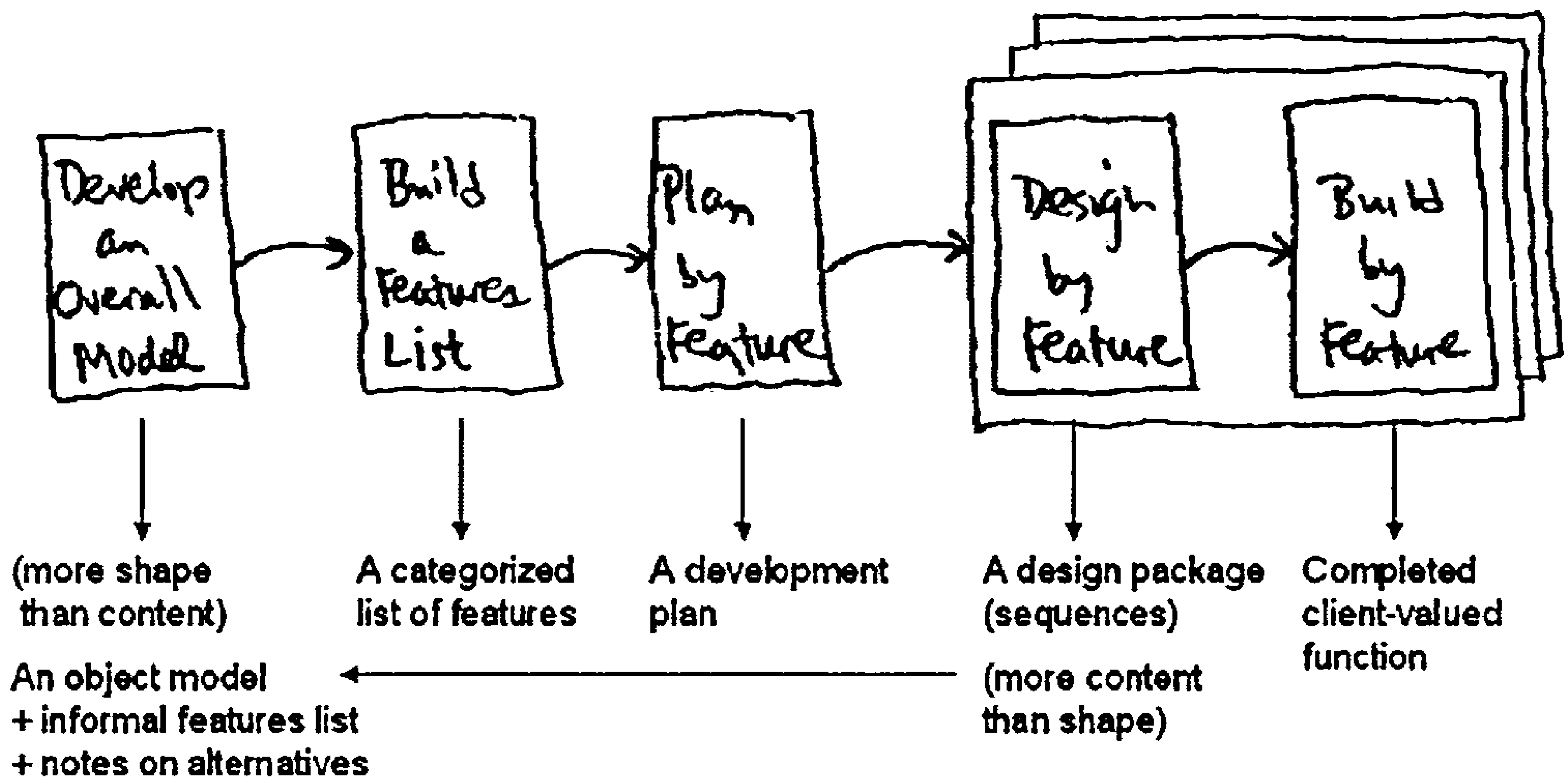



**Figure 4.1:** FDD process and its deliverable [3]

and implementation activities, done in a series of short iterations. Briefly, these five sub-processes are (details in [3]):

- Develop an Overall Model (DOM), an initial project-wide activity with domain and development members under the guidance of an experienced modeller in the role of Chief Architect. During this phase, the problem domain is modelled; this is called the *Object Model.* This model mainly consists of full featured class diagrams and interaction diagrams (if needed) for capturing important behavioural patterns of interactions in problem domain.

- Build a Features List (BFL), an initial project-wide activity to identify all the features to support the requirements. It is done by first identifying the *areas* of functionality in the system, and the *activities* performed in each *area.* Features are then identified as *steps* in the activities.

- Plan by Feature (PBF), an initial project-wide activity to produce the development plan. It focuses on scheduling the features for development, assigning the feature sets (*activities*).

- Design by Feature (DBF), a per-feature activity to produce the feature design package. It focuses on determining how the features in the work package should be realised at run-time by interactions among objects. In general, interaction diagrams are drawn for each of the features, resulting in additions and modifications being made to the object model, and refined class and method descriptions being produced.

- Build by Feature (BBF), a per-feature activity to produce a completed client-value function (feature). It focuses on programming and unit testing the necessary items of the features in the work package. The implemented items that pass the tests are then promoted to the main build.

There are lots of arguments about the agility of FDD process. The Agile Manifesto [25] says that the values of an agile software development are:

- Individuals and interactions
- Working software
- Customer collaboration
- Responding to change

The most focused argument against FDD comes from the second and fourth bullet. The second points out that the value of an agile software development is working software, rather than a set of comprehensive documentation; the fourth bullet says that responding to change is more valuable than strictly following a plan. FDD considers **good** documentation and planning a required part of any project. However, agility requires maturity to understand the balance between these values in the manifesto and what it really means to a real project. In practice, it is very difficult to draw a dividing line between *"too much"* and *"too little"* of documentation and planning. That is the key reason that during agile software development, applying agile practices is still an art.

Compared with XP, there is a view that FDD is less agile than XP because it employs many of the traditional practices that XP practitioners are probably not accustomed to see, including conducting up-front planning, design and documentation and relying very heavily upon domain modelling. There are several comprehensive comparisons between FDD and XP, such as [136, 137]. As to what constitutes as an Agile method, the widely accepted view is that the Manifesto for Agile Software Development [25] is the only criterion. Software engineers may have their own opinions based on their understanding of the Agile Manifesto. In the author's eyes, FDD is agile because it fully presents the values of agile software development.

In FDD, the most significant characteristic is the *feature* which is the basic unit of work. A feature is a small, client-valued function expressed in the form:

$< action >< result >$ $[of|to|for|from]$ $< object >$ [3].

This expression can be restated as: an action causes a result to an object. The section in the brackets is optional to make the feature easier to read. An example of a feature is: *Search* the *matched properties* for a *customer*. Here *search* is the $< action >$, *matched properties* is the $< result >$, and *customer* is the object participating in this feature.

In a project, Features can be combined into Feature Sets, and Feature Sets can be aggregated into Subject Areas. Requirements are usually gathered in a top down approach, defining all the Subject Areas for the system, breaking these down into Feature Sets, and eventually down into Features, from which tasks can actually be defined and estimated. Feature Sets typically reflect business activities, and Subject Areas commonly correspond to general business practices.

Feature capturing and planning is the main emphasis of FDD. Again, it is essential to understand that everything in the development is absolutely planned, built, managed, and tracked on a per-feature basis. Other units such as feature sets and subject areas are available for higher level planning and reporting, but the key universal unit is the feature.

FDD also uses several unique and specific mechanisms to report project activity and progress [138]. The least unique of these is the feature list and the task list. Many Agile methodologies use some sort of list to track requirements and work done on requirements. In FDD, these lists all correspond to a specific feature.

Capturing exactly where a feature is in its development cycle is done using a table that tracks six specific milestones:

1. Domain Walk-though
2. Design
3. Design Inspection
4. Code
5. Code Inspection
6. Promote to Build

These milestones are tracked by the date each is completed and by the Chief Programmer responsible for that specific feature.

Feature milestones are summarised in a table showing feature sets. This allows project stakeholders and managers to see how the project is progressing. Things can be summarised into subject areas as well if there is value in doing so. Completed features are also tracked across the entire project using a line graph. This graph shows the cumulative total by day or week of all features that have been completed.

The final group of reports is the Feature Set Progress Report, which is very unique to FDD. This report is colour coded and shows each Feature Set in the project, what percent complete it is, how many features are in each area, the name of the Chief Programmer for each Feature Set, and the month and year when each is targeted to be completed.

## 4.5 FDD and Architecture

Many Agile method practitioners, like Kent Beck (founder of eXtreme Programming (XP)), put energy into avoiding (but not excluding) any up-front architectural design [131]. FDD encourages developers to consider the architecture of system in advance. In fact, FDD suggests partitioning an application into layers that include (some of) a

User Interface layer (UI), a Problem Domain (PD) ("business logic" layer), a System Interface (SI) layer and a Data Management (DM) layer. The models from each of the layers build up the overall model of the system which is the basis of *feature analysis*.

The UI layer is all about presentation. As it is typical in the Model-View-Control architecture of Web applications, UI layer is decoupled, and most importantly there is a one-way dependency from the UI layer to the PD layer and not the other way around.

The SI layer deals with interfaces to external systems. FDD uses a similar approach to the UI layer to decouple these, which achieves a dependency from the SI back onto the PD and not the other way around.

The DM layer deals with persistence. In many modern Web applications, the DM layer is sliding away from the centre of the development: the developers care only about the structure of their data store, and leave other issues, such as connection mechanism, performance tuning, and backup/restore, to mature commercial products and tools. It is more often a "buy-not-build" solution.

The real core of the development is always the PD layer, where the domain — the functionality of system — is modelled.

This four-layer architecture is proposed for general information system. Considering the Web application and its MVC architecture, the four-layer architecture is still reasonable. The View layer is equivalent to UI; and the Model and Controller is made up of PD.

## 4.6 Agility, FDD and Security

Several researchers have contrasted general Agile methodology [27, 28, 29, 30] or XP development [139, 140] with traditional security engineering processes, and while these reviews are generally favourable to XP, they also highlight the gap between the documentation requirements of traditional engineering and the lightweight approach of agile methods.

Beznosov presents a conceptual analysis of the suitability of Extreme Programming for building secure systems, and also introduced the notion of "good enough security" without giving any description of how good the security will be [140]. The objective of the Planning Game, one of the XP Practices, is defined so as to plan small releases in short iterations while delivering 'good enough' security through tested functionality units. [140] also gives extended definitions of other practices such as testing, continuous integration, simple design and refactoring which are adapted to so-called 'Extreme Security Engineering'.

Chivers et al [27] proposes that agile security can be achieved by using an Incremental

Security Architecture (ISA). Moreover, [27] states that "instead of following traditional techniques, [an agile process] must have its own, agile, security practices". Aydal [139] presented an interesting attempt to use Refactoring and a modified version of the Planning Game to iteratively and incrementally retrofit security mechanisms to a complete system, within the context of an Extreme Programming development.

None of the work discussed above has focused strictly on the domain of web applications and FDD, though [139] explores refactoring patterns for producing Web services.

## 4.7 Conclusion

There are several established Agile methods; how and when to apply these methods depends on the nature of a real project. Among them, FDD is an agile software development because it supports the values from Agile Manifesto [25] which is believed to be the only criterion to evaluate a development method in terms of its agility.

FDD has the more traditional progression of a systems engineering life-cycle model (i.e, it uses distinct phases in its iterations while still being highly iterative and collaborative.). FDD does emphasise up-front planning, design and documentation and relies very heavily upon domain modelling.

Up-front planning and architectural description may not be common features in an agile process, but they are necessary in security development. That is the major reason why FDD is selected because the security development needs software architecture.

So far, this part of the thesis demonstrates that FDD and risk assessment are suitable choices to be integrated together. The next chapter will explain the practical problems that arise when integrating these two methods.

# Part II

# Integration of Agile and Security Development

# 5 Security in Agile Development

*The previous chapters have reviewed background knowledge of security engineering, Web applications, and agile software development. In Chapter 4, FDD was selected as a suitable method for the study of integrating security and agile development. This chapter will discuss the challenges and solution of the integration.*

## 5.1 Difference of Agile and Security Development

Integration means combining two different things into an integral whole. The previous chapters, especially Chapter 2 and Chapter 4, reviewed definitions of security development and agile development. To integrate agile and security development, the first thing is to understand how and why they are different.

First and foremost, their philosophies are different. Security development aims to build a secure system. In the development, a target for evaluation is always set before the project starts; the system and its development process are evaluated when the project ends. In other words, the developers know what to do to achieve security before they think about how to do it, so they plan carefully. Agile development aims to make the development process more manageable, especially when changes to requirements occur. In an agile project, the developers use a few practices in order to manage change, but actually they cannot predict what the change will be, so they cannot plan it easily, if at all.

A technical indication of this difference is in their respective development processes. Security development is based on a plan-driven process. In the development process, security evaluation criteria and risk assessment play an important role. Figure 5.1 shows the process again.

By contrast, plain FDD [3][1] has an iterative development process. In order to compare with security development process, Figure 5.2 shows the process of plain FDD, drawn in the same style as Figure 5.1.

---

[1] In order to distinguish the security integration of FDD, the original FDD [3] is labeled as plain FDD in this chapter.

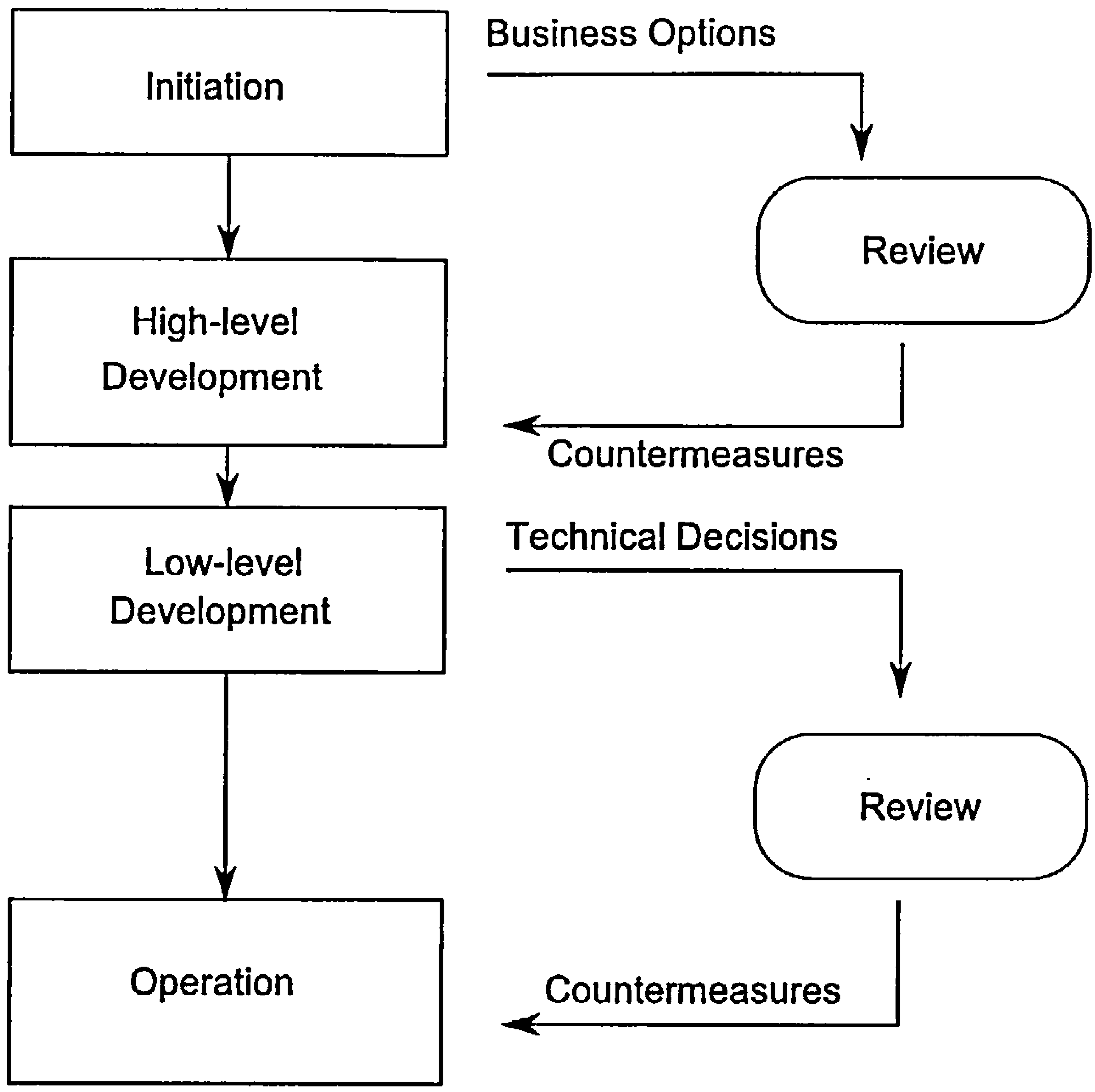



**Figure 5.1:** Framework for the Security Development Process

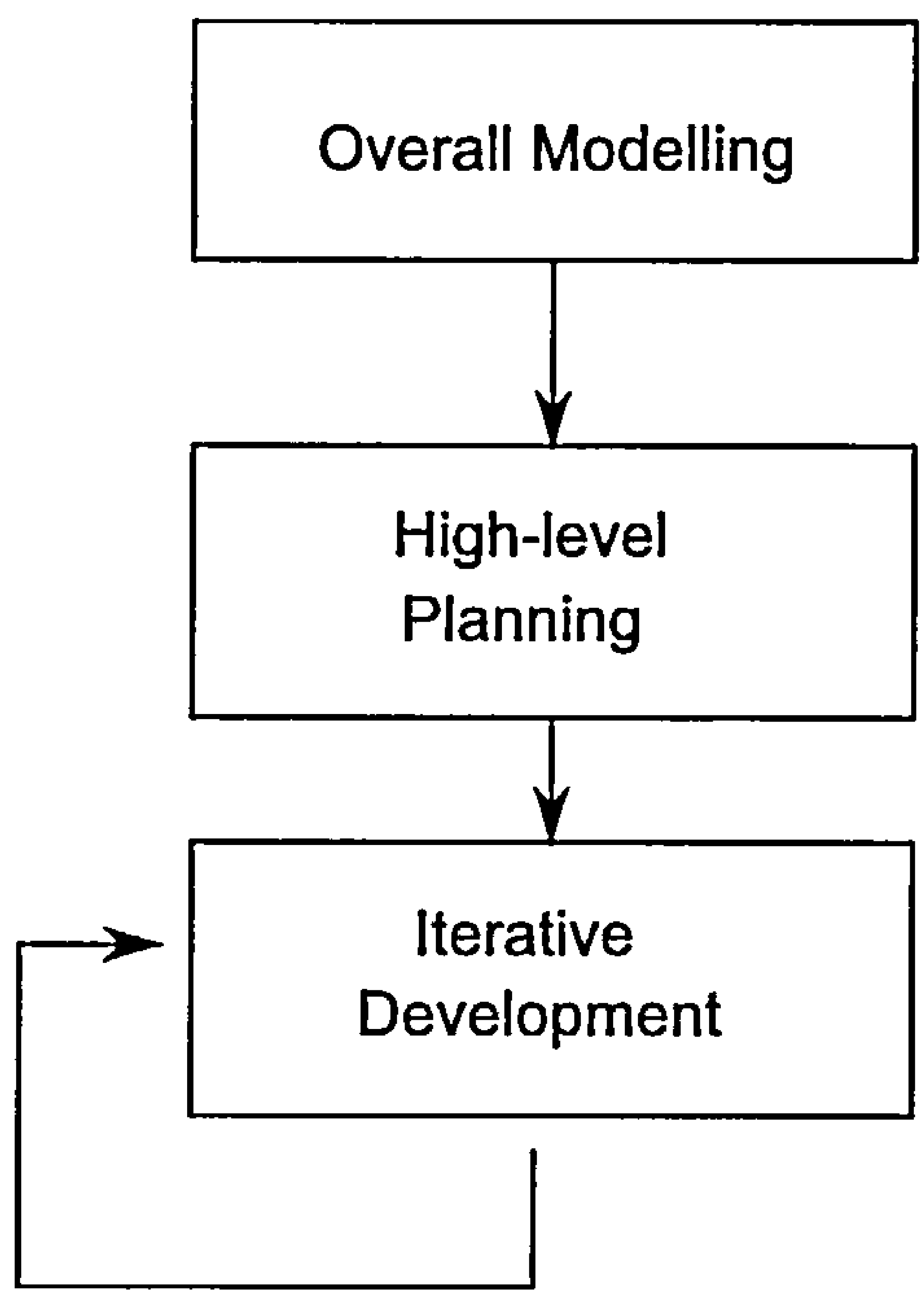



**Figure 5.2:** Framework for the Plain FDD Process

From these figures, similarities can be found. Both processes have two stages: high-level development and low-level development; during the high-level development, the process is generally sequential. Although they are similar, the two processes are different in content: different materials and activities of development are emphasised. Most importantly, the differences are in *architecture* and *documentation*.

Changes, especially late in development, typically lead to substantial costs if a traditional software development methodology (i.e, plan-driven methodology) is adopted in the project; the effort put into developing security artefacts within these methodologies can be lost during a change process. To manage the problems caused by changes, a plan-driven methodology emphasises the thoroughness of up-front design and the importance of the architecture, especially for a critical system. The nature of software systems leads many developers and analysts to assume that a code-level description of software is sufficient for spotting design problems[2]. Although this might occasionally be true, it does not generally hold. Security development requires an essential understanding of system architecture. This attitude to the architecture is the first obvious difference between agile development and security development.

In general, some Agile method practitioners emphasise avoiding any up-front architectural design. But *"(architectural) design is not dead"* [141]. The role of architecture depends on the project at hand. For example, if the business logic of a software system is very complex, it may be better to have an overall domain model — which is an architecture — to capture early on how to layer the application. Whether the architecture is needed at the beginning of development is not a big problem of plain FDD because plain FDD has already outlined the importance of an overall model, which provides an overall framework in which to add features. The overall model more or less serves the same purpose as architecture does.

In security development, it is necessary to establish an initial architecture to capture early on how to satisfy security requirements and place security mechanisms, as well as to provide a basis for security risk assessment. A central activity of risk assessment at an architectural level is to build up a consistent view of the target system at a **reasonably** high level. The most appropriate level for this description is typically an overall object view of boxes and arrows describing the interaction of various critical design components. Without such a *"white-board"* level of description, the risk assessment is likely to miss important risks related to flaws. For some small system projects, a security architecture can be produced in an incremental way [27].

Another obvious difference between agile and security development is the importance placed on documentation. The gap between agile and security development is well understood [140, 142]. For critical systems, security is normally evaluated by inspections which is a document-based process against security evaluation criteria; three of them were literately reviewed in Chapter 2. The requirements of documentation are clearly defined in security evaluation criteria. A set of documents is the evidence used for se-

[2]XP's claim that *"the code is the design"* represents one radical extreme to this approach [133].

curity evaluation; they are the formal means of communication between developers and evaluators.

Communication is explicitly highlighted in almost every agile development process. But communication in agile development is people-oriented, instead of document-oriented: agile development encourages more communications in many informal ways, such as talking in person. Agile development always emphasises a smaller amount of documentation for a given task.

There has to be a compromise about documentation in a real project. Security evaluation criteria are rigorous: without certain documents, the system cannot be evaluated. It seems that there is nothing to do to reduce the amount of documentation required by security development. However, some documents, such as design documents, are interim products of the development process. Applying selected agile practices may improve their quality; for example, design inspection from plain FDD can verify the design documents. The agile documentation in the developing process of critical systems is a different story; it is beyond the scope of this thesis. [143] summarises important characteristics of agile process and the documents produced, but it does not discuss the case of critical systems.

## 5.2 An Integration of an Agile Process and Security

Agile development (and in particular plain FDD) is different from security development. Considering the integration of these two types of development, there are two possible approaches: including applicable security artefacts in the process of plain FDD or including applicable agile practices in the process of security development. These two approaches have different starting points: the first one is based on extending the process of plain FDD; and the other is based on extending the process of security development. This research takes the first approach because agile approaches have already been adopted in non-critical software projects, especially Web applications.

The aim in integrating security concerns with plain FDD is to propose a solution which can *balance* agility and security assurance for Web applications. Plain FDD is an Agile method, but it does not consider the security development. In my view, a better solution is to consider security (i.e. designed, and implemented) systematically during the development, rather than in an ad-hoc way. Also by extending plain FDD, the process is also classified as an agile process, because it keeps the common features of agile software development.

Consequently, the integrated process proposed in this thesis is evaluated on the following criteria:

- Quality of the delivered application. The quality, here in the research, focuses on security. Formally, the way to evaluate the security of a software system is by security evaluation criteria. Critical systems are not the natural "home ground" of agile software development [7]. Hence, the delivery of an agile development is not proposed to achieve the highest level of security evaluation criteria. But it does not mean that the deliverable of an Agile method cannot be *acceptably* secure. There are some industry security guides, like [65], which is a handbook for a particular type of information system. The developer can reference guidance when they specify security requirements. Once the system is built, the system can be validated against these requirements. Whether the system is evaluated against established security criteria or against bespoke security requirements, the target degree of security is decided by the customer. In the research, the security requirements of the system are established by using the check-list in Chapter 3 which is based on [65].

- Agility of the development process. In Oxford English Dictionary (OED), agility means "the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion." In practice, Agility requires that the software development should be [11, 131, 133, 144, 145]:
  1. incremental (small software releases, with rapid cycles),
  2. cooperative (customers and developers working constantly together with close communication),
  3. straightforward (the method itself is easy to learn and to modify, well documented),
  4. adaptive (able to make last minute changes)

  The proposed solution will be discussed against these concrete requirements.

First of all, the problems of actually carrying out the integration are identified and discussed in the following sections.

### 5.2.1 Overall Process

Chapter 4 argued why FDD is a suitable Agile method for integration with security development. Conceptually, for the integration, several security concepts should be added to plain FDD. Figure 5.3 gives an overview of the concepts in *secure FDD* (sFDD). The red labelled[3] components are additions to plain FDD. Now, the additions will be discussed and justified.

First of all, plain FDD does not explicitly address the issues of collecting and managing requirements. It assumes that documented requirements exist. For sFDD, it is reasonable to assume that the security requirements based on particular security evaluation criteria have been agreed between the customers and developers. The security requirements always include the required security functions and assurances.

[3] The original figure is coloured. In black-and-white, red elements appear in boldface.

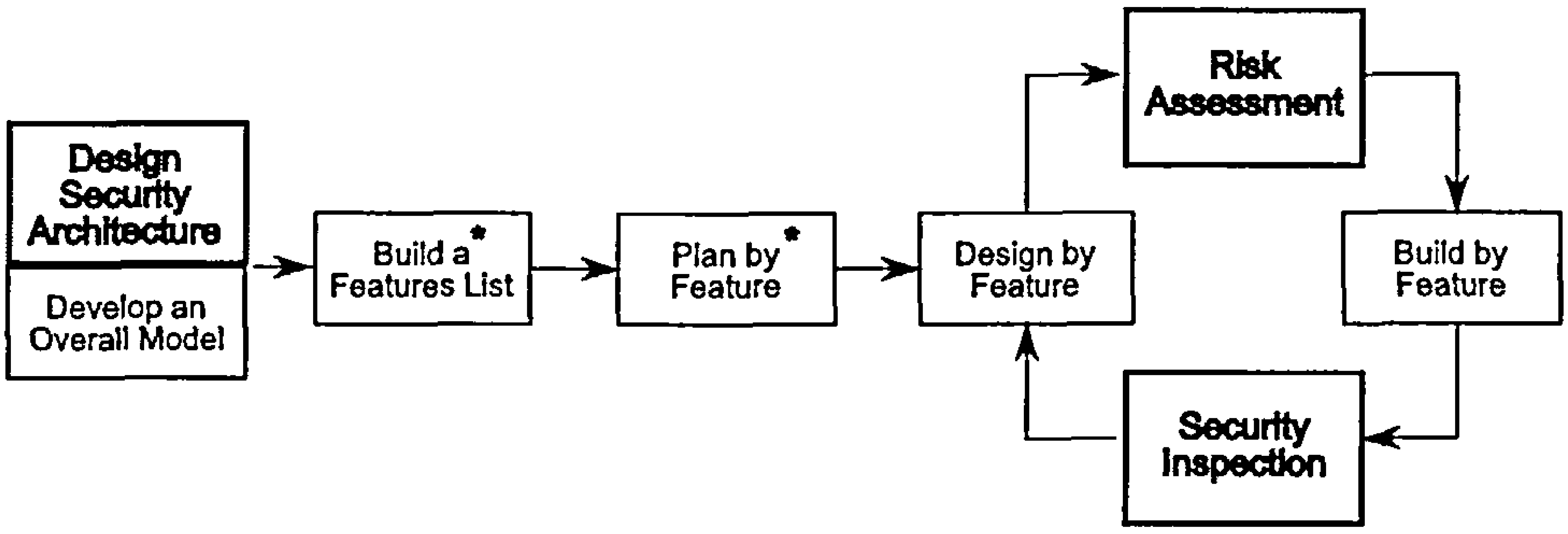



**Figure 5.3:** Process of Secure FDD (sFDD)

The key element introduced in sFDD is a *security architecture.* The overall model (which is part of plain FDD) structures the functional features; then the security architecture locates the security function features.

As in plain FDD, sFDD also has a plan in which the features are prioritised and the development iterations are managed. Within each development iteration, the features are implemented, and the security is ensured. The security assurances are gained by testing or inspection.

### 5.2.2 Overall Models

The models in plain FDD are always used to refer to business logic. Plain FDD [3] suggests that the results of *Develop an Overall Model* are a set of object models, including

- Class diagram focusing on the structure of the application.
- Methods and attributes identified are placed in the classes, to partially capture behaviours.
- Sequence diagram(s), if any, to refine behavioural specifications.
- Notes on models.

For a Web application, the overall model should cover the aspects of all three layers: View, Controller, and Model. Table 5.1 lists the logic each layer should implement and the models they produce.

When developing a Web application, the design of its graphic interface (the presentation) always takes a lot of time and effort, but it is not the focus of this research. Instead the author focused on the design of access control and business logic because of their impact on security.

| Architecture | Implemented Logic | Model |
|---|---|---|
| View | presentation logic | GUI design |
| Controller | access control logic | navigation & functional design |
| Model | business logic | functional design |

**Table 5.1:** Overall Model of Web Application

### 5.2.3 Initial security requirements

The stage of collection and analysis of security requirements is not covered in sFDD, but it is important to security development. Requirements engineering is an important and mature domain in software engineering [146, 147, 148]. A security requirement is a manifestation of a high-level organisational policy into the detailed requirements for a specific system [149]. Although some security concerns are addressed during the requirements engineering stage, many security requirements come to light only *after* functional requirements have been determined. The literature on general security requirement engineering has many examples of how to specify the security requirements, e.g, [66, 78, 150, 151, 152, 153, 154, 155, 156].

International security evaluation criteria can help the specification of security requirements. In Chapter 2 three important criteria are reviewed. The security requirements of a information system may consist of two parts: security functions and security assurance. In the security criteria, these requirements are explicitly clarified.

On the other hand, certain industry handbooks, such as [65], can provide guidance to document the initial security requirements. Based on these handbooks, Chapter 3 summarised the common vulnerabilities of Web applications in Table 3.2. This list of common vulnerabilities (or the goal of the protection against these vulnerabilities) provides thoughts of what security functions are needed in a Web applications, especially Table 3.3 map these vulnerabilities into MVC architecture.

In a real project, the security requirements are decided by the customer, sometimes with the help of security evaluation criteria or an industry handbook. Security assurance in security evaluation criteria includes operational assurance and developmental assurance[4]. Neither plain FDD nor secure FDD covers the operation phase of SDLC; only developmental assurance is addressed in sFDD. The basic developmental assurances are *security testing* and *design specification and verification* according to security evaluation criteria, for example TCSEC [8][5]. Risk assessment which is discussed in Chapter 2 cannot totally replace security testing, but at least it can give more confidence about security, especially of the design. Plain FDD introduces two inspections in development iterations to improve the quality of design and code. They are *design inspection* and

[4]life-cycle assurance of TCSEC [8].
[5]see Table 2.2

*code inspection*. Risk assessment and security inspection are also used to examine the flaws and errors in the design and implementation. To highlight the security artifacts, these two security activities are identified as two separate steps in sFDD (see Figure 5.3).

### 5.2.4 Security Architecture

A security architecture is a plan and set of principles that describe the security functions that a system is required to provide to meet the needs of its users [64]. The security requirements provide the list of security functions required by the system. Considering how to structure these security functions, the overall model (or models) is the major resource of consideration. The most important is to build up a **consistent** view of the target system, from both security and business perspective. Besides this, the following elements have also to be considered in the architecture:

- threats present in each layer.
- kinds of vulnerabilities that might exist in each layer.
- business impact of such technical risks.
- probability of such a risk being realised; and
- any feasible countermeasures that could be implemented at each layer.

The first two bullets are about threats and vulnerabilities, and threats and vulnerabilities of Web applications were discussed in Chapter 3, Table 3.3. The other bullets are about risks and their recommendations, which are the product of risk assessment.

## 5.3 Conclusion

This chapter gave an analysis of the problems encountered when integrating security with plain FDD. In short, the architecture and documentation are two major issues of security integration. This chapter explains that the architecture and documentation are necessary for the security development, but how much architecture and documentation depends on the project: too much architecture and documentation will make the development a plan-driven process.

The overall model (which is an unique feature of FDD compared with other Agile methods) provides a foundation for security development and development planning.

Security requirement is an extra factor which is required by the security integration because plain FDD does not cover the stage of requirement elicitation and specification.

Both international security evaluation criteria and industry handbooks can help specifying the security of a software system.

The overview of sFDD presented in this chapter is insufficient to guide software engineers. A concrete (step-by-step) description of the process is presented in the next chapter. It will explain the details of activities in the sFDD method: then, at the end of this part, a case study is used to demonstrate how the process is applied.

# 6 Secure FDD in Detail

*The previous chapter described the problems associated with security integration. A general overview of secure FDD was presented, which included separate steps for security and agile development activities. This chapter will give the technical details of the secure FDD process.*

## 6.1 Overall Process

A general SDLC for information systems consists of the following phases [157, 18]: **Initiation, Development, Operation and Maintenance**, and **Disposition**. Plain FDD focuses on the phases of how the system is built, i.e, **Development**; it does not consider the activities in other phases of a SDLC. The entire process of plain FDD consists of two large stages: development at a high level (i.e, architecture level) and development at a low level (i.e, feature level). At the beginning of plain FDD, the first stage of development is a linear process, of which a list of features and a plan is the product; the second stage of development is an iterative process, which is based on the plan and builds the system feature by feature.

Inheriting this structure from plain FDD, secure FDD also consists of two stages, but introduces security activities. In order to emphasise these security activities, they are highlighted in separate steps. In the first development stage, sFDD addresses the importance of security architecture; and in the second development stage, there are two steps of security assessment added. These two steps are different: working on different material and perhaps applying different detailed techniques.

Figure 6.1 shows the process of sFDD. The * mark indicates the new security activities in the step. The process of sFDD has two stages: linear development and iterative development. During each stage, there are also some steps. There are four steps, **Design an overall model, Design security architecture, Build a features list**, and **Plan by feature**, in the linear development stage; and there are four steps, **Design by feature, Risk assessment, Build by feature**, and **Security inspection**, in the iterative stage.

The stages and their steps in Figure 6.1 are briefly described in the following.

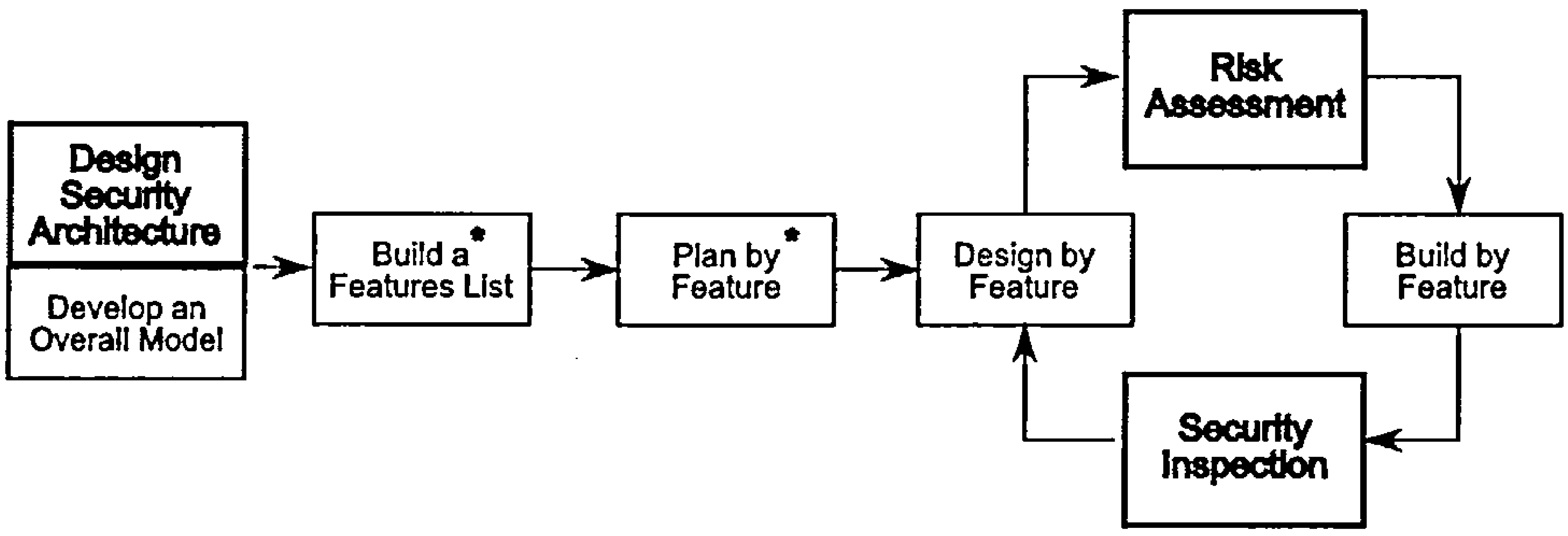



**Figure 6.1:** Secure FDD Process

### 6.1.1 Linear Development

The first development stage of sFDD is a linear process. It focuses on establishing the overall framework of the system, and a plan for progressing the development. This stage consists of the following steps.

#### Development an overall model

This is a step in plain FDD. Its objective is to develop an outline blueprint of the system, rather than a comprehensive analysis in depth. In this step, the developers begin to consider some security issues related to the security requirements — for example, the way in which security is evaluated, the development and operation environment.

In plain FDD, the outcomes of this step are the object models which identify the relationships among the pieces of the system's functionality. In addition, developers of security have to make some decisions about the security development at the end of this step. Some of these decisions will have to be documented. The nature and form of documentation depends on the nature of the project.

#### Design security architecture

This is a new step in sFDD. In the previous step, a high-level walk-through of the scope of the system and its context was performed. A Web application is only one link in a chain of Web system (see the discussion in Chapter 3). To protect the application, security mechanisms should be considered in advance. For instance, a Web system is always deployed in a three-layer architecture — *presentation*, *application* and *data store* layer. But to protect the application, the most commonly used security mechanisms are generally provided by other components in the system. For example, a Web server can provide the typical mechanism of authentication and access control; a commercial DBMS can often provide some security mechanisms, such as access control.

This step aims to identify the essential security mechanisms to protect the application; determine which of them are provided by related components in the system; then document security mechanisms around the business functions of the application in a model of the security architecture.

#### Build a features list

This is a step in plain FDD. The developers are organised into a team to functionally decompose the problem domain (i.e, business logic) into features. A feature is a piece of a customer-valued functionality. Features are granular in accordance with the rule that a feature will take no more than two weeks to complete. The estimation of feature development relies on the experience of developers, especially when security is considered. Therefore, security professionals are one of the major participants of the feature list team.

#### Plan by feature

Once the features are identified and categorised, the managers of the project (including the project manager, development manager, and other chief members in the team) begin to plan the order in which the features are to be implemented. The priority of features is based on dependencies, and the complexity of the features.

The security mechanisms attached to the features are also an important factor which may affect the order of feature development. For example, the strength and coverage of security mechanisms are always considered when planning the development of features. A typical example is that a Web application always consists of two parts: some pages are public to every users; but some of them are only for members. The public pages should be developed first. The "private" functions of the application would be built only after the mechanism of access control is set up.

#### Summary

The linear development stage consists of four steps. At this stage, sFDD does appear similar to security development based on a plan-driven approach: it employs up-front analysis at the first stage. But this up-front analysis stops at the feature level. The objective of up-front analysis is to identify and prioritize the features, rather than every detail of functions to be implemented in the application. This is the key difference of sFDD compared with traditional security development.

By introducing these security development activities into plain FDD, the developers' focus can always be drawn to security issues alongside the process of functionality development. In addition, the security architecture can also be reviewed during the iterative development — if a feature in the features list is changed later, it is easy to go back to the security architecture and check whether the change does not harm the overall security "plan". (If it does, changes of architecture will be needed.)

### 6.1.2 Iterative Development

Iterative development is a low-level (i.e, feature-level) development stage; it aims to produce the feature package. The entire application is delivered by multiple iterations. During an iteration, the development consists of the following steps.

#### Design by feature

**Design by feature** is a per-feature activity to produce the design of each feature. In this step, the selected features are specified and designed in detail. Security functions (e.g, authentication) have been identified in the previous stage and are treated as same as functional features in this step.

#### Risk assessment

Features are granular functions expressed in *"client-value terms"* [3] using a template: $< action > < result > < object >$, for example, displays a list of a database query's results to the user. Hence, performing risk assessment on the design of a feature is not difficult — the scope of risk assessment is a feature; the threats are always the actions to the objects. The objective of this high-level risk assessment is to ensure that the known security vulnerabilities are addressed in the design.

Although the mitigation plan of security risk is listed at the end of risk assessment, good communication which involves customers, developers and security experts, is required because security decisions have to be made. The security experts explain the vulnerability to the customers, and the customers decide the impact; then security experts and developers leave mitigation methods to customers; finally customers decide how to treat risks with the help of developers and security experts.

After risk assessment, the original design of a feature may be altered. The new design will be checked against the security architecture and implemented in next step.

### Build by feature

Once the design of a feature is assessed by security experts, the feature is built in this step. This does not introduces new obligations for security development, except for referencing security programming guidelines for a particular programming language, such as [85].

### Security inspection

Unlike **Risk assessment**, the code is inspected in this step. The development activity in this step is security testing. Plain FDD does not cover the whole process of software testing [11]. In this step, the developers check the code to ensure that they avoid the common security errors associated with the programming language. If security errors are reviewed by this process, the developers will have to fix them. (or at least accept the risk of not doing so).

### Summary

The iterative development stage is one of the key features where the integrated approach is different with plan-driven security development. Features, including security functions, are designed and built in an iteration. In addition to plain FDD, there are two security checks: risk assessment focuses on the design; and security inspection reviews the implementation.

The following sections describe the details of these steps and the activities in the steps. In order to avoid the repetition of detailed description of activities which have already been explained in plain FDD textbook [3], the following focuses on the newly added activities in secure FDD. But for the reason of the familiarity to plain FDD users, the steps of secure FDD are introduced using the same structure as in the textbook [3]. The description consists of four parts:

- Description — A description of a step in the method.
- Entry Criteria — A list of requirements for starting and completing the tasks of the step.
- Tasks — Descriptions of the tasks in the step. Each task is boxed and noted whether it is required or optional. The tasks which are already listed in plain FDD [3] are not included.
- Exit Criteria — A list of results when completing the tasks of the step.

## 6.2 Secure FDD in Details: Linear Development

The following explains the details of each step in sFDD. But first, let's look at the stage of linear development, which encompasses **Design an overall model, Design security architecture, Build a features list,** and **Plan by feature** (see Figure 6.1).

### 6.2.1 Develop an Overall Model

#### Description

Plain FDD [3] does not explicitly address the issue of collecting and managing requirements. It assumes that documented requirements such as use cases or a functional specification exists. The development process of the overall model is the process of *"domain walk-through"*, i.e, that the scope, contents and requirements of the overall problem domain are determined, and divided into different areas; then object models for each of the domain areas are built.

But to develop a secure system, there are additional things to do before the development. For example, the customers and project managers need to consider the evaluation of security, the management of the security environment including the development environment and the operation environment. These key security decisions are made in this step.

First of all, the goal of this step is to give an overall blueprint of the project. The modelling activity at this step involves the following.

- The domain expert provides user stories and answers questions related to functional requirements,
- The chief architect builds a class diagram in UML to describe the fundamental architecture,
- The development team discusses the overall model,
- The chief architect refines the overall object model.

The development starts with a series of modelling activities where UML is heavily utilised. Plain FDD initially used coloured UML [135]. Coloured UML [135] is regular UML with colour-encoded classes. All the classes are divided into four different types; each type has its own colour. The auxiliary classes and interfaces are colourless.

- *Yellow.* A role being played, usually by a person or an organisation. For example, the user of the estate agency site may play different roles: it can be a buyer, a seller or system administrator.

- *Blue*. A catalogue-like description. For example, the estate agency site may have descriptions of the properties it sells, such as flat, house etc. The description gives all the characteristics of the property, but it is not the property itself.
- *Green*. A party, place or thing. In the case study, the actual property would be modelled using green. The green class usually has some identifying attributes, such as serial number, identification number etc.
- *Pink*. A moment in time or an interval of time usually associated with some business process. For example, the result of user's query may be shown a pink class, since it is generated dynamic, which is tracked by the estate agency system.

The advantage of using coloured UML is that it allows rapid understanding of the problem domain's dynamics. The chain of *pink* classes represents the main flow of business. The *blue* descriptions and the *green* objects usually surround the *pink* chain. The *yellow* roles are normally added to the design at the beginning. They are often removed later if a person is playing only one role.

A good understanding of the target system is also the basis of security development. During the process of drawing overall models, the team begins to consider security issues. The key issues which have to be decided in advance are:

- *The evaluation of security*. The evaluation of functionality relies on testing, which is an integrated phase of SDLC. The evaluation of security is always done in an extra phase. For highly critical systems, security is often evaluated against international security evaluation criteria; but the security of many commercial systems is assessed based on customer requirements. The evaluation of the target system has to be decided in advance because it will affect the following development activities. For example, if evaluating security against security criteria, the security functions requested by these criteria have to be considered in the phase of detailed design.
- *The management of development and operation for security purposes*. Discipline and management during the development and operation of the target system are very important factors for its security. The management of security development and operation is a broad field related to asset management, physical security and human resource safety functions. It entails the identification of an organisation's information assets and the development, documentation and implementation of policies, standards, procedures and guidelines. Some activities of security management, such as security role assignment, organisation of security team, overlap with the activities of project management. There are also a series of references about security management, for example white-papers [158, 159].

Decisions on these two issues are crucial to the entire development process. A clear decision on security evaluation and a good plan of security management will build a sound foundation for the ongoing project.

### Entry Criteria

In this step, the requirements, including functionality and security, are specified and documented. Unlike plain FDD, initial security requirements have to be identified before the development starts.

### Tasks

In the step, **Develop an Overall Model,** there are two overall tasks:

- *Develop overall models*
  The activities of how to build up overall models have already been introduced in plain FDD [3]. By utilising modelling technique (i.e, coloured UML), overall models help developers understand the context of services in the system.

- *Consider issues of security management and evaluation*
  The managers and developers have to make some security decisions. The big decisions include the management of security development, the method of security evaluation, and administration of security operation. Among them, security evaluation will directly and significantly affect the following development activities.

### Exit Criteria

Several documents containing overall models are required in plain FDD [3]. The additional documents in sFDD are about security decisions:

- *Documents for security management.* These documents are usually informal for a small project, but they should contain enough material to demonstrate that these management issues are carefully considered.

- *Agreement of security evaluation.* Neither customers nor developers can individually decide how to do the security evaluation; it needs good communication and negotiation. At the end, the method of security evaluation and the degree of security should come to an agreement. This agreement will dominate the philosophy of the entire security development.

### 6.2.2 Design a security architecture

#### Description

A security architecture is a plan that describes the security functions required to meet the needs of security protection. Security functions should provide protection against possible threats to the system. Ideas for security functions come from security requirements and technical guidance, including security evaluation criteria (e.g, [8]) and industry guidances (e.g, [83, 65]). Because of defense-in-depth, the degree of protection which security functions provide depends on the desired degree of security which is a decision of security evaluation made in previous step.

A security architecture is a key characteristic of sFDD. The activities of developing a security architecture involves the following:

- a security team selects essential security functions with the knowledge of security evaluation;
- the experts review the overall object models and the selected security functions, and produce a system architecture that clearly describes the functions and their relationships;
- the developers discuss the revised models (i.e, overall models and security architecture) with the security team in order to have a good understanding.
- final models are refined and shown to customers. It is necessary that the customers have an awareness of the security architecture especially when extra effort outside of the system is needed. For example, network security may not be an issue in the security architecture of a Web application, but it is important to inform the customers that there are several issues not in the architecture but which may affect system security.

This step is to refine the security functions and map them to the architecture of the system. Doing so requires solid security knowledge and rich engineering experience.

#### Entry Criteria

To fulfil the tasks in this step, a set of documents are needed, including the overall models, initial security requirements, and plans for security evaluation and management.

#### Tasks

The tasks in this step are as follows:

- *Design security architecture*
  The developers propose an initial architecture of the system based on the overall model. Considering the security requirements, the security team selects available security functions and designs a security architecture which includes the security functions. This initial security architecture needs to be refined and detailed along with the object models that are evolving.

#### Exit Criteria

The product of this step is a security architecture.

- *Security architecture.* Secure FDD considers security as a system property, and a security architecture is an essential document. In this document, security functions are selected and described. The most important is that the security architecture outlines the dependency among security functions and business functions.

### 6.2.3 Build a Features List

#### Description

In plain FDD, a feature is purely a small piece of **business** logic. But with security considerations, a feature (security feature) may be different: a security feature may be a business feature with (or without) some security constraint; or a component of a larger security function.

Object models and security architectures give a good basis for building a security features list. The real process of building the list is top-down and straightforward: a feature team is formed to functionally decompose the entire system in the security architecture into **subject areas**; then analyse the **business activities** within **subject areas**; and finally analyse the **steps** within each **business activity**. Each **step** is a security feature, and it is also a step in the overall development process.

Unlike features produced in plain FDD, a security feature may be bigger because the team of security development has to consider: 1) what are the protections provided by the feature? 2) what are the consequences from its use? The first question aims to generate ideas about how to authorise and enforce the action presented in the feature; the second question generates ideas about the side-effects of the feature and how to deal with them. These questions often make the dependencies among security features complicated; the feature team has to prioritise them after they are identified.

### Entry Criteria

The security functions in the security architecture are identified. For example, to build a Web application for an estate agency there are security functions, namely authentication and logging, which are provided by a Web server.

### Tasks

In secure FDD, the objective of this step is to create a features list. The list includes not only the features from functional development, but also the security functions. The tasks are:

- *Build a features list*
  This is an iterative task involving the security team and functionality feature team. The teams identify the set of features using knowledge obtained from the previous step. In general, features and the assets on which each feature operates are located in the layers of the application architecture (i.e, UI, PD, DM, and SI). According to this mapping, the dependency relationships among features are classified into *within layer* and *crossing layer*. The operation of most features only depends on the other features on the same layer. But some of them rely on the features located in a lower layer. For instance, feature *retrieve a query result* in PM layer may depend on some features in DM layer. Meanwhile analysing the effects and side-effects of security functions, new features may be introduced, beyond those for the original requirements.

- *Document the features*
  The documentation of features aims to make a reference for planning and future development. In practice, the development of these features is on the basis of business activities. All features of a business activity are documented in one sheet. On that sheet, the business activities are described firstly; then all features in that business activities are described.

- *Review the security architecture*
  When analysing features from a security point of view, new features may need to be introduced. These new features are always added to mitigate the side-effect of "old" features. For example, to turn on the authentication function of Web server may affect user's access to existing pages. There will need to be a new feature to tune the existing build of the application. To review the security architecture is to check whether the new features fit well into the security. If there is a conflict, it is easier to correct the problem at a higher level and at an early stage.

#### Exit Criteria

The product of this step is a features list, which is always longer than the one that resulted in plain FDD because new features will likely have been added for security purpose.

### 6.2.4 Plan By Feature

#### Description

This is the last step in linear development. Once security features are identified, a planning team comprising the customers, managers of the project and development, and chief programmers is formed to consider the sequence of the development, based on dependency and complexity of features. Then jobs are allocated to development teams. This is similar to the previous step, **Build a features list**, in timing. Sometimes, these two steps are carried out as one; there need be no significant milestone between these two.

The process of planning is to simplify, weight and sequence features for development. Each entry (work package) in the plan will be a task of a development iteration. The iterations are usually short, *"from a couple of weeks to a couple of months, with a preference to the shorter timescale"* [25]. The real difficulty in this step is to balance the priority of business function and security function. A security feature may be a pure business function; or a security function which provides (a part of) protection to the entire application; or a combination of both. One thing to be addressed here is that the customers should be aware of and be willing to accept an *insecure* release from early iterations. That is mainly because security is a system-wide issue. The security protections are implemented (or employed) iteratively according to the plan and the security architecture produced in the previous step is a framework to ensure that this plan is completed eventually.

In practice, a work package contains the work of building one or two business activities, depending on the estimation of their implementation. In order to gain more control and security of the development process, each security feature within the business activity is assigned to a owner.

#### Entry Criteria

The entry criterion is a comprehensive features list.

### Tasks

The objective of this step is to produce a plan of development. The tasks of this step are:

- *Determine the development sequence*
  The planning team shall assign a date for completion of each feature. The completion date and the development sequence is based on the dependencies, complexity, and workload of features.
- *Assign ownership*
  The owner of a security feature can be an individual or a team; ownership should depend on the complexity of the feature. Plain FDD [3] explains the duties of the owner of a business feature, which is to ensure the quality of a feature from design to code. In addition, the owner of a security feature is also responsible for verifying the effects and side-effects of security.

  Meanwhile, the jobs of developing features are assigned to development teams. The leader of each team is responsible for performing a security inspection after a new release is built for the system at the end of each iteration.

### Exit Criteria

The product of this step is a development plan. In this plan, the following should be explicitly stated or determined:

- *Work Package*, a series of development activities.
- *Business Activity*, the business activity to be built in this iteration.
- *Owner of Security Feature*, the owner of every features to be developed.
- *Leader of Development Team*, the leader who is in charge of the development in this iteration.
- *Completion Time*, an estimation of completion date.
- *Note*, a blank column at the stage of planning; but contains informations about actual progress (e.g, on time/delay; the brief reason of delay and real completion date, if delayed.).

## 6.3 Secure FDD in Details: Iterative Development

At the end of planning, a sequence of development iterations is determined. In each iteration, there are four steps: **Design by feature, Risk assessment, Build by feature,** and **Security inspection**. Among these, **Risk assessment** and **Security inspection** are newly added in sFDD.

At the beginning of an iteration, the development team prepares related information for a selected business activity, then follows a design-cycle to develop features one by one. There are eight milestones to manage the development progress. Among these, six are defined by plain FDD [3]; and two are introduced in secure FDD.

These milestones are:

- *Domain walk-through.* The experts give an overview of the domain area for the feature to be designed. They should consider all related but not all implemented information for the iteration.
- *Design.* The developers develop diagrams for the features to be designed. In addition, all alternative designs, design decisions, requirements clarifications and notes are also recorded and documented in the design package.
- *Design inspection.* Design inspection aims to remove errors from the design. It is inspected by the project team members. The inspection often leads to revising the design. On acceptance a to-do list is generated per affected feature, and the items in that list are assigned to feature team members.
- *Risk assessment.* Once the design is thought to be error-free, the security team members perform a security risk assessment process and examine the design. This security assessment works on a high level model (i.e, conceptual level) to identify possible security risks and judge the acceptability of each risk in the design.
- *Coding.* The developers implement the design to satisfy the requirements of the iteration.
- *Code inspection.* A successful code inspection (which also includes a successful completion of unit test) is the verification of the output of this process. The coding work is done individually; however the code is inspected by the other members in the team. Ideally, the code is as good as that written by the best developer. The unit test aims to ensure all requirements of the iteration are satisfied.
- *Promote to build.* Features are only promoted to the build after a successful code inspection. This is the integration point for the entire iteration.
- *Security inspection.* This milestone is added in sFDD. After features are integrated into the build, members of the security team should check the entire system against the security requirements to determine whether unacceptable risks have arisen due to integration of features.

These milestones of the development progress are evaluated by the leader of the development team. These milestones also indicate the major tasks of these steps. Table 6.1 summarises the steps of sFDD and their milestones.

Compared with plain FDD, there are not many changes to the *"old"* steps — **design by feature** and **build by feature**, but security features may be considered differently

| Secure FDD step | Milestones |
|---|---|
| **Design by feature** | Domain walk-through<br>Design<br>Design inspection |
| **Risk assessment** | Risk assessment |
| **Build by feature** | Coding<br>Code inspection<br>Promote to build |
| **Security inspection** | Security inspection |

**Table 6.1:** Secure FDD Steps and Their Milestones

with business features (when considered). **Risk assessment** and **security inspection** are brand new steps in sFDD, and they will be explained in detail in the following.

### 6.3.1 Design by Feature

#### Description

This is a per-iteration activity to deliver the design of a package of features. A number of features are scheduled for development by assigning them to a development team. If there are multiple development teams, they may work in parallel.

#### Entry Criteria

The planning process has completed. All development iterations have been scheduled.

#### Tasks

The major tasks of this steps are:

- *Domain walk-through*
  The domain expert gives an overview of the domain area for the feature to be designed. This should also include domain information that is related to the feature but not necessarily a part of its implementation. This may be an optional task based on the complexity of the feature and its iterations.

- *Design*
  The feature is designed. The design results should be documented and checked into a version control system if possible. Alternative design, design decisions, requirements clarification and notes should also be documented, if there are any.
- *Design inspection*
  A design inspection is held to examine if there are any flaws in the design. The design may be altered, so there may be a mini-loop inside of this step in order to ensure that the deliverable of this step is verified.

#### Exit Criteria

The result of this step is a successfully inspected design package. There is no new type of document delivered in this process compared with plain FDD. The design package comprises:

- A covering memo, or paper, that integrates and describes the design package such that it stands on its own for reviewers.
- The referenced requirements (if any) in the form of documents and all related confirmation memos and supporting documentation.
- Design diagrams.
- Design alternatives (if any).
- Modification of design (if any).

The next thing is to carry on the risk assessment of the design package.

### 6.3.2 Risk Assessment

#### Description

Risk assessment is not only a process of identifying risks and their characteristics, but also a determination of the exposure to risks. Chapter 2 listed some security risk assessment methods. Secure FDD does not mandate any particular form of risk assessment method. But in general, risk assessment is performed in an iterative, incremental, time-boxed, and ongoing manner.

#### Entry Criteria

The features are designed and inspected against relative requirements.

### Tasks

Any risk assessment method can be used in sFDD as long as it refers to the recommendations of NIST [32]. The tasks of risk assessment include:

- *Understand the feature contents*
  The contents include system boundary, business functions, assets at risk, business processes at risk, threats to these assets and business processes.
- *Identify the attacks, evaluate the impacts, and analyse the risks*
  This is the body of risk assessment. It comprises the following sub-activities:
  - List the potential vulnerabilities.
  - Analyse the vulnerabilities of the assets and business processes to these threats.
  - Estimate the risks' probabilities of occurrence.
  - Estimate the potential impact of each risk to the success of the endeavor.
  - Estimate the importance and priority of each risk.
  - Categorise the risks.

  This step involves substantial estimation efforts; overall, the job of risk assessment needs experienced people who know security risk assessment and understand the design well.
- *Recommend specific actions and techniques to each significant risk*
  Risk assessment is not only to identify the risk, but also to provide choices which can mitigate the risks to customers and developers. Customers have to be involved because they have to judge the degree to which risks can be accepted.
- *Evaluate the recommendation of risk assessment*
  The recommendations from risk assessment may introduce new features to be built. Before these new features are introduced into the development iterations, these new features have to be evaluated and inspected to ensure that they are right for purpose and correct (i.e, no more risk) security functions
- *Assign responsibilities and resources to perform risk avoidance and document the risks*
  Once the mitigation decision is made, the reaction of mitigation is planned. Developers may redo the jobs of **design by feature** to integrate the new feature suggested by **risk assessment.**

### Exit Criteria

This step works on the design package and refines it. In addition, the results of risk assessment include a report about the security risks and their mitigation. Once the mitigation is integrated into the design, the developers can begin to write code.

### 6.3.3 Build by Feature

#### Description

This step aims to implement the design of feature package. Starting with the design package, the development team writes the necessary codes to support the design for features. Then the code developed is unit tested and inspected — the order of which is determined by the team leader. After a successful code inspection, the code is promoted to the system.

#### Entry Criteria

The design process and risk assessment have been completed. That is, the design package has successfully been inspected and assessed for business and security reasons.

#### Tasks

The major tasks of this step are:

- *Coding*
  The developers implement the code to satisfy the requirements of the design for features.
- *Code Inspection*
  A code inspection is held before or after the unit test; the decision is made by the team leader depends on the complexity of features. There are many references, such as [85], to be used as guidance for coding and code inspection.
- *Unit Test*
  The team tests the code to ensure all requirements of features are satisfied.
- *Promote to the Build*
  Because the features are designed and implemented individually, the features of one iteration are integrated into the build.

#### Exit Criteria

The results of the process are:

- Classes and methods that have been successfully inspected.
- Classes that have been promoted to the system.

### 6.3.4 Security Inspection

#### Description

Security is an emergent property. The security architecture is a plan of security implementation, but its existence cannot guarantee that there is no conflict when security features are built into the system. A security inspection is used to ensure that the new features which are promoted to the system will not bring new risks to the system. In contrast to other steps in iterative development, a **security inspection** works on the entire existing system, rather than features just developed.

#### Entry Criteria

The process of implementation has finished. That is, the features are implemented and promoted to the system.

#### Tasks

The major task of this step is:

- *Security Inspection*
  Once code is successfully inspected and tested, it is promoted to the existing system. A security team may apply methods to examine the entire system, including existing application, hardware, and network etc. There are many reference texts about security of a Web system, such as [65, 83, 113]. In addition, a complete risk assessment can also be performed to fulfil the task.

#### Exit Criteria

This is the last step in a development iteration. The result of this process is a *reasonably* secure system. At the end of a development iteration, the system may not fully satisfy the relative security requirements because security protections may be implemented in successive iterations.

## 6.4 Summary of Secure FDD

The analysis of plain FDD can be found in [3, 11]. Plain FDD has many values: it is simple; focuses on development; produces an overall model and model of feature. Secure FDD inherits all these values, and introduces security architecture, and security assessment in the development iterations. The following will explain the similarity and differences between pFDD and sFDD.

### 6.4.1 Similarity

Secure FDD has many similarities with plain FDD.

- Secure FDD is a simple method with a few steps; supports feature based development; it has very short iterations.
- Second, secure FDD is a general process model. It does not define the detailed practices used in the method. Like plain FDD, it does not define how to model the problem domain, carry out risk assessment, and inspect the design and code. It provides a flexibility to suit the method to different kind of projects.
- Third, secure FDD encourages the involvement of customers, especially when evaluating the risks. Good communication between different part of development team (i.e, between security-related staff and non-security staff) is important.

### 6.4.2 Differences

The major difference between secure and plain FDD is the addition of security development activities into the process. The other differences are side effects of considering security:

- Secure FDD considers implementation earlier than plain FDD. The information needed for implementation includes platform architecture, implementation language, deployment plan.
- The milestone and efforts of iterative development are different because of the extra work involved with security inspections.
- Plain FDD does not cover system testing and deployment; but secure FDD considers the deployment and final risk management. Secure FDD always has more iterations if there are new risks found during risk management.

Since there are some differences between plain FDD and secure FDD, a new question emerges, namely whether secure FDD is still an Agile method. The next section will address this question.

# 6.5 Conclusion

Plain FDD is an Agile method, but it has been argued by some agilists as being "less Agile" [160] due to the fact that FDD uses many traditional software development practices (e.g, up-front design). In particular, FDD involves planning and up-front modelling and design. For addressing security concerns, the planning and modelling included with FDD provides a solid basis to assist security development. Indeed, up-front modelling and design are normally necessary for security development. The key element in FDD is the domain object model. When developers learn of requirements they start forming a mental blueprint of the system, making assumptions and estimating on that basis. Developing an overall domain object model forces those assumptions out into the open, so that misunderstandings are resolved and a more complete, common understanding is formed. From a security perspective, a security architecture is the key element. In secure FDD, the security architecture is embedded in the object models and is also considered in the planning sub-process. FDD is described as a "right first time" approach to agility [160]. Hence secure FDD is a *"right first"* step towards agile security engineering, by integrating security risk assessment with the FDD process.

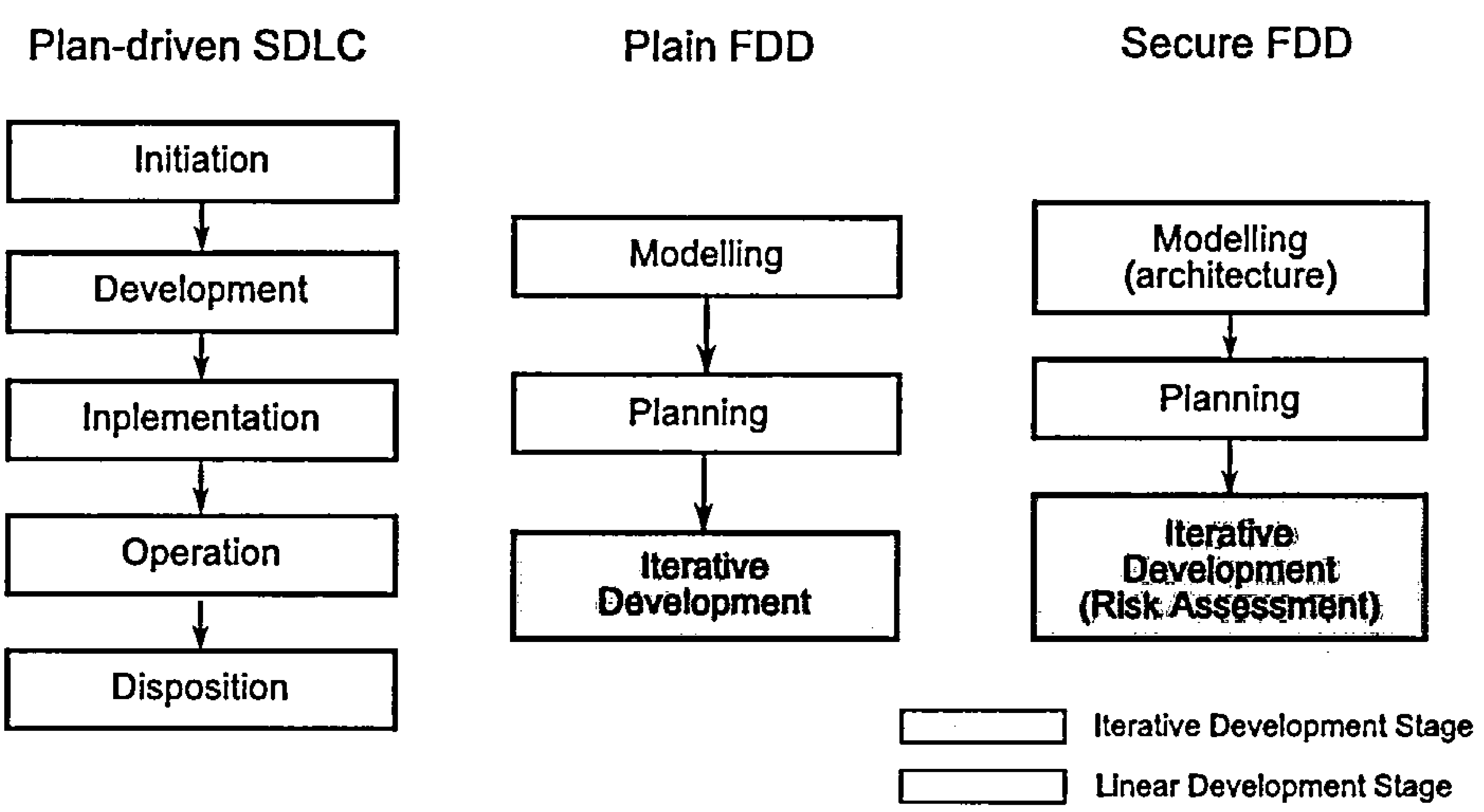



**Figure 6.2:** Plan-driven SDLC, Plain FDD and Secure FDD

Figure 6.2 highlights the differences between the process of plan-driven SDLC, plain FDD and secure FDD. In the diagram, stages have been separated by shading into two types:

- Iterative Development Stage. This kind of stage contains features and includes a plan to state how the development is iteratively completed.
- Linear Development Stage. This kind of stage does not have features, but optionally it may have tasks. The tasks in this kind of stage are always done only once.

The details of sFDD were presented in this chapter. Next, a small case study is used to demonstrate the adoption of sFDD. To support the case study, it will be helpful to make use of suitable tools, which can help reduce the costs of management of the process and its artifacts. There is a tool — Project Management Application of FDD (FDDPMA) — for the management of software projects based on plain FDD method. In the next chapter, this tool and the role it can play in sFDD are explained. However, the majority of the next chapter focuses on the development of case study.

# 7 Case Study

*The previous chapters in this part of the thesis presented the problems of security integration and described the approach taken to combine security and agile development. In this chapter, a case study will be introduced to demonstrate how sFDD could be applied in a real project, and an evaluation of sFDD is provided based on the experience of case study.*

## 7.1 Introduction

Due to its characteristics (discussed in Chapter 3), a Web application is a good domain to evaluate an Agile method. This chapter focuses on the development of a Web application; more precisely, its business logic and security features. Effort on the graphic design (i.e, user interface of Web application) is omitted in the description of the case study. This chapter demonstrates how sFDD is followed in a case study and evaluates the method based on experience from the case study. This chapter is structured as follows:

- *Overview.* This section will briefly describe the requirements of the case study, including functional requirements and security requirements. Criteria to ensure project success are also included.
- *Implementation.* This section is about how to develop the system following sFDD, including the preparation and development process. In order to evaluate sFDD's agility, the second part introduces a new requirement to the case study.
- *Evaluation.* This section presents an overall evaluation of the case study, focusing on two parts: security enhancement and agility. As well, it explains the limitations of the case study.

## 7.2 Overview

Secure FDD does **not** intend to deliver security-critical information systems (i.e, higher security level systems evaluated by security criteria); nor does any current Agile method.

The case study is designed to show security enhancement as well as the ability to achieve incremental and iterative development.

The case study is simple. An estate agency (HouseHunter) wants a Web application to allow its properties to be searchable by potential buyers; then a buyer can put offers on the property he/she is interested in. The working scenario is that a user browses the Web site, fills in the search criteria, then submits a query. When the results of the user's query come back, the user is authenticated and can place an offer on a property. In terms of security, the offer function is strictly limited to authenticated and authorised users. Therefore, there are several failure scenarios that have to be considered. One typical failure scenarios is the redirection of any unauthenticated access to the index page.

### 7.2.1 Initial Requirement Analysis

Neither plain FDD [3] nor secure FDD explicitly addresses the issue of requirements gathering, they assume that documented requirements already exist. The documentation of requirements occurs before the development starts. But in order to demonstrate the integrity of the development process, the analysis of requirements is in the description of the case study.

#### Functional Requirements

Secure FDD does not explicitly favour any modelling language and technique for requirement analysis, but a general Object-Oriented (OO) modelling technique is applied in this project because it is suitable and familiar.

The business logic of the target application is simple. It is straightforward to identify the three use cases of the application (shown in Figure 7.1[1]).

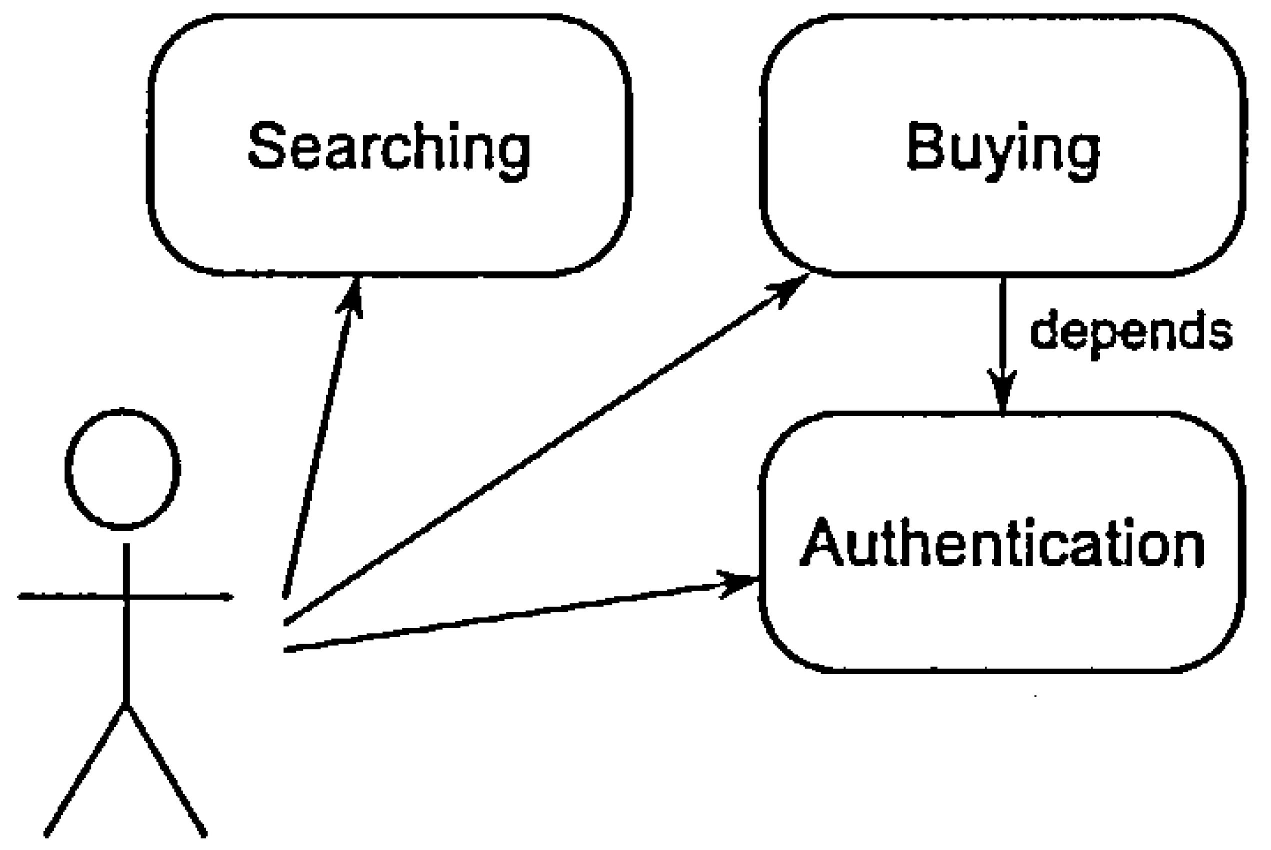



**Figure 7.1:** Use Cases of the Application

[1]Figure 7.1 is a simplified UML use case diagram.

Figure 7.1 also indicates that the use case **Buying** depends on the use case **Authentication**. This dependency relationship helps the design of the system architecture, as well as the planning of the development process.

The use cases of the application are briefly described in the following. The details of these use cases (with security considerations) are listed in Appendix A.

- **Searching,** which allows a user to input search criteria that are used in the process of querying for properties; and displays the results to the user.
- **Authentication,** which verifies that the user is registered.
- **Buying,** which permits an authenticated user to submit an offer on a property.

### 7.2.2 Initial Security Requirements

The hypothesis of this thesis declared that the integrated approach is proposed to deliver a user-accepted secure enough Web system. The role of customer is emphasised in the sFDD process because the involvement of the customer (e.g, at requirement eliciting and risk assessment step of sFDD) is one of the agile software development principles [25].

Secure FDD is not intended to build a security-critical system which is evaluated against security evaluation criteria, such as the Orange Book [8]. In the case study, the initial security requirement is elicited via a checklist approach. In Chapter 3, Table 3.2 and Table 3.3 listed the common vulnerabilities of a Web application. The preparation of security requirements is based on this list. Some entries on the list may not be applicable to every use case. The following are the initial security requirements of each use case of Figure 7.1.

#### Searching

- *Input Validation.* Inputs must be validated before processing. Input is a potential source of vulnerability (see Table 3.2).
- *Sensitive Data.* Because there is a great deal of information about properties stored in the database, allowing a user to search the entire database increases risk. So an initial security requirement is that there is protection of sensitive data so that only necessary data is public to each user of the searching service.
- *Exception Management.* Exception management is a fundamental requirement in any system. Poor handling of exceptions can lead to the loss of integrity or leakage of structural information of database in this use case. So there is a requirement that the service should catch all exceptions, especially those generated by database

operations, and in the process it should not display any sensitive information to the user.

- *General Logic Error.* Any unwanted functions may be explored to attack the system. So it is required that the design and implementation of service functions should be inspected for logic errors; for example, that there is no insecure data flow.

### Authentication

- *Input Validation.* The amount of input to this use case is much less than to the use case of **Searching,** but the operations of this use case target on one of the most critical database tables in the application. The success of any SQL attack in this use case may cause serious impact to the secure and correct operation of the entire application. The input should be validated in this use case.
- *Authentication.* Authentication is a functional requirement of this use case too. This use case also provides the authentication function to the entire application.
- *Session Management.* Once a user is authenticated, the credentials of the user are stored and transmitted in sessions. Therefore, management of the session becomes important.
- *Sensitive Data.* Any information about the credentials of users is sensitive data. Therefore secure management of sensitive data is also important in this use case.
- *Auditing.* A good user access control involves the auditing of every user's activities. Authentication is the first activity recorded in the log.
- *Exception Management.* Like the analysis in use case **Searching**, the exceptions of this use case may leak information of database structure. So it is necessary to have good management of exception for this use case.
- *General Logic Error.* Like other use cases, general logic errors have to be detected and mitigated for this service.

### Buying

- *Input Validation.* The database operation of this use case has *write* permission. Any attack may alter or destroy the database. Hence input validation is also very important for this service.
- *Sensitive Data.* Offers on properties are sensitive data. Only individuals who are authenticated and authorised should have access rights to such data.
- *Auditing.* Because it is a modification operation of database, auditing is very important for security protection and database backup. So there is a requirement for auditing for this use case.

- *Exception Management.* Like other use cases, there is a requirement for this use case to have good management of exceptions.
- *General Logic Error.* Like other use cases, the use case **buying** should be free of functional logic errors.

By analysing these initial security requirements, it was determined that these security requirements can be layered into two levels depending on how they are treated in the architecture: some requirements are common for the entire application, such as exception management and general logic error; some are particular for one use case, such as authentication and auditing.

The general security requirements for the entire application are more ambiguous than the others. This is because these requirements rely on the design of functionality more heavily than others, and the design of functionality is not clear at this moment. During the process of sFDD, these security requirements will therefore be refined in the design and implementation phases.

## 7.3 Preparation of the Case Study

Once the initial requirements are collected, the sFDD process can start. Like any other software project, there is preparation before the start of project. The preparation includes selection of relevant tools and preparation of the development environment.

### 7.3.1 Tool Support

In the case study, there are two tools used: Project Management Application for FDD (FDDPMA) and WebScarab. FDDPMA helps to manage the progress of the development. WebScarab [2] is a tool to analyse applications that communicate using the HTTP and HTTPS protocols. In particular, it can be used to help to test the security of the system. These are briefly described next.

#### Project management with FDDPMA

Project management with tool support is helpful because the tool automates repetitive tasks, prevents mistakes, and it is essential for large projects. The Project Management Application for FDD (FDDPMA) [4, 161] was developed for plain FDD, as a project at Harvard University. It was subsequently released on sourceforge.net[3].

[2] see http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
[3] http://sourceforge.net/projects/fddpma

FDDPMA is a Java Web application that manages software development. It facilitates iterative development by reducing FDD management overhead, producing graphical progress reports, and providing a place where all the FDD related documentation can be collected.

FDDPMA consists of a number of components and services, shown in Figure 7.2. The boxes without background denote components. Each component is a relatively independent part of application which usually has its own user interface and implements business logic. The boxes with gray background[4] are services which do not have a user interface but provide some useful functions to other components. Figure 7.2 also shows the relationships between components and services. The line points from the consumer to provider of a service.

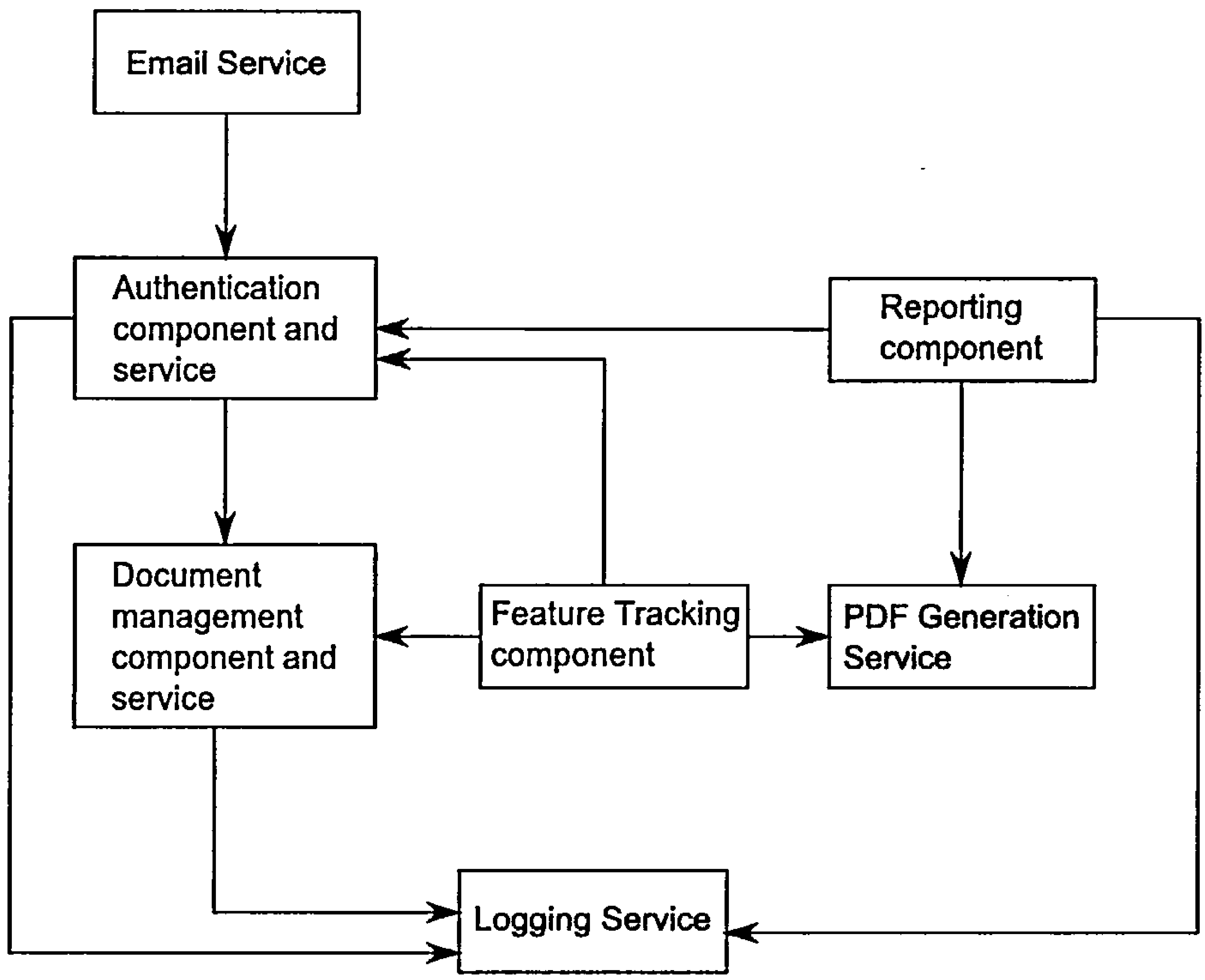



**Figure 7.2:** FDDPMA Components and Services [4]

To run FDDPMA requires Apache Tomcat[5] and the MySQL database management system.

Initially, FDDPMA can be applied to four kinds of roles in the development team:

- A project manager, who logs into FDDPMA to view the software progress reports.
- A chief programmer, who uses FDDPMA to initiate the project with a list of features, groups them into work packages, and manages development iterations.

[4]The original figure is coloured, and the background is in yellow.
[5]http://tomcat.apache.org/

- Developers, who can access FDDPMA to obtain the information published for work packages and to participate in forum discussions.
- A customer, who logs into FDDPMA to answer problem domain questions in the forums, report bugs, and watch progress reports.

In the case study of this thesis these four roles are taken by the author due to the nature of the research. Therefore there is an obvious limitation that the communication among roles and the management of team cannot be accurately and adequately assessed by the case study; other issues, such as the agility of development process, are evaluated.

FDDPMA is designed for projects of plain FDD, but it is sufficiently flexible to be customised for projects of secure FDD. In this case study, I aim to use the tool to help the management of the project. The following changes to FDDPMA have been made:

1. *Create a new project type.* FDDPMA is designed to support plain FDD. To be adopted in the project of sFDD, a new project type has to be made which contains three stages[6], shown in Figure 7.3. Each stage of an sFDD project consists of sFDD steps: **Modelling** consists of *Develop an overall model, Design security architecture*, and *Build a features list*; **Planning** consists of *Plan by feature*; and **Iterative development** consists of *Design by feature, Risk assessment, Build by feature*, and *Security inspection.*

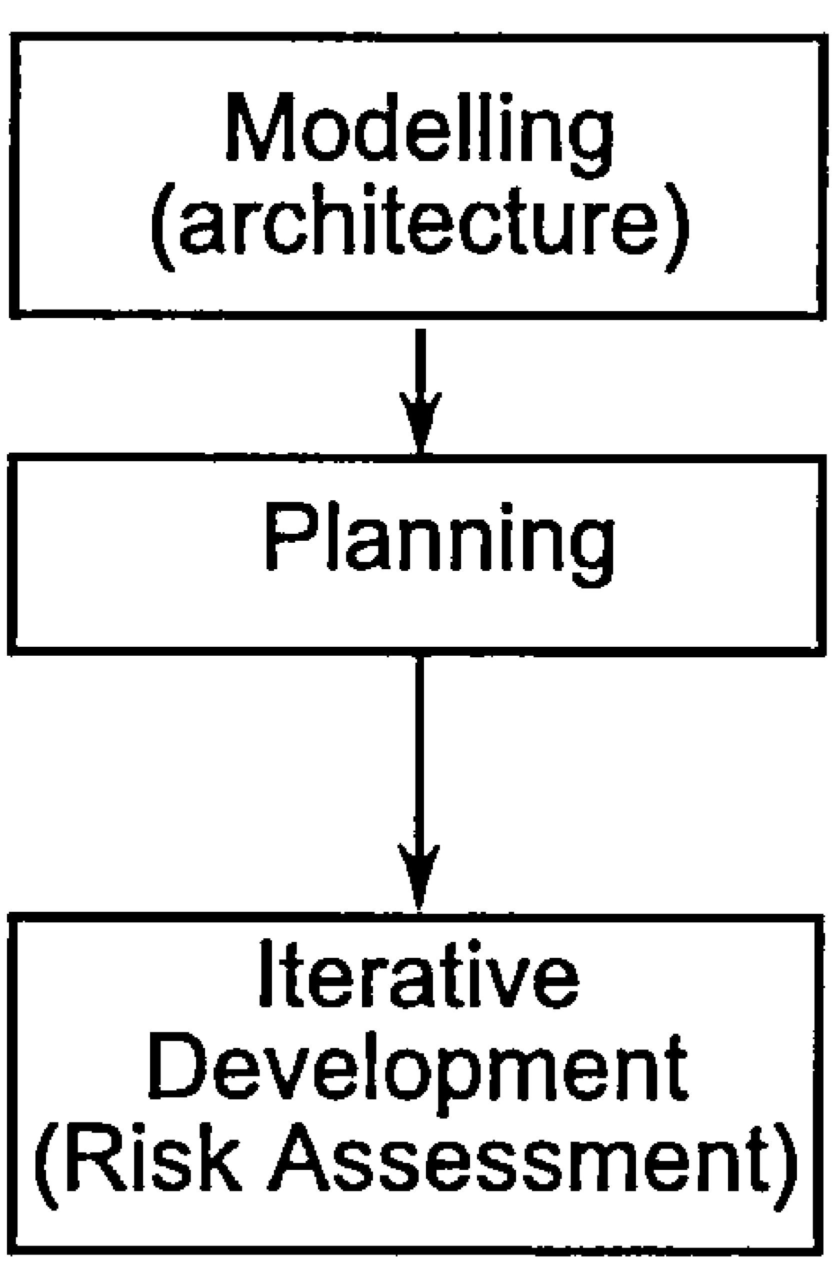
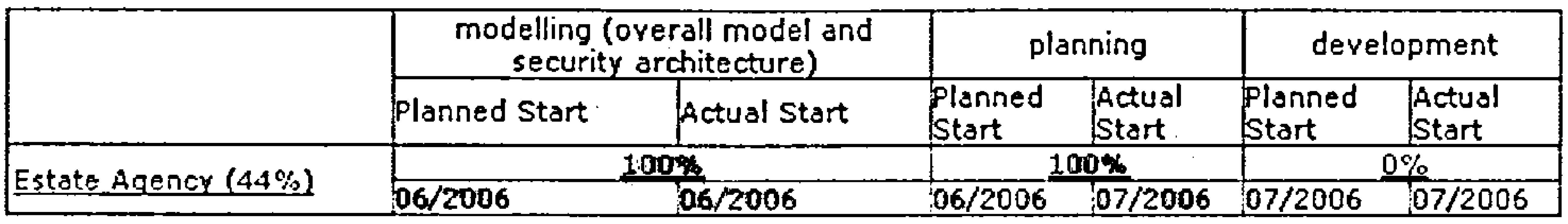



**Figure 7.3:** Development Stages of Case Study

2. *Create a secure FDD project and assign the roles of development team.* The case study is a sFDD project involving only one person, so all roles are assigned to the author.

In the case study, the tool helps to keep a record of information about development progress, deadlines and class ownerships. Figure 7.4 is a screen shot illustrating how the project is managed by the tool.

[6]The extra explanation added in brackets in Figure 7.3 is to distinguish from the stages of plain FDD in original FDDPMA.



| | modelling (overall model and security architecture) | | planning | | development | |
|---|---|---|---|---|---|---|
| | Planned Start | Actual Start | Planned Start | Actual Start | Planned Start | Actual Start |
| Estate Agency (44%) | 100% | | 100% | | 0% | |
| | 06/2006 | 06/2006 | 06/2006 | 07/2006 | 07/2006 | 07/2006 |

**Figure 7.4:** Screen Shot of Project Management

Figure 7.4 shows that there are three stages on the project **Estate Agency** which currently finished 44%. It also displays the progress and the planned and actual start data of each stage.

### Security Testing with WebScarab

WebScarab is an OWASP project — a powerful, free, open-source tool for reviewing web applications for security vulnerabilities. It is a framework for analysing applications that communicate using the HTTP and HTTPS protocols. It is written in Java, and is thus portable to many platforms. WebScarab has several modes of operation, implemented by a number of plug-ins. In its most common usage, WebScarab operates as an intercepting proxy, allowing the operator to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser. WebScarab is able to intercept both HTTP and HTTPS communication. The developer can also review the conversations (requests and responses) that have passed through WebScarab.

In the case study, WebScarab is used as a tool to help the testing of the application during the step **Security Inspection**. It has been used in its simple mode, a Web proxy. When testing the application, the Web pages are browsed via WebScarab. WebScarab hijacks the requests and responses, analyses whether there are any comments and scripts in the page, and detect whether there is some attacks, such as any possible SQL injections, Cross-site Scripting (XSS), and Carriage Return Line Feed (CRLF) injections. The vulnerabilities opened to these attacks are all in the OWASP top-ten list [117]. The results of all analysis will be shown in the interface of WebScarab, e.g, Figure 7.9. With the help of WebScarab, the developer can ensure that common vulnerabilities (i.e, vulnerabilities listed in OWASP top-ten list [117]) are considered so that the application is acceptably secure.

## 7.3.2 Environment and programming language

Last but not least, the application is built on a platform which includes many other components, including network and hardware, supporting software, and programming language.

#### Network and hardware

The case study is built in an environment of a closed local network, which consists of four network connected computers. One computer is used to host a Web server and the application; another is used as a database server; the others are temporarily used to simulate the connections of Internet users.

#### Software

The Web application is the kernel of a Web system. Around the application, there are several supporting software applications or systems; for example the operating system, Web server engine, and database management system.

In the case study, the operating system is Slackware Linux[7]; the Web server engine is Apache Tomcat 5.5.9[8]; and the database management system (DBMS) is MySQL 5[9]. These software are chosen because they are mature, powerful, and well documented.

#### Programming language

Java is the programming language selected in the case study project because of author's familiarity. [162] is a key reference on guidance for Java security programming.

## 7.4 Development Process

Once the preparation has been done, the secure FDD project is ready to start. The following will describe the process of how the case study was carried out.

### 7.4.1 Develop an Overall Model

The case study aims to build a Web application. Web browsers will render its user interfaces. Design details of user interfaces are omitted. Most user interfaces are de-

[7] www.slackware.com
[8] tomcat.apache.org/
[9] www.mysql.com/

signed with pen and paper; such a low-tech approach often has benefits over high-tech modelling tools, as everyone can participate and make changes.

By applying object-oriented modelling techniques and coloured UML, the system can be modelled as shown in Figure 7.5.

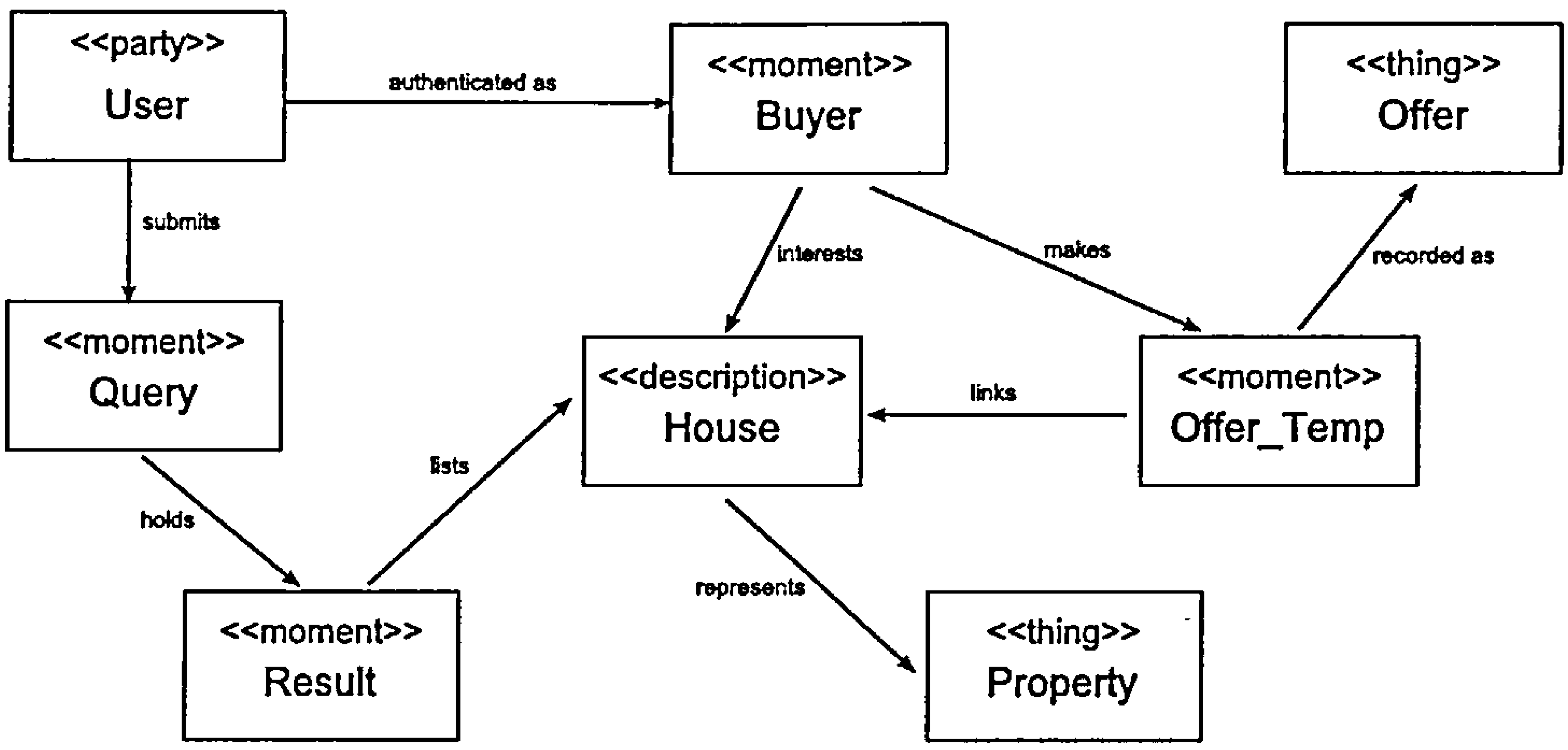



**Figure 7.5:** Overall UML Class Diagram

Figure 7.5 is a coloured UML class diagram which shows the overall class diagram of the application. The class diagram 7.5 shows how the **User** has access to the **Property** and **Offer** which are the assets of the application. Instances of **Query** and **Result** are generated dynamically; and an object of **House** contains a list of properties which satisfies the criteria of the query. When an instance of **User** wants to add an offer, the user has to be authenticated, and become a **buyer** till the end of this session; then the buyer selects a property from the list of **House** and create a temporary **offer** which will be transferred finally to the database by some database operations.

This class diagram is simplified; the attributes and functions of classes are not identified. The key points of this class diagram are to show the business logic and the scale of the application.

## 7.4.2 Design Security Architecture

The process of designing the system architecture, and later, feature identification, is one of decomposition and refinement of the application. During requirements analysis, the application has already been decomposed to the use case level; the three use cases are the major functions of the application. In sFDD, the majority of development is on a feature basis. Initially, the three functions are simple enough to be developed within weeks. Therefore, there are three blocks: **Searching**, **Authentication**, and **Buying**, which are the major functions of the application as well. Besides those, there are also several security mechanisms provided by supporting software, such as the DBMS. Figure 7.6 is the security architecture of the application.

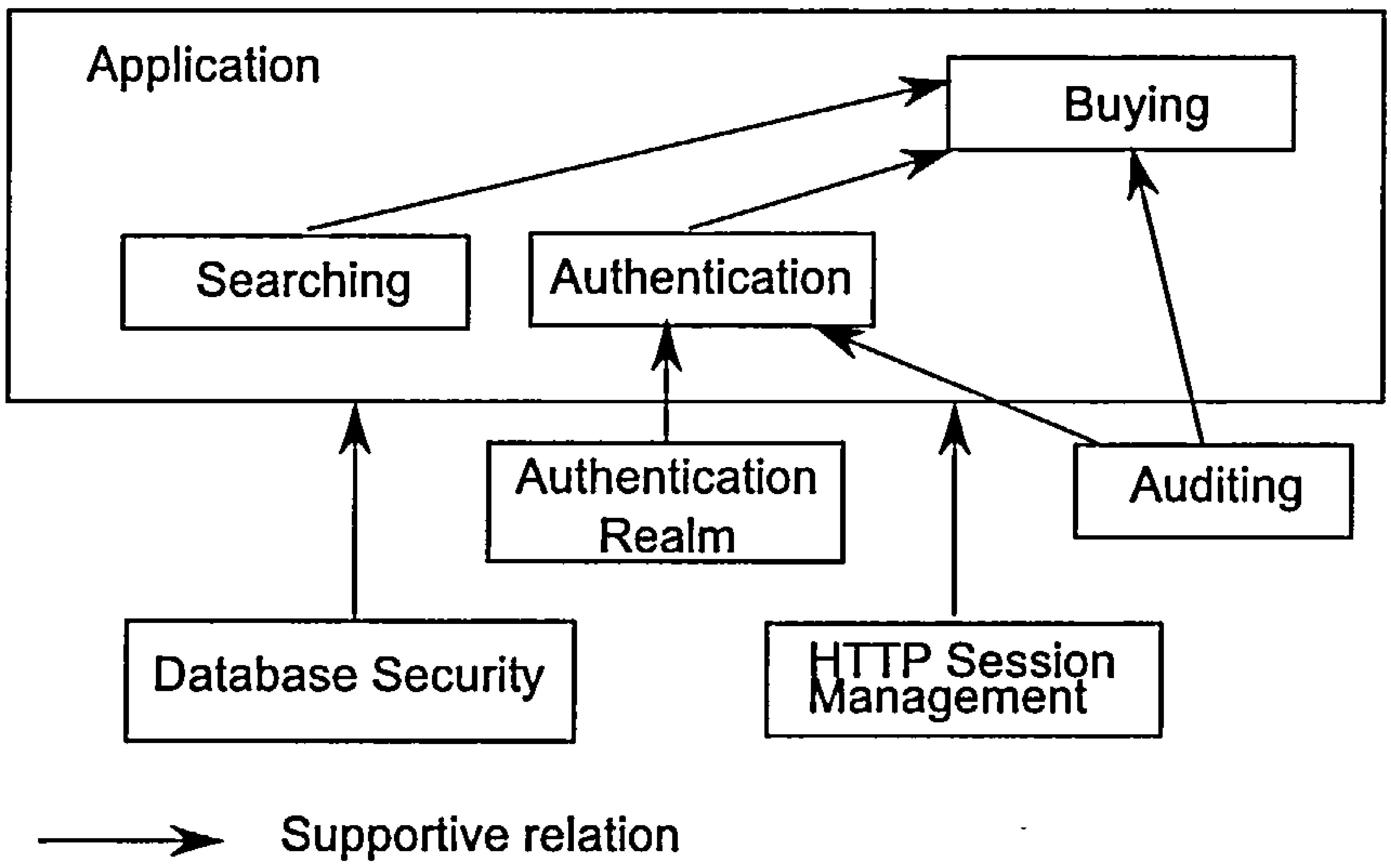



**Figure 7.6:** Overall Security Architecture

In the architecture, the **Searching**, **Authentication**, and **Buying** are three packages of the application. Among them, **Buying** relies on the functionality of **Searching** and **Authentication**. By analysing the initial security requirements, security protections of input validation have to be implemented in all three services, and these protections are designed and implemented in later steps. In addition, there are some extra security protections which are required by the initial security requirements, such as auditing, session management etc. These extra security protections are also considered in the security architecture.

In the case study, the following security protections are planned in the architecture.

- *Authentication Realm*. The Web server provides functionality to deal with the question of whether a particular request for a resource will result in that resource actually be returned. Apache Tomcat provide three types of authentication methods: basic authentication, digest authentication and database authentication [163]. The functionality of **Authentication** depends on one or more of these methods. Initially, the case study will implement the basic authentication. Once it is found that the basic authentication is strong enough, or the operation environment changes, it can be refactored in later iterations.
- *Auditing*. The extra auditing service from Apache provides the function which is required by **Buying** and **Authentication** (see initial security requirements in the previous section). It will keep records of user activities, such as database transactions, user authentication, and operations of offers.
- *Database Security*. Different from *Authentication Realm* and *Auditing*, the service of *Database Security* provides security functions to all the application (i.e, three

function packages). The DBMS (i.e, MySQL) used in this case study provides several security protections, such as role based access control. *Database Security* helps to fulfil the security requirements of input validation too.

- *Session Management.* Same as *Database Security*, the service of *HTTP Session Management* provides services to all three function packages. It is provided by Apache as well. It will be used to mitigate the security concerns about HTTP sessions in the case study.

The introduction of these extra security services will help to fulfil some of the initial security requirements. Table 7.1 summarises the initial security concerns and the contributions of these security services.

| Security Requirement | Security Service Contribution |
|---|---|
| Input Validation | Database Security (partially for SQL injection) |
| Authentication | Authentication Realm |
| Authorisation | Authentication Realm |
| Sensitive Data | Database Security |
| Auditing | Auditing |
| Session Management | Session Management |
| Exception | n/a |
| General Logic | n/a |

**Table 7.1:** Contributions of Security Service

Table 7.1 also shows that the requirements of Exception Management and General Logic Error have to be implemented during the detailed development. The extra security services cannot yet fulfil the requirements of these two.

In addition to producing the security architecture, there are also some significant security decisions to be made at this stage of development. In the case study, the development and operation of the application are carried out in the control of the developer, and are in a secure environment. The biggest decision for the case study is how to evaluate the software security of the application. In the case study, the elicitation of security requirements is based on a checklist which is the summary of several industry guidance, including [65, 2, 83]. Therefore, the evaluation of the application security should be based on the satisfaction of the requirements. However, the security features implemented in the application are also compared with the entry level of security evaluation criteria, TCSEC [8]. The purpose of this comparison is to demonstrate the security enhancement of sFDD development process; it is also an indirect evaluation of the checklist-based requirement approach with security evaluation criteria.

### 7.4.3 Build a Features List

Based on the use cases, the decomposition of the application continues. In the case study, the three use cases (**Searching**, **Authentication**, and **Buying**) are the business activities of the application. There are several features within these business activities.

#### Searching

There are two features in this business activity:

- Prepare the query clause. This feature takes a user's inputs, generates the query clause, connects to the database and submits the query to the database. In terms of security considerations, this feature has "read" permission to the database, and it should only be allowed to read the right data from database.
- Format the result. This feature fetches the results of the query, and generates a list back to the user. There are two possible types of results: a successful result of the query; or an exception thrown by database operations. With the security considerations, the exception should be handled well. Normally, it needs cover stories for some exceptions.

#### Authentication

The security architecture clearly indicates that the implementation of **Authentication** relies on the security feature **Authentication Realm**, therefore **Authentication Realm** has to be set up before the development of **Authentication**. In this business activity, there are two features:

- Tune **Authentication Realm**. This feature does not need any design work. The developer has to configure Apache, and set up the basic authentication mechanism.
- Authenticate the user. This feature completes the function of user authentication, including taking the user's ID and the password, checking the userID/password in the database, store and transfer authentication token in the HTTP session. In terms of the security considerations, the developer may consider the same mechanism of input validation and exception management as in the previous business activity. In addition, the developer also considers the mechanism of session management, including the format, the expiration, and the assessment of authentication token.

#### Buying

This business activity depends on the previous activities, but in and of itself is quite simple. There is only one feature in this business activity.

- Submit an offer. This feature completes the function of submitting an offer to database and feeding back the result of the database operations. In terms of the security considerations, security protections should be implemented to ensure the integrity of data.

### 7.4.4 Plan by Feature

The steps of Build a Features List and Plan by Feature are closely related. When the features are identified, a plan of their development is naturally made on the list of features. In practice, the feature is a relative concept: it can be decomposed to different levels of detail. The only judgement on a feature is how soon it can be built. A development iteration is about one or two weeks. In the case study, the detailed development iterations are based on the business activities.

Before the three development iterations start, there are several security features in the security architecture that have to be established because they are fundamental services to all others. The configuration of these security features need to be planned in the timetable as well. The plan of development is listed in Table 7.2.

The workload of the development can now be estimated. In the case study, the development of each feature is less than a week. Therefore, each development iteration is going to deliver the package of a business activity. The first iteration is to prepare the extra security features which do not contain any design work, so their details are omitted in this thesis. Now it is time to present the individual development iterations.

### 7.4.5 Development Iteration I: Searching

The first iteration is to build the business activity **Searching**. In the features list, there are two features to be developed in this iteration. The detailed process of feature development has four steps: *design by feature*, *risk assessment*, *build by feature*, and *security inspection*. The following documents the progress of features development in details.

| ID | Feature | Business Activity | Security Considerations | Duration |
|---|---|---|---|---|
| 1 | Tune **Database Security Mechanism** | Preparation | Security environment management | 1 day |
| 2 | Tune **HTTP Session Management** | Preparation | Security environment management | 1 day |
| 3 | Tune **Apache Auditing** | Preparation | Security environment management | 2 days |
| 4 | Prepare the query | Searching | Input validation, Access control | 3 days |
| 5 | Format the result | Searching | Exception management | 2 days |
| 6 | Tune **Authenticate Realm** | Authentication | Security environment management | 2 days |
| 7 | Authenticate the user | Authentication | Input validation, Session management, Auditing | 2 days |
| 8 | Submit new offer | Buying | Input validation, Auditing, Session management | 4 days |

**Table 7.2:** Features List and Plan

### Design by feature

This iteration is to develop the function of business activity **Searching**. Figure 7.7 is a simplified design of **Searching**, which is a part of an overall model (Figure 7.5).

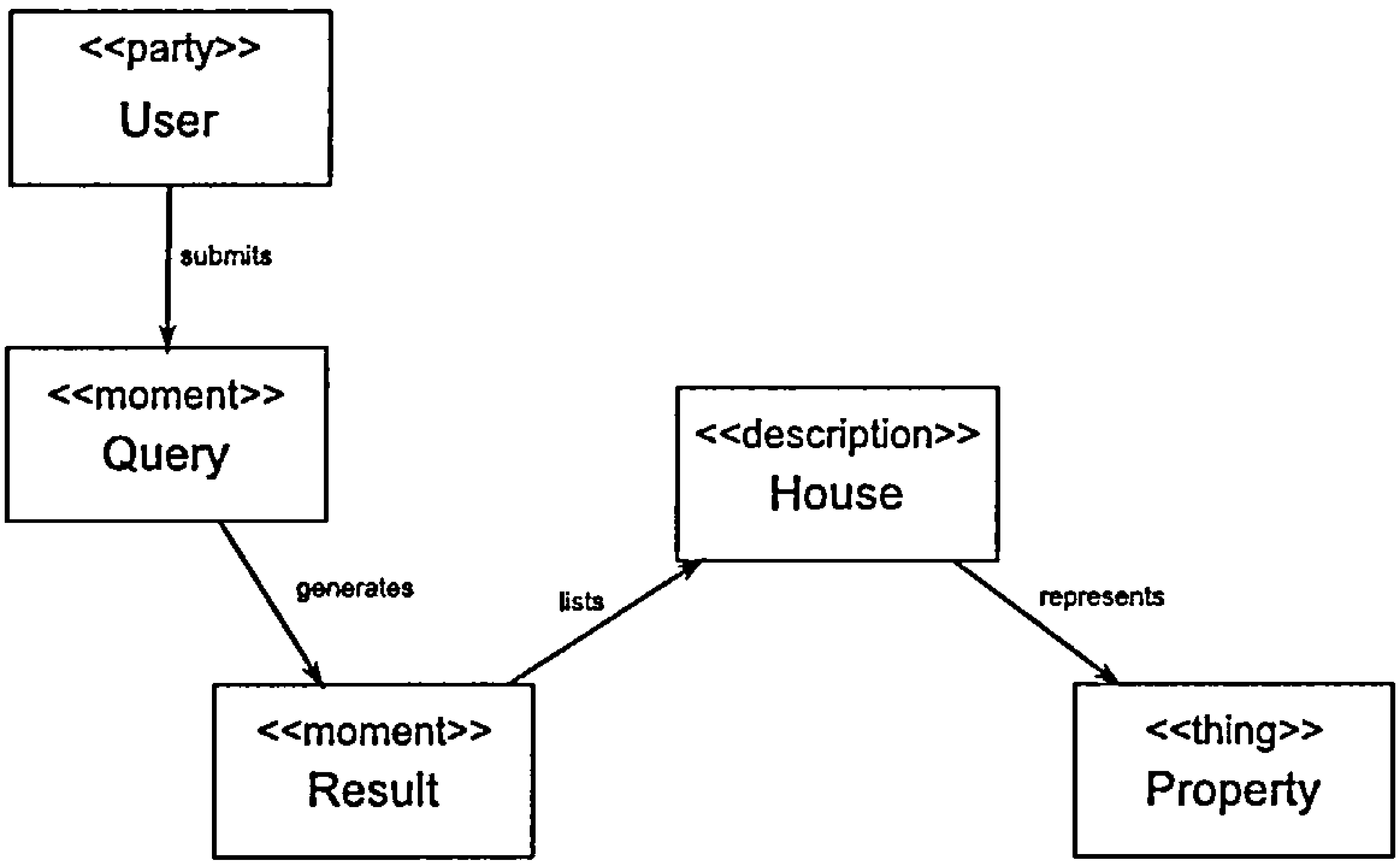



**Figure 7.7:** Design of *Searching*

Figure 7.7 does not contain details of two features: *Prepare the query* and *Format the results*. The detailed design of these two features is performed in this step. In the design of **Searching** (Figure 7.7), there are two temporary classes: *Query* and *House*, which are the key element of detailed feature design.

- Class *Query*. It is the product of feature *Prepare the query* and stores the details of the query clause. It has the following contributions:
  - Postcode, the post code used for property search. It is a String attribute. A value has to be assigned by the user when an object of Query is created.
  - Bedrooms, the minimum number of bedrooms. It is an Integer attribute. Its value setting is optional; the default value is 1.
  - MinPrice, the lower band of searchable price range. It is an Integer attribute. Its value setting is optional; and the default value is 0.
  - MaxPrice, the higher band of searchable price range. It is an Integer attribute. Its value setting is also optional; the default is 0.
- Class *House*. It is the class of the result of a database query operation. It has the following attributes:
  - Address, the address of a property.
  - Postcode, the postcode of a property.
  - Price, the asking price of a property.
  - Bedrooms, the number of bedrooms.
  - Description, a brief description of a property.

The class *House* is generated in a JavaBean (see the discussion of MVC architecture in Chapter 3, Section 3.3) because it can be reusable. The class *Result* is a copy of *House* on the client side.

The Figure 7.7 shows the business logic of **Searching**. The risk assessment will examine the design, and try to identify any unwanted flows.

### Risk assessment

The first and most important task of risk assessment is to understand the target application. The overall object model and the security architecture provide the general blueprint of the entire application. During the design step of this iteration, the developer has a good understanding of the application delivered after this iteration. Considering security, the previous requirements analysis is also based on business activities. The risk assessment is hold to identify the vulnerabilities of the application so far, which may be caused by design flaws (general logic errors) and insufficient coverage of security protections.

In the desired process of **Searching** (Figure 7.7), objects of class **Property** are the assets which should be protected against arbitrary access and modification. The result of risk assessment is documented in Table 7.3.

### Evaluate suggestions and re-design

In Table 7.3, there are two recommendations. The first suggestion is very commonly used in a database application which relies on a database management system. The second suggestion needs extra work when coding to make sure the input is validated. These two suggestions are proved by experience to be effective solutions for such kinds of security risk.

By adopting the suggestions of risk assessment, the design work is revised. An extra class, *Validated_Query*, is added in order to ensure the SQL statement generated in the application is correct. Figure 7.8 is the revised design diagram.

The revised design has to be inspected again in order to ensure that there are no new vulnerabilities introduced. Once the working package of this iteration is designed and inspected, the developer begins to implement it.

Iteration: Searching

| Risk Assessment | Description |
|---|---|
| Threat | Attacker uses SQL injection to access or modify the data stored in the database. |
| Vulnerability | The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed. |
| Risk Evaluation | The worst consequence of this kind attack may cause the loss of confidentiality and integrity of data stored in the database, or even destroy the entire database. Therefore, it may have a serious impact. The risk is not acceptable at all. |
| Mitigation | There are two suggested mitigation plans:<br>1) creating a filter between the application and the data by using database techniques, such as view. It will limit the access to the application to the database (see [14, 15]);<br>2) securing the application by enforcing input validation. It will ensure the SQL statements sent to the database are correct. |

**Table 7.3:** Risk Assessment of **Searching**

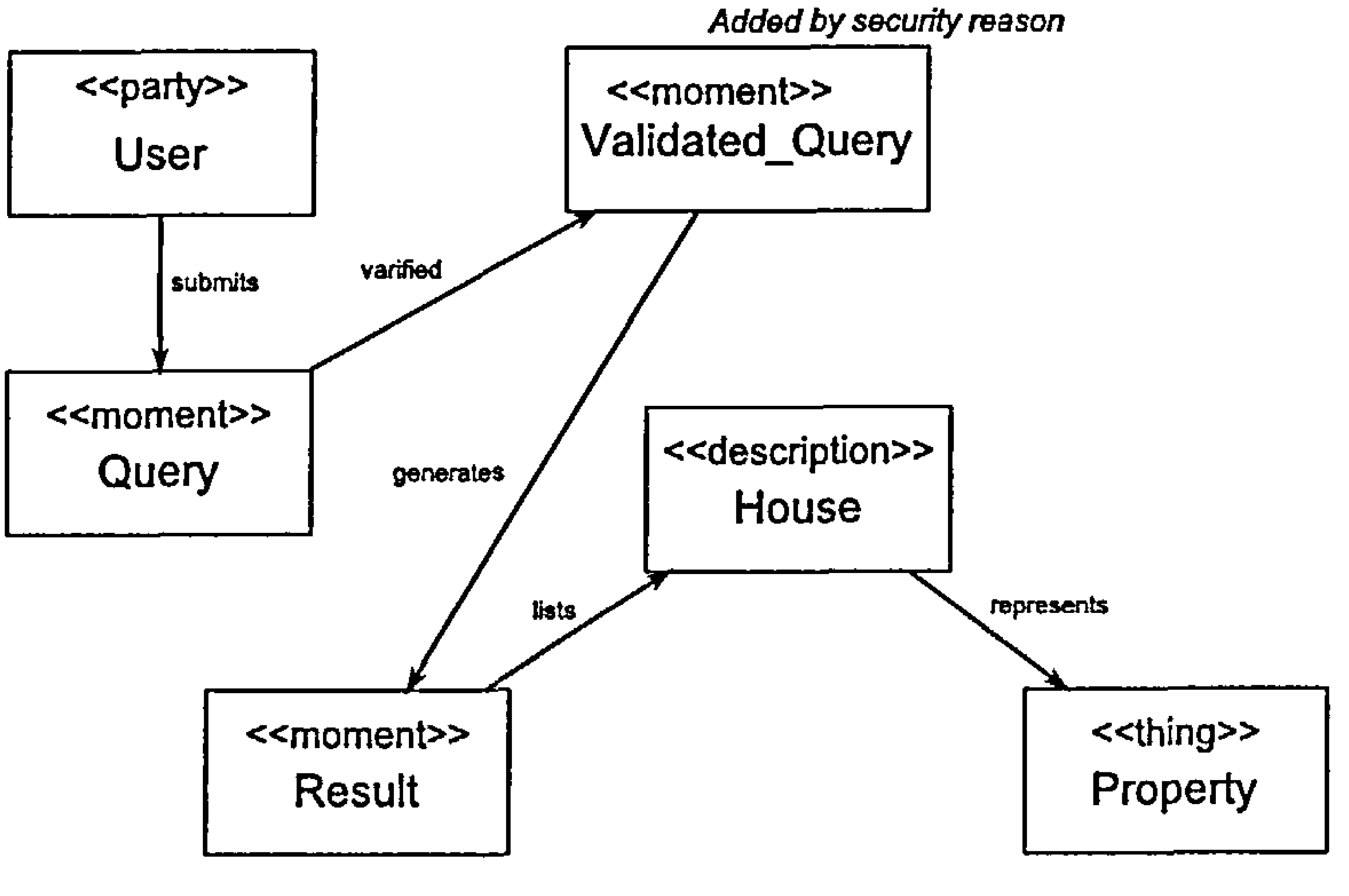



**Figure 7.8:** Revised Design of *Searching*

### Build by feature

The full source code of the implementation is not listed in this section. In order to demonstrate the implementation of risk assessment suggestions, only parts of the implementation are shown here.

**Validated Query** It has been shown in practice that an easy and efficient way to protect against SQL injection attack is to check the type of all parameters in an SQL sentence. In the case study, the parameters are transmitted via HTTP so they are of type String, which is dangerous. When reading the parameters into a JavaBean, these parameters are converted and validated.

**View** View is a database technique which is supported by MySQL. The following is the source code for generating a view in a general query for a property.

**Program 1** SQL Script of GeneralQuery

```
CREATE VIEW GeneralQuery (Price, Bedrooms, Address,
  Description, Agent)
AS
SELECT price, bedrooms, address, description, agent
  FROM property
```

The GeneralQuery only selects five data columns from the database and this information is not sensitive. Therefore, the Internet user can have access to the View. Another advantage of Views is that they are not writable. Hence, the use of Views can also help the protection against SQL injection attacks. (i.e, the user cannot modify any data in the database.)

### Security inspection

Having implemented type checking and the database view, the code is ready to be deployed. The security expert will inspect the system in order to find any remaining unacceptable risks, and may look at evidence provided in the build-by-feature step. At the end of this iteration, the security expert tests all Web pages in a browser.

By using WebScarab to test the Web site (i.e, localhost:8880), it can be concluded that there are no injection attacks. Figure 7.9 is a part of the screen shot because the whole scan result is too long.

| ID | ... | ... | ... | Path | Parameters | ... | ... | Possible Injection | XSS | CRLF | ... | ... | Comments | Scripts |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 36 | ... | ... | ... | /Adv/style_g.... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 35 | ... | ... | ... | /Adv/sign.jsp | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☑ |
| 34 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 33 | ... | ... | ... | /Adv/session... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 32 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 31 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 30 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 29 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 28 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 27 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 26 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 25 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 24 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 23 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 22 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 21 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 20 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 19 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 18 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 17 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 16 | ... | ... | ... | /Adv/housei... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 15 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 14 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 13 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |

**Figure 7.9:** Screen Shot of WebScarab Scan

### 7.4.6 Development Iteration II: Authentication

When the previous iteration is completed, the Internet user can browse the site and search for properties. There is a URL link which leads the user to the authentication part of the application, which will be developed in this iteration.

#### Design by feature

Authentication is a functional requirement as well as a security requirement. This iteration is to provide an authentication function to verify the user's inputs of their username and password; then the application validates the pair and generates a token if the pair is matched according to records in the database. The desired logic of authentication is shown in Figure 7.10.

The key issue in the detailed design of **Authentication** is about generating and storing an authentication token. In this iteration, the authentication is also implemented as a JavaBean because it may be reused. The token has a time stamp and stored in the session.

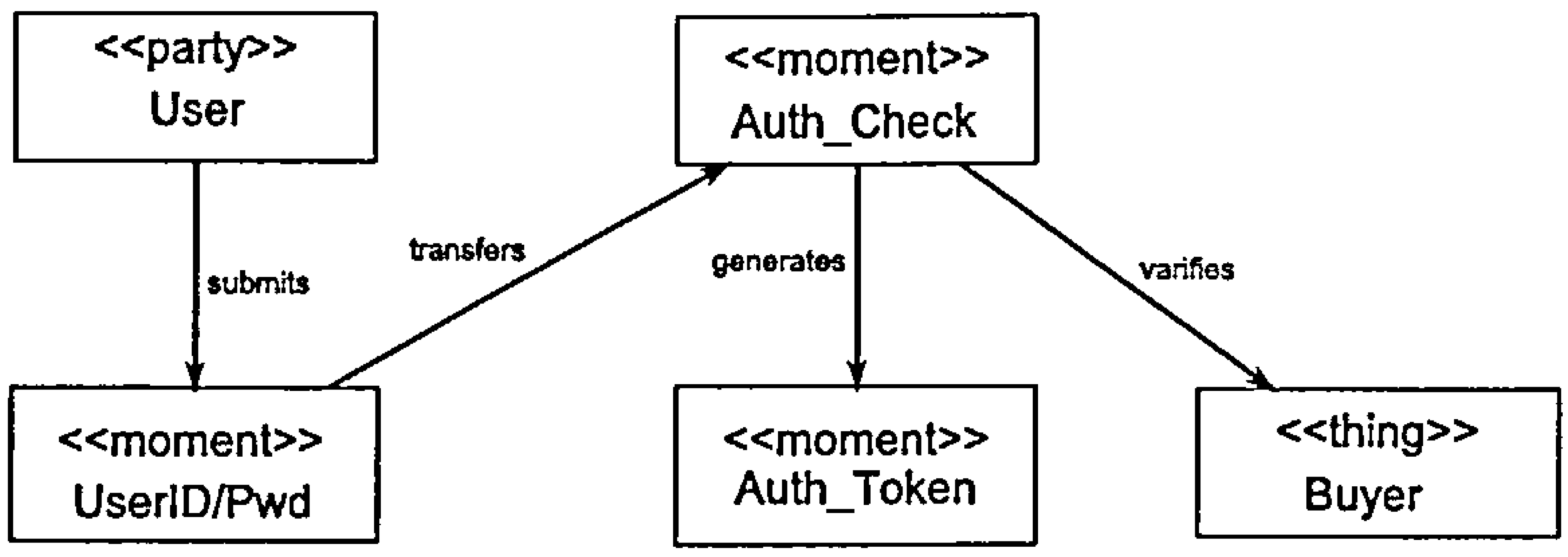



**Figure 7.10:** Design of *Authentication*

## Risk assessment

Authentication is also a security requirement so it is important to make sure there is no flaw in the design first; then assess the design to look for any risks to mitigate. It is easy to identify that *userID/Password* and *Authentication Token* are sensitive data. There is also a potential SQL Injection attack because of the query to the database. The results of risk assessment are documented in Tables 7.4,7.5, and 7.6.

Iteration: Authentication

| **Risk Assessment** | **Description** |
|---|---|
| Threat | Attacker uses SQL injection to access or modify the data stored in the user database. |
| Vulnerability | The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is validated and thereby unexpectedly executed. |
| Risk Evaluation | The worst consequence of this kind attack may cause the loss of confidentiality and integrity of user's information stored in the database. Therefore, it may lead to serious impacts. This risk is not acceptable at all. |
| Mitigation | The suggested mitigation plan is to create an filter between application and data in the database by using database techniques, such as view. It will limit the access of the application to the database. |

**Table 7.4:** Risk Assessment I of **Authentication**

Iteration: Authentication

| Risk Assessment | Description |
| --- | --- |
| Threat | Attacker steals userID and password when they are transmitted through a HTTP connection. |
| Vulnerability | The vulnerability is that the text transmitted through HTTP connection can be hijacked and may not be encrypted. |
| Risk Evaluation | The worst consequence of this vulnerability may cause the loss of authentication. Thus it may lead to serious impacts. The risk is high, but acceptable because the network infrastructure of case study is secure. |
| Mitigation | There are two recommendations to improve the security:<br>1) using strong authentication method such as digest authentication.<br>2) securing the HTTP connection by using SSL. |

**Table 7.5:** Risk Assessment II of **Authentication**

Iteration: Authentication

| Risk Assessment | Description |
| --- | --- |
| Threat | Attacker may steal an authentication token so that the authentication can be bypassed. |
| Vulnerability | The vulnerability is that the authentication token is stored in the client side, such as cookies, and there is no protection of this token against its manipulation. |
| Risk Evaluation | The worst consequence of this kind attack may cause an invalid authentication. Therefore, it may lead to serious impacts. The risk is not acceptable at all. |
| Mitigation | The suggestion of mitigation for the vulnerability is to assign a timestamp to each authentication token and automatically check it. |

**Table 7.6:** Risk Assessment III of **Authentication**

### Evaluate suggestions and re-design

The first potential security risk in the product of this iteration is from SQL injection attacks. During the development of previous iterations, the developer has already gained the experience how to deal with it. The second risk is username/password hijack. It is impossible to prevent a user losing their password. Technically, the application can adopt some mechanisms to make it difficult for an attacker to guess or steal the password. In the case study, this risk is acceptable because the operating environment is secure. The third is about authentication bypass. Normally, the session is destroyed when the initial browser is closed. But in most time, users may forget to close the browser when they leave the Web site so that the session is still alive, and the authentication token is still valid too. Therefore, it is important to add an attribute to the token in order to keep a record of when it is expired.

In the new design, a new class **Expiration** is attached to the authentication token (shown in Figure 7.11).

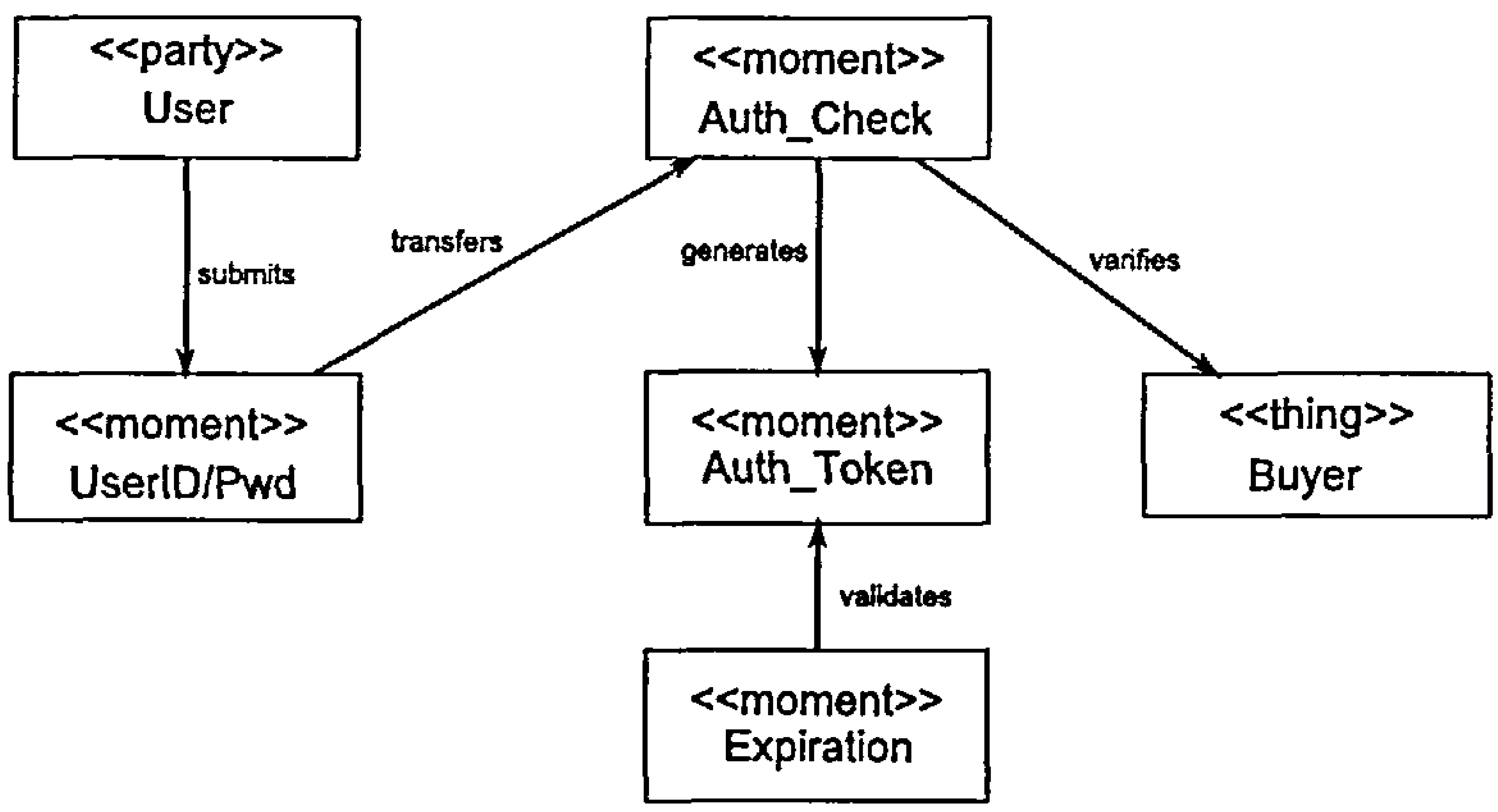



**Figure 7.11:** Revised Design of *Authentication*

Once it is successfully inspected, the new design will be implemented.

### Build by feature

The entire implementation of this iteration will not be presented in this section. Since the authentication token is the key security object, this section will explain how the authentication token is implemented.

The authentication token is implemented as a name/value pair and is assigned to an attribute of the session because it is a simple implementation. Keeping things simple is in the spirit of an agile process. The name of the authentication token is a String "authentication"; and the value is a time stamp which is assigned when the attribute is created.

In the servlet which is the controller in the MVC architecture (see Chapter 3), there is a piece of code to check the expiration of the authentication token.

**Program 2** Validation of Authentication Token

```
...
    Calendar token = (Calendar) session.getAttribute(
        ''authentication'');
    Calendar rightNow = Calendar.getInstance();
    if( token == null ){
        response.sendRedirect("login.jsp"); }
    else {
        token.set(Calendar.MINUTE,token.get(
            Calendar.MINUTE)+20);
        if ( rightNow.compareTo(token) > 0 )
            response.sendRedirect("login.jsp");
    }
...
```

In the code, the default expiration time is 20 minutes which is commonly used in most Web applications.

### Security inspection

When the iteration has been completed, the user can browse the Web site, search the properties, and follow the link to authenticate themselves. The security expert reviews all the Web pages. For example, to examine the expiration mechanism of the authentication token, the expert searches the properties, and signs in the system successfully. The application will redirect to the page of search results. Next, the security expert waits for 20 minutes, then click the link of any property listed in the result page. The system redirects to the login page again which shows that the expiration of the authentication token works.

In addition, the application is scanned by WebScarab. So far, the application is injection-free. Figure 7.12 is part of a screen shot.

## 7.4.7 Development Iteration III: Buying

The final planned iteration focuses on the business activity **Buying**.

| ID | ... | ... | ... | Path | Parameters | ... | ... | Possible Injection | XSS | CRLF | ... | ... | Comments | Scripts |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 77 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 76 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 75 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 74 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 73 | ... | ... | ... | /Adv/servlet/s... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 72 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 71 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 70 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 69 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 68 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 67 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 66 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 65 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 64 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 63 | ... | ... | ... | /Adv/servlet/s... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 62 | ... | ... | ... | /rss20.xml | | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 61 | ... | ... | ... | /safebrowsin... | ?client=nav... | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 60 | ... | ... | ... | /Adv/ | | ... | ... | ☐ | ☐ | ☐ | ... | | ☐ | ☑ |
| 59 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 58 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 57 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 56 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 55 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 54 | ... | ... | ... | /Adv/images/... | | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |

**Figure 7.12:** Screen Shot of WebScarab Scan

### Design by feature

The previous two iterations successfully built parts of the application. This iteration is to provide a function allowing authenticated users to submit offers on properties. Therefore, all accesses to this part of the application should be protected. In the previous iteration, the authentication and the expiration checking mechanisms were implemented. Based on that work, the desired flow of this business activity is shown in Figure 7.13.

In Figure 7.13, the class **House** and **Property** have already been developed during the iteration of **Searching**. The design job of this iteration focuses on class **Offer**. It has several attributes:

- User, the username. It references an entry of User table in the database.
- Offer, the offer which the user submits. It is an integer.
- OfferTime, the date and time when the offer is submitted. The data type of this attribute is String.
- Response, the response of the offer. It is a Boolean value which indicate whether the offer is accepted or not.

### Risk assessment

This feature is only accessible for authenticated users, so the functions of this feature should be checked by an access control mechanism every time the function is called. The boundary and user group for this part of the application is smaller than others. The results of risks assessment are shown in Tables 7.7 and 7.8.

Iteration: Buying

| Risk Assessment | Description |
|---|---|
| Threat | Attacker uses SQL injection to modify the data stored in the database. |
| Vulnerability | The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user inputs are not validated and thereby unexpectedly executed. |
| Risk Evaluation | The worst consequence of this kind attack may cause the loss of confidentiality and integrity of information stored in the database. Therefore, it may lead to serious impacts. The risk is not acceptable at all. |
| Mitigation | The recommended mitigation is to create a temporary table to store the initial data submitted by users, then the data in the temporary table are updated later to the main table by the database administrator or automated database programs. The contents of the temporary table can be subject to further checks before they are uploaded to the permanent table. This will protect the integrity of data because the database administrator or automated programs are more trustable than ordinary users. |

**Table 7.7:** Risk Assessment I of **Buying**

### Evaluate suggestions and re-design

The risk assessment suggests the application adopts database security mechanisms to protect the integrity and confidentiality of data because the database solution for such security problems is mature and reliable.

In this iteration, the application will write data into the database. So using a view is not sufficient enough to mitigate the risks, but it is still a good choice to limit a user's access.

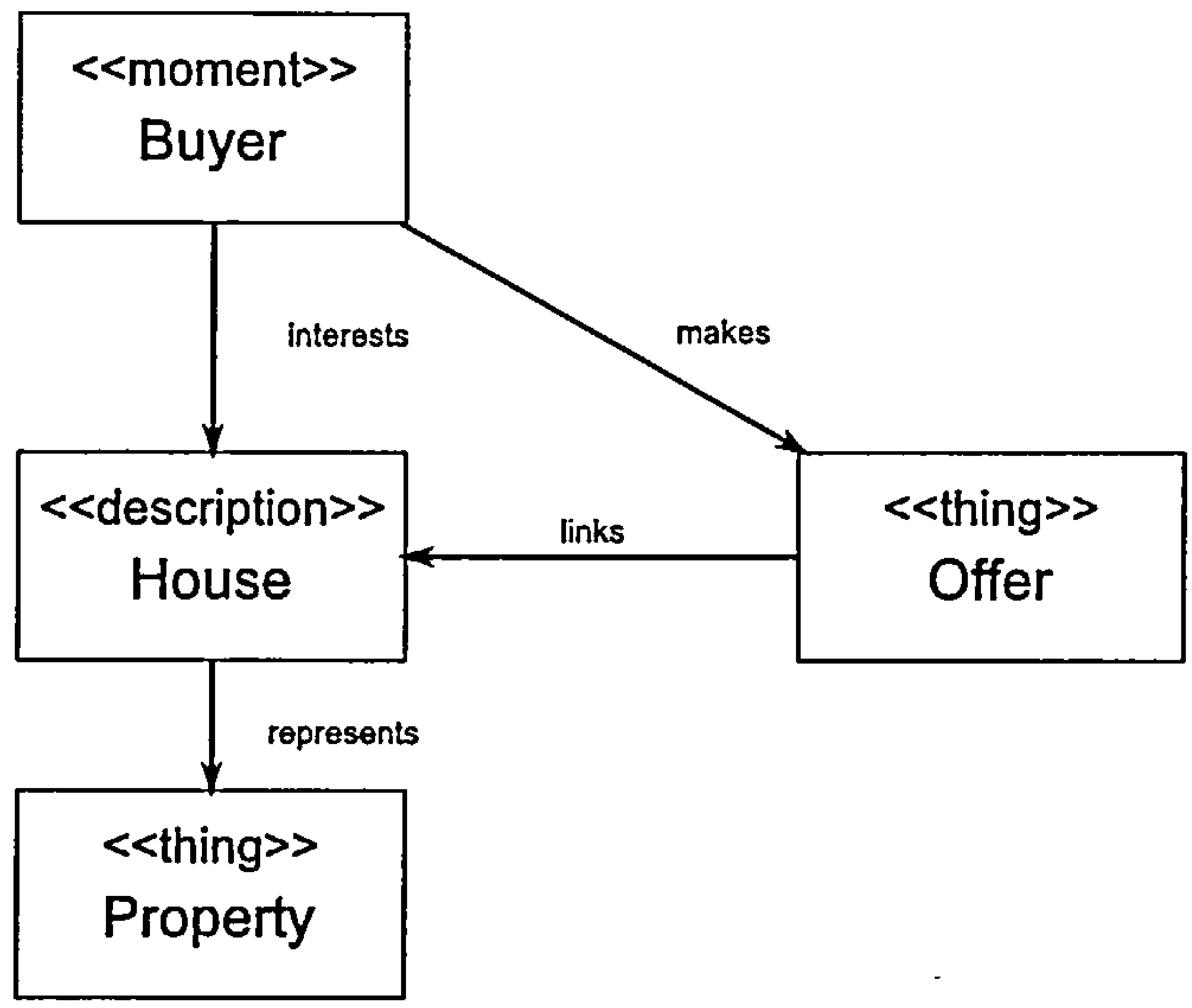



**Figure 7.13:** Design of *Buying*

Iteration: Buying

| Risk Assessment | Description |
|---|---|
| Threat | Attacker may have chance to view other data stored in the database. |
| Vulnerability | The vulnerability is present when the application submits an SQL query to the database. If the query is not validated, it will leak some data to the user. |
| Risk Evaluation | The worst consequence of this kind attack may cause the loss of confidentiality and integrity of information stored in the database. Thus it may lead to serious impact. The risk is not acceptable at all. |
| Mitigation | The suggested mitigation plans is to create a view or temporary table to store the information which are bound to a buyer. |

**Table 7.8:** Risk Assessment II of **Buying**

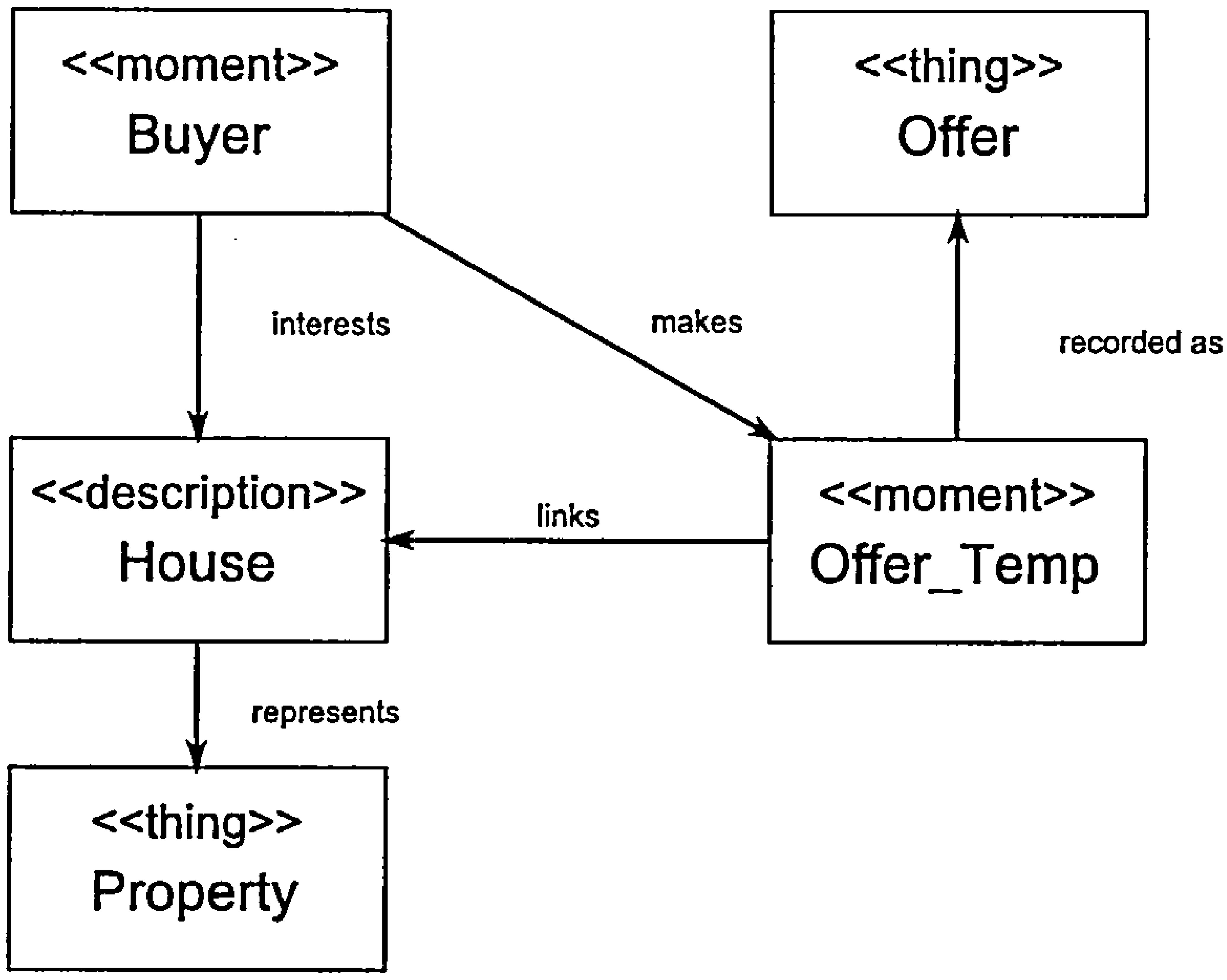



**Figure 7.14:** Revised Design of *Buying*

To mitigate the risk in Table 7.7 and protect the data integrity, creating a temporary table in the database is a cost-effective way.

The class **Offer_Temp** has same attributes as the class **Offer**. The new design is shown in Figure 7.14.

### Build by feature

The followings shows the code that insert an entry into table offer_temp.

The code also demonstrates how the application utilises the Java exception mechanism to manage exceptions. The exception is commonly thrown by programs when the *"write"* operation is executed, such as insert and update. When an exception is caught, some information are printed in order to debug the program, but when it is released, a less informative set of data will be given. The database connection has already been tested in previous iterations, so it is not re-tested problem in this iteration.

### Security inspection

The case study has been incrementally built and deployed. When this iteration is completed, the entire application has been built. The task of security inspection at this time is the entire system. The security expert tests all Web pages and links to ensure there is no obvious security problem.

**Program 3** Program of Database Insertion

```
...
try {
...
    int update_count =
    stmt.executeUpdate(''INSERT INTO offer_temp
        (user,offer,offertime) ''
        + ''VALUES(''' + User.toString + ''', ''
        + offer + '','''
        + rightNow.toString + ''')'');
    stmt.close();
}
catch( Exception e ) {
    //debug info
    e.printStackTrace();
}
finally {
    try { con.close(); }
    catch( SQLException e ) {
        //debug info
        e.printStackTrace();
    }
}
```

Figure 7.15 is a screen shot of the latest WebScarab scan. The application is examined by WebScarab to determine if there are possible injection attacks.

### 7.4.8 Evaluation of First Release

The evaluation of the first release is based on satisfaction of security requirements, which are summarised before the development starts. The following will describe how these security requirements are satisfied by the implementation of security features.

#### Input validation

The requirement of input validation is that all inputs of the application are validated. The inputs from Internet users are criteria for search, username and password, and the price of a user's offer. The validation mechanisms for these inputs are implemented in relevant work packages. Some are implemented in client-side, for example using

| ID | ... | ... | ... | Path | Parameters | ... | ... | Possible Injection | XSS | CRLF | ... | ... | Comments | Scripts |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 108 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 107 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 106 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 105 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 104 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 103 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 102 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 101 | ... | ... | ... | /Adv/offer.jsp | ?house=10... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 100 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 99 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 98 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 97 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 96 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 95 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 94 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 93 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 92 | ... | ... | ... | /Adv/submit.j... | ?house=10... | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 91 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 90 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 89 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 88 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 87 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 86 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 85 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |

**Figure 7.15:** Screen Shot of WebScarab Scan

HTML input lists, such as the inputs of bedroom number, price range, so that user can only select pre-defined (and valid) values (see Figure 7.16). Another example of client-side input validation is the validation of textual values by using Java script to checks the invalid characters in a textual string. Some input validation mechanisms are implemented server-side too. For example, the program uses several pre-defined SQL query templates which can reduce the risk that the system is attacked by SQL injections.

In addition, there is another injection attack, the HTTP injection attack (which is that the Internet user types a URL address in the Internet browser directly or generates a fake HTTP request which will bypass the application function and the validation mechanism on the client side). In the case study, the application only responds to the HTTP requests by "POST"[10] method because HTTP injection attack via POST method is much harder to carry out than the attack via GET method. And the application checks some hidden value from client to protect the system against HTTP POST injection attack.

At end of development, the application is scanned by WebScarab which shows that there is no possible injection attack in the application.

[10]In HTML, the developers can specify two different submission methods for a form: "GET" and "POST". The major difference is: the former means that form data is to be encoded (by a browser) into a URL (visible to users) while the latter means that the form data is to appear within a message body (invisible to users).

| ID | ... | ... | ... | Path | Parameters | ... | ... | Possible Injection | XSS | CRLF | ... | ... | Comments | Scripts |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 108 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 107 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 106 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 105 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 104 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 103 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 102 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 101 | ... | ... | ... | /Adv/offer.jsp | ?house=10... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 100 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 99 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 98 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 97 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 96 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 95 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 94 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 93 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 92 | ... | ... | ... | /Adv/submit.j... | ?house=10... | ... | ... | ☐ | ☐ | ☐ | | ... | ☐ | ☐ |
| 91 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 90 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 89 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 88 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 87 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 86 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |
| 85 | ... | ... | ... | /DSS/Query | ?Entitleme... | ... | ... | ☐ | ☐ | ☐ | | | ☐ | ☐ |

**Figure 7.16:** Screen Shot of Search Inputs

### Authentication

The requirement of authentication is that the authentication mechanism should be strong enough so that the user cannot bypass or break it easily. The case study adopts the most common authentication mechanism — passwords. The authentication mechanism is also integrated with Apache Tomcat which will increase the strength of the authentication mechanism. In addition, the application also successfully implements the expiration of the authentication token (see Iteration II). The usability and security are balanced by only requiring password authentication on sensitive user operations (i.e, the user can search the properties without logging on).

### Sensitive data

The confidentiality and integrity of sensitive data should be protected. In the case study, the sensitive data are client information and offers on properties. These data are stored in the database. The confidentiality of sensitive data of properties are protected by adopting database views. And the integrity of these data are protected by using a temporary table in the database. In addition, some mechanisms, such as implementation of session and exception management, are also helpful to protect sensitive data. The successful implementations of security protections ensure the satisfaction of security requirements.

### Session management

A session is a special object in a Web application. It is a common means to transmit some data between clients and the server so that there are some sensitive data stored in the session. The protection of sessions is supported by the Apache Tomcat application server. In the case study, the application is designed and implemented to manage the contents of a session, including authentication tokens, token expiration, and navigation information.

### Audit

In the case study, the user's authentication and offer's submission are recorded in a log. The auditing function is provided by the database.

### Exception management

It is required that the exceptions of the application are managed well so that the confidentiality and integrity of data are not threatened. In the case study, there are two kinds of exceptions: database operation exception, such as exception of database connection, insertion; and HTTP exception, such as wrong URL address. These database exceptions are caught by the application; and there is a separate page to display the error messages to the Internet user. These implementations ensure that the user cannot get any sensitive data from exception messages.

### General logic error

The application is generally tested by the author. The author browses pages, clicks links, and types in arbitrary text in the input boxes. No logic errors were found in the application. The design and code inspection are performed to ensure the design and implementation are right. The security review also helps to correct the problems of general logic error.

In summary, the security protections designed and implemented in the application satisfies the security requirements. The application which is the product of sFDD is sufficiently secure to satisfy the requirements.

## 7.5 A New Requirement

In order to help to evaluate the agility of sFDD, a new requirement is now introduced, after the application has been built. Tolerance to and acceptance of new requirements is considered as an important part of agile development. The application now requires to be able to share the users' information with other estate agencies. This means that there is a third-party system to deal with users' profiles and authentication. In the case study, a new authentication function is handled in a Web service and the existing application communicates with the Web service via SOAP messages[11]. In the new system, the authentication is done by a Web service. It makes the authentication service more independent, service-oriented which is a trend of new Web application.

### 7.5.1 Response to Change

Introducing a new functional requirement may affect existing functionality as well as satisfaction of security requirements. When introducing a new functional requirement, the following shall be checked:

- Whether there is any fundamental change of security requirements and the existing security architecture. The Security architecture is built on the security requirements and an overall model of functionality. The change can be fundamental or substantial to the security architecture. If the change is fundamental to the security architecture, for example, if the access control requirement changes from discretionary access control (DAC) to mandatory access control (MAC), the security architecture may be rebuilt; therefore, the development process may restart from the very beginning, and re-design security architecture.

After the check, the developer has to estimate the workload for the new change, and assign a new iteration to implement the change. In the case study, the new requirement does not change the business logic but introduces a new Web service replacing an existing function block. In this case, the developer rebuilds the **authentication** service and then integrates it into the existing system. The development process of the new **authentication** service is the same as following the steps of sFDD.

From a Web application to a Web service, the business logic behind the functionality is not changed, but the interface changes. Considering security, a Web service is self-contained. That means it handles the internal security for its own, such as access control. The application who calls the Web service only has to focus on the interface. The development process for the new **authentication** service is straightforward. The case study focuses on how to evaluate the change, and what to plan for the new development iteration, as these are new features of sFDD.

[11] Introduction of SOAP at http://en.wikipedia.org/wiki/SOAP

### 7.5.2 New Feature

Considering the new requirement, the existing features of the application, *Searching* and *Buying*, have not been affected much. The feature *Authentication* is now replaced by a Web service. The new architecture of the system is similar, as shown in Figure 7.17.

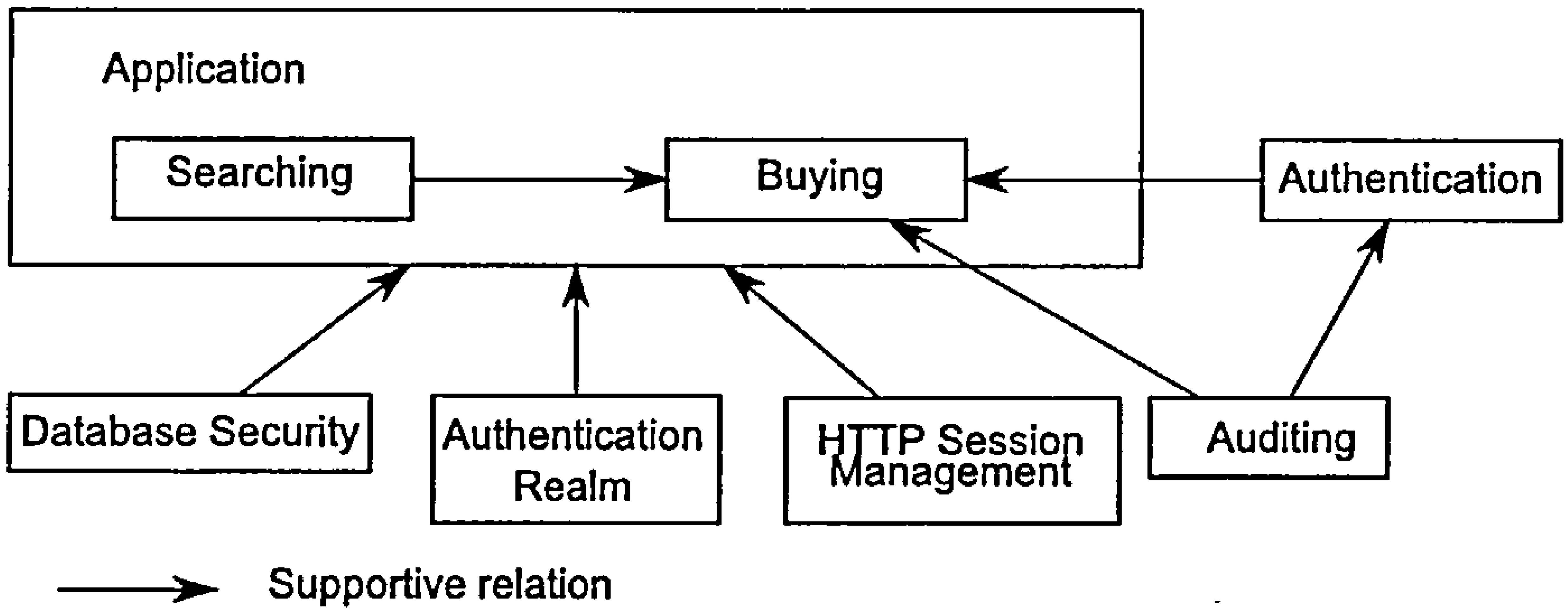



**Figure 7.17:** New Architecture

Compared with the old architecture (Figure 7.6), the difference in the new architecture is that the system boundary is smaller. Meanwhile, some supporting relations are changed too. In the new architecture, there is no fundamental conflict with the existing application when introducing this requirement because the change from a function of Web application to a Web service is not a fundamental change to the architecture. The supporting relationships are still existing, unchanged.

#### Analysis of new feature

An analysis of the change is also very helpful when considering an updated plan for new development. In the old architecture, the supportive relations within the application boundary can be mapped to URL links. But Figure 7.17 highlights a changed relation, from the new *authentication* Web service to the *Buying* feature. One significant characteristic of Web service is loose coupling. It is very easy to integrate a Web service to an existing application in Web environment. In the case study, the job took a week.

#### Design of new feature

The Web service for authentication has defined an interface: it takes a username and password as parameters and returns a text string which is the authentication token. If the authentication fails, the returned string is null.

In the case study, the Web application invokes the function of the authentication Web service. So there is still a block in the design which provides functions to handle the communication with authentication Web service and the application. The new design looks similar to the original design at a high level of abstraction.

### Risk assessment of new feature

Now that authentication is provided by an external service, communications must be secured. This inevitably means that communications must be encrypted (and so there is a need for encryption functionality). For illustrative reasons (see below) the author has chosen the Data Encryption Standard (DES), but any almost any reputable block cipher would suffice. Since communications must be encrypted issues regarding key management arise. Keys must have and maintain appropriate confidentiality and integrity. The author must ensure that keys are appropriately distributed — only the application and the web service are supplied with the shared key — requiring the choice of some distribution protocol. (Depending on the affiliations of the two parties this could even be largely manual, though many on-line protocols exist too.)

The key must be stored appropriately at the two hosts. This raises issues regarding how accessible the key is. Is it stored in a tamperproof environment? Is direct acccess appropriately policed? Similarly, actual use of the key must not provide avenues for compromise. Are there possibilities of using covert timing channels to break the key? Can the key be overused so that the sheer volume of examples of encryption allows cryptanalysis? (This introduces the need for periodic key change, which must be policed.) Is the actual cryptosystem likely to be broken soon? The choice of DES here is deliberately instructive.

In fact DES is declining in popularity with the emergence of the Advanced Encryption Standard (AES). DES is beginning to come within brute force range. Several successful challenge attacks have been carried out on DES, mostly using massive distributed computational resources or else special purpose hardware. With the emergence of very cheap and widely available programmable hardware, it is likely that DES would need to be replaced. This illustrates that the system must be periodically reassessed with respect to security. It will become just too susceptible to easy attack.

The author will not pursue this avenue further. The author simply notes that the distribution of the authentication service has non-trivial knock-on effects and that a thorough analysis of any particular key distribution and management scheme would be required.

### Implementation of new feature

The implementation of encryption is a Java class. The implementation is listed in followings:

```
public class DesEncrypter {
    Cipher ecipher;
    Cipher dcipher;

    DesEncrypter(SecretKey key) {
        try {
            ecipher = Cipher. getInstance ("DES");
            dcipher = Cipher. getInstance ("DES");
            ecipher. init (Cipher.ENCRYPT_MODE, key);
            dcipher. init (Cipher.DECRYPT_MODE, key);

        } catch (javax. crypto .NoSuchPaddingException e) {
        } catch (java. security .NoSuchAlgorithmException e) {
        } catch (java. security .InvalidKeyException e) {
        }
    }

    public String encrypt(String str) {
        try {
            // Encode the string into bytes using utf-8
            byte[] utf8 = str .getBytes("UTF8");

            // Encrypt
            byte[] enc = ecipher.doFinal(utf8);

            // Encode bytes to base64 to get a string
            return new sun.misc.BASE64Encoder().encode(enc);
        } catch (javax. crypto .BadPaddingException e) {
        } catch ( IllegalBlockSizeException e) {
        } catch (UnsupportedEncodingException e) {
        } catch (java. io .IOException e) {
        }
        return null;
    }

    public String decrypt(String str) {
        try {
            // Decode base64 to get bytes
            byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str);

            // Decrypt
            byte[] utf8 = dcipher.doFinal(dec);

            // Decode using utf-8
```

```
            return new String(utf8, "UTF8");
        } catch (javax.crypto.BadPaddingException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (UnsupportedEncodingException e) {
        } catch (java.io.IOException e) {
        }
        return null;
    }
}
```

When the implementation completes, the source codes are inspected and promoted to build on the existing system.

## 7.6 Evaluation of Case Study Against Security Criteria

It has been mentioned several times that secure FDD is not designed to develop any high integrity system. Thus, the developer may not take any security evaluation criteria into account when analysing security requirements. But it interests the author what the result is if a Web application developed in sFDD project is evaluated against any security criteria (for example, TCSEC) and what the extra security requirements are in order to achieve a security class (for example, class C in TCSEC). The purpose of evaluating the case study against an international security evaluation criteria is to give the user of sFDD a flavour of to what degree of security a Web application can be developed by following the process of sFDD.

### 7.6.1 Security Evaluation

Security evaluation aims to not only evaluate the security level of the application, but also to evaluate the development process of the application. Normally, security evaluation of a system is to classify the system's security level using security criteria. The case study in this thesis is a small size Web application, operated in a closed network environment. The security development for case study is fully based on understanding of industry guidance [2, 65]. The security requirements are also from these industry guidance.

Because there is no plan to evaluate the application against security criteria, the developer has no idea which security level the application can achieve. In Orange book [8], Division C, Discretionary Protection, is the entry level of a secure system. The following paragraphs give a brief introduction of Division C in Orange book [8].

### Brief introduction of Level C

In the Orange Book [8], the security categories range from D (Minimal Protection) to A (Verified Protection). Division C is an entry level of security. There are two classes in this division: Class C1 — Discretionary Security Protection and Class C2 — Controlled Access Protection.

**C1 - Discretionary Security Protection** In the Orange book, class C1 defines following requirements:

- Discretionary Access Control. The evaluation target shall define and control assess control between named users and name objects in the system. The enforcement mechanism shall allow users to specify and control sharing of those objects by named individuals or defined group or both.
- Identification and Authentication. The evaluation target shall require users to identify themselves to it before beginning to perform any other actions that the system is expected to mediate. Furthermore, the system shall use a protected mechanism to authenticate the user's identity. The system shall protect authentication data so that it cannot be accessed by any unauthorised user.
- System Architecture. The system shall maintain a domain for its own execution protects it from external interference or tampering. Resources controlled by the system may be a defined subset of the subjects and objects in an environment.
- System Integrity. Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the system.
- Security Testing. The security mechanisms of the entire system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorised user to bypass or otherwise defeat the security protection mechanisms of the system.
- Security Features User's Guide. A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the system, guidelines on their use, and how they interact with one another.
- Trusted Facility Manual. A manual addressed to the administrator of the entire System shall present cautions about functions and privileges that should be controlled when running a secure facility.
- Test Document. The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the the security mechanisms were tested, and results of the security mechanisms' functional testing.
- Design Documentation. Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the system. If the system is composed of distinct modules, the interfaces between these modules shall be described.

**C2 - Controlled Access Protection** System assigned class C2 enforce a more finely grained discretionary access control than C1 system. The following are minimal requirements:

- Discretionary Access Control. The requirements of class C1 is the basic requirements of class C2. In addition, the target system shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorised access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorised users.
- Object Reuse. All authorisations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.
- Identification and Authentication. In addition to the requirements of class C1, the target system shall be able to enforce individual accountability by providing the capability to uniquely identify each individual user. The target shall also provide the capability of associating this identity with all auditable actions taken by that individual.
- Audit. The evaluation target shall be able to create, maintain, and protect from modification or unauthorised access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the system so that read access to it is limited to those who are authorised for audit data. The system shall be able to record the following types of events: use of identification and authentication mechanisms, introduction or objects into a user's address space (e.g., file open, program initiation), deletion of objects, and actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The system administrator shall be able to selectively audit the actions of any one or more users based on individual identity.
- System Architecture. In addition to the requirements of class C1, the system shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.
- System Integrity. It is same as in class C1.
- Security Testing. In addition to the requirements of class C1, testing shall also include a search for obvious flaws that would allow violation of resource isolation,

or that would permit unauthorised access to the audit or authentication data.

- Security Feature User's Guide. It is same as class C1.
- Trusted Facility Manual. In addition to the requirements of class C1, the procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given.
- Test Documentation. Same as in class C1.
- Design Documentation. Same as class C1.

In summary, the security requirements of the classes in Division C are compared in Table 7.9.

In Table 7.9, three types of criteria may be present for each requirement. Each will be preceded by the word: NEW, CHG, or ADD to indicate the following:

- NEW: New criteria appears since this class and higher classes.
- CHG: The criteria have appeared in a lower class but are changed for this class.
- ADD: The criteria have not been required for any lower class, and are added in this class to the previously stated criteria for this requirement.

Meanwhile, two abbreviations are used as follows:

- NR: (No Requirement) This requirement is not included in this class.
- NAR: (No Additional Requirements) This requirement does not change from the previous class.

Table 7.9 clearly shows that the security requirement changes from class C1 to class C2 are that: object reuse and audit are new in class C2; there are something additional about identification and authentication, security architecture, security testing, and trusted facility manual; there are something added and changed about access control but the fundamental mechanism is still discretionary access control. Next, there will be an evaluation of the case study against these two classes.

### Security Evaluation

The security evaluation for the estate agent case study is straightforward, focusing on examining whether these security requirements are satisfied in the application. The evaluation is an attempt to give the users of sFDD method a flavour of how to accomplish this.

| | | C1 | C2 |
|---|---|---|---|
| Security Policy | Discretionary Access Control | NEW | CHG + ADD |
| | Object Reuse | NR | NEW |
| Accountability | Identification and Authentication | NEW | ADD |
| | Audit | NR | NEW |
| Assurance | Security Architecture | NEW | ADD |
| | System Integrity | NEW | NAR |
| | Security Tesing | NEW | ADD |
| Documentation | Security Features User's Guide | NEW | NAR |
| | Trusted Facility Manual | NEW | ADD |
| | Test Document | NEW | NAR |
| | Design Document | NEW | NAR |

**Table 7.9:** Requirements of Level C (derived from [8])

**C1 - Discretionary Security Protection** The first step is to check the application against class C1.

- **Discretionary Access Control**
  Access control is in the list of initial security requirements (i.e, some Web pages are only accessible to authenticated user). The information for these users are stored in a database table which only an administrator can modify. The Internet user cannot browse or edit any information on it. This access control mechanism is stronger than discretionary access control because only an administrator has rights to specify and control the access list.
- **Identification and Authentication**
  The application developed in the case study does implement a basic authentication mechanism (i.e, a pair of username and password). The user's information is stored in the database which can only be accessed by the administrator.
- **System Architecture**
  The application does have an architecture which is produced at the early stage of the development process. However the architecture is not as detailed as that required by class C1 because it requires that the objects (i.e, resources) should be defined, but the architecture of the application only defines the modules, rather than detailed assets.
- **System Integrity**
  In the case study, the problems encountered in development are key concerns. The periodical integrity check is done during the operation. So this requirement is not applicable.
- **Security Testing**
  Testing is not fully covered in sFDD. But by doing security inspection at the end of each iteration, the developer at least is sure that there are no obvious way for an unauthorised user to bypass the authentication or defeat the security protection mechanisms (e.g, database security mechanisms).
- **Documentation**
  Class C1 requires four types of documents: security features user's guide, trusted facility menu, test document, and design document. In the case study, the related software (or systems), such as Apache Tomcat, MySQL, are all documented very well. The documentation of security features used in the application can be found in these documents. In sFDD, there is no requirement of testing documentation. But there are several design documents which specify the system with detail.

In summary, the application satisfies most of these requirements. It fails the requirements of periodical integrity check because this is out of the scope for the method. About the documentation, there is no extra requirements for sFDD because sFDD also requires design documents, and others are provided by other organisations.

### C2 - Controlled Access Protection

- **Discretionary Access Control**
  In the application, there are only two kinds of users: Internet user who does not sign in the system so can only have access to the function of searching; and authenticated user who have already signed in the system and can have an access to all functions of the application. Therefore the access permission management is not complex, and only controlled by administrator.

- **Object Reuse**
  All objects during the operation of the application, including authentication tokens, database query, cannot be reused at all. The only object reused is the database connection for the sake of performance. But the access control of database is based on role based access control mechanism which is separated in the application (i.e, there are two roles created for the database connection from the application).

- **Identification and Authentication**
  The entire system (especially, the database) provides a function to enforce individual accountability and record all auditable actions by that individual.

- **Audit**
  The audit is the initial requirements for some iterations of the application. The system has implemented the function to record all auditable operations of the system.

- **System Architecture**
  Like the discussion in Class C1, the application has an architecture but is less detailed than what is required in the criteria.

- **System Integrity**
  Like the discussion in Class C1, the requirements are not applicable because the development of the application, rather than the whole life-cycle, is considered in the case study.

- **Security Testing**
  The testing of the application is incomplete because sFDD does not fully cover the stage of testing.

- **Documentation**
  Like the discussion in Class C1, the development does not provide as much as documents required by the security evaluation criteria. Some of required documents can be found in the documents provided by supported software, such as Apache Tomcat, MySQL, and Java language reference.

In summary, the application and its development satisfies a few requirements of class C2. The application partially satisfies the two "new" requirements of class C2: object reuse and audit. They are not fully satisfied because of the trade-off of performance and the nature of the application. Another obvious unsatisfied element is security testing. Security testing delivers important evidence of security assurance. Secure FDD address security inspection during the development stage, but more rigours testing is needed if evaluated against class C2.

#### Conclusion of security evaluation

The application developed in the case study has met most requirements of class C1; and a few requirements of class C2, for example, access control. Assessing the reasons of why some requirements are not satisfied, the following are addressed:

- The development method (i.e, sFDD) does not cover the full life-cycle of an information system. Security is a property crossing all stages: from initialisation to final disposal. Some requirements are not met because of the lack of life-cycle coverage, e.g, a system integrity check.
- Agile methods encourage a style of development with less up-front design and less documentation. But a security development process requires a comprehensive plan (i.e, detailed architecture). Some requirements are not met because of the lack of details in the architecture, e.g, system architecture. If it is important to meet these requirements, more detail should be added (at the cost of loss of agility).

It is very important to address in the security evaluation that the development of the case study does not take security evaluation criteria into account at the start of the project (i.e, security requirements). Its security requirements and security considerations come from industry guidance [65]. It has already been illustrated that all initial security requirements have been met in the application. The evaluation against Orange Book [8] is more formal and structured than the checklist approach of security requirement eliciting.

### 7.6.2 Agility Evaluation

Secure FDD needs to be evaluated against agile development principles because it is important to know whether secure FDD still has the agility benefits of plain FDD. Unlike security evaluation, an agility evaluation is purely about the development process. The development of the case study follows the steps of secure FDD. So far, there is not a widely accepted set of metrics to evaluate the agility of a software process. In studying the literature on agile software development [11, 131, 133, 144, 145], the common characteristics of an Agile method are that it is :

1. incremental (small software releases, with rapid cycles),
2. cooperative (customers and developers working constantly together with close communication),

3. straightforward (the method itself is easy to learn, to modify, and well documented),
4. adaptive (able to make last minute changes)

Except for the second bullet (cooperative), the case study has demonstrated that the sFDD process is an incremental, straightforward, and adaptive method. The thesis has not demonstrated that it is cooperative, but nor has it shown in any way that it is not. The results have already shown that sFDD also has other agile characteristics, namely responsiveness to change, and balancing flexibility and structure, as now discussed.

#### Balancing flexibility and structure

Building an overall architecture and planning in advance are the best ways to balance flexibility and structure. In the case study, the system is built within a pre-defined architecture so that iterations of development are planned. The case study demonstrates that the overall architecture provides such a flexibility for analysing and planning new development activities.

#### Conclusion of agility evaluation

The development of the case study demonstrated some advantages of the agility of sFDD. In another words, secure FDD is an incremental and adaptive way to build a system, and the method itself is straightforward.

#### Qualitative evaluation, not quantitative evaluation

Qualitative evaluation is a good starting point to evaluate a agile development process, but quantitative research can provide much stronger evidences. In the case study, the quantitative measurements (i.e, software metrics) are not considered in the research. Therefore, practical figures cannot easily be collected and analysed.

## 7.7 Conclusion

This chapter described the development process of the case study, a Web application for an estate agency. The development followed the steps of secure FDD. By focusing on the design and risk assessment during the development, this chapter gives a flavour of how to apply secure FDD.

Secure FDD has three distinct characteristics: 1) it has an explicit and straightforward process; 2) it builds overall models in advance; 3) and it considers the security considerations (e.g, security architecture, requirements) up-front. The case study showed how these characteristics benefit security development.

The case study is a non-critical system. With making use of technology that has been shown in the case study, such as WebScarab and Java security features, the application built in the case study was checked against the list based on the OWASP Guide [65]. But in order to give more discussion of the security, the application was evaluated more formally than the check-list approach, against TCSEC [8]. The attempt of this security evaluation suggests that the application which is built following sFDD process may achieve class C1 of the TCSEC [8]. But that may not be conclusive because security evaluation criteria are not considered at the beginning of the project. To evaluate the system security against these security criteria properly, a formal process which is defined in these criteria has to be followed. The agility evaluation shows that the development process remains incremental and adaptive.

Several things have to be concluded. Firstly, features are important elements in secure FDD. The size of a feature depends on many factors, including the complexity of the application, the experience of developers, etc. Secondly, during the development process, especially iterative development, risk assessment plays an important role to refine the security requirement and ensure the design and implementation. Last but not least, the overall architecture is also important in secure FDD, not only to specify the security requirements but also to plan the iterative development. It is the key to whole project's success.

Agile software development and security are both broad domains. To study the integration of them is a challenge. The next chapter will discuss the limitations of the research and the case study in order to give the readers a comprehensive overview of this research.

# 8 Limitations of the Case Study and Research

*The previous chapters described the approach taken to combine security and agile development and demonstrated the development process with a case study. In this chapter, the limitations of the case study and the research will be discussed in order to give the readers a comprehensive view of the research.*

## 8.1 Domain of Agile Software Development

The research is about the integration of agile and security software development. Security engineering is mature compared with agile development. But what really is new about agile development? There is a myth that agile development is novel because it is iterative. Actually, iterative development was considered important to development at least 30 years ago, for example as discussed by Boehm [164] in 1988. Then what is agile development about?

Firstly, agile development is more of a philosophy than a method. It is a set of principles that exists in the mind of developers and guides their development activities.

Secondly, agile development is not only about technical ideas, such as feature driven, test driven, user stories, refactoring and so on. None of these ideas would have created a revolution on their own. Agile development consists of a range of very practical development activities. These practices may have been existed separately for many years, but it is the agile approach that selects them and makes them work together in a new spirit.

Last but not least, agile software is about communication, people, and a team, about how to manage people, how to make decisions. These issues belonging to social engineering. The mental principle, techniques, and the social engineering are the three elements of agile software development; without any one of them, an approach will not be agile. Therefore, agile software development is a broad domain.

In Chapter 2, the review shows that security engineering is also a broad domain. It also consists of policy, techniques, and, again, issues from social engineering. Integrating two such broad domains, there is a large area of intersection, but also areas of incompatibility to overcome. The research presented in this thesis covers only part of the whole picture. There are a lot of issues that the research does not touch, for example the issues of social engineering are not considered to any extent in the research.

The following sections discuss several limitations which directly effect the outcomes of the research.

## 8.2 Limitations Related to Size of a System

The size of a system to be developed is one of critical factors in the selection of a development method. There is no direct evidence to show how agile methods are applied in various sizes of system, but Ambler [5] gives some figures about applying agile techniques on projects with various sizes of development team. An assumption is made that a large-scale systems development needs a larger development team. For example, the chart in Figure 8.1 shows how the proportion of successful agile development projects reduces as team size increases.

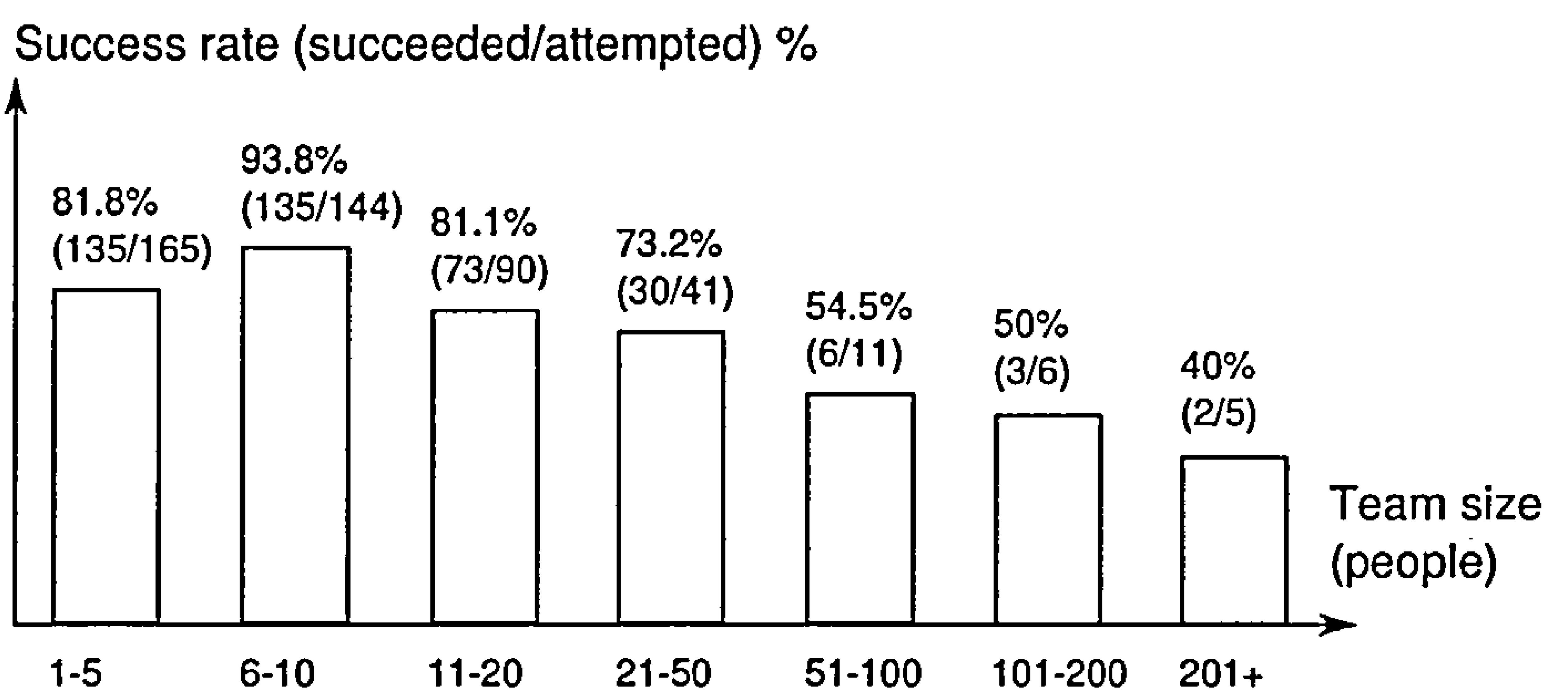



**Figure 8.1:** Success of Agile Practices on Various Projects (figures from [5])

From the chart, there are two interesting observations. First, the majority of organisations surveyed were applying agile techniques with a team of 10 or fewer people, with relatively few larger (>50 people) projects. Second, the organisations surveyed reported greater success with small teams (smaller projects) than larger ones. A partial explanation of this trend is that, the bigger the team, the harder it is to succeed, regardless of the development method. The effect may be exacerbated for agile developments – much of the agile literature focuses on collaboration and communication.

Agile software development, in general software engineering, is an art of practice. The figures in figure 8.1 suggests that organisations enthusiastically apply agile techniques on smaller projects. There are fewer agile attempts at larger projects. This reflects the reality of the agile development approach: its application to large-scale systems is an ongoing topic of research, and the findings from research and practice on larger-scale projects are far from conclusive.

The state of the art of agile research and practice limits the research, and especially the case study, presented in this thesis:

1. The object of the research is small applications and the case study is small too.
2. The changes introduced in the case study are small.

The rest of this section considers these limitations.

### 8.2.1 Small Case Study

The research focuses on a small project because, as an agile approach, the scope of use of plain FDD is limited. In relation to security, however, small systems should be treated the same way as big systems of equivalent criticality; small systems also need good defence in depth. Thus, the size of the case study system is not a problem for the security aspect of the research.

The case study is about developing a Web application of an online estate agency originally comprising three features. The case study illustrates the use of the sFDD approach in developing a Web application; it shows how sFDD exploits best practice, namely domain object modelling, security architecture modelling, developing by features, inspections, security risk analysis, regular integration, and visibility of progress and result.

Many problems of larger-scale systems development are not present in the case study, although some of them have been briefly (preliminarily) investigated. For example, a system with a complex security architecture is difficult to decompose into features.

As reviewed in Chapters 2 and 3, a secure system should have a layered architecture of security protections. Application security is a challenge if undertaken without any considerations of other layers such as the operating system. This means that decomposing a system in sFDD is not as easy as in plain FDD. Furthermore, sometimes an sFDD development iteration may involve some tasks other than software development, for example implementing network security mechanisms. This means that a single sFDD feature may consist of several sub-features, and need to be built in several iterations. For example, implementing a database security mechanism may involve many functional features, and thus require a number of iterations.

Another issue that is at least partly related to the size of the case study is the quality of security architecture modelling. Security architecture is important in sFDD because it outlines the dependencies of security mechanisms and decides development priorities.

sFDD includes an inspection of the architecture design. In a small project, such as the case study, the inspection is relatively straightforward because the security architecture and security requirements are not very complex. In a larger or more complicated project, systematic methods would be needed to verify the model against the requirements, and to guide the inspections. Such guidance is difficult to give, as the verification needs and inspection will take different forms in different projects. The issue of inspection approaches is not addressed in plain FDD, either.

### 8.2.2 Small Changes

All agile methods claim that they embrace change, but none of them give a clear description of their ability in this area, and none consider the effectiveness of change management – for example what kind and scale of changes can be handled within the agile processes.

Actually, the study of software change and change impact analysis started much earlier than the emergence of agile software development. Software changes can be large or small, simple or complex, important or trivial — all of which influence the effort needed to implement the changes. There are many papers about software changes and their consequences, such as [165, 166, 167, 168, 169, 170]. In such papers, changes are classified in various ways, such as the type of change activity, the kinds of items being changed, or the motivation for the change. For example, Weiss [171] reported in 1985 that the most frequent type of software change is the unplanned design modification; such modifications are usually made to optimise the program, to improve the services the program offers to its users, or to clarify and improve the maintainability of the software products.

The items subject to change may be requirements, designs or programs. For example, typical requirements changes include changes to accommodate customer needs or wants (demand for enhanced functionality), changes in operational environment, changes due to obsolete or clarified requirements, and changes due to lessons learned from software prototypes.

In relation to the motivation for change, Bonner [172] listed reasons for design changes as: requirements changes, design trade-offs and elaboration, interface changes, scope and visibility issues, performance, timing, sizing issues, and feedback from prototypes. Bonner also summarised typical reasons for program changes, including bug fixes, algorithm coding adjustments, arithmetic precision modifications, data structure modifications, initialisation modifications, control and sequence changes, and parameter changes.

There are two major technical approaches to software impact analysis: dependency analysis (such as [173, 174, 175]) and traceability analysis (such as [176, 177]). These complementary areas approach impact analysis from quite different perspectives and have their respective advantages in enhancing the potential for identifying software change impact.

In the case study, there was no classification and analysis of the impact of the changes introduced – no requirement of software impact analysis is included in either sFDD and plain FDD. In general, software impact analysis is not mentioned in any agile literature. This is a failing of agile approaches, and could be remedied by consulting the wide ranging research on software change.

The change introduced in the case study is small at the highest level, but was shown to have significant knock-on effects. The change in any system from local to distributed service provision necessarily introduces significant changes in security functionality. However the impact is fairly easily assessed. The change clearly leads to the introduction of encryption and the services needed to support it appropriately.

A framework of change analysis for sFDD is shown in Figure 8.2.

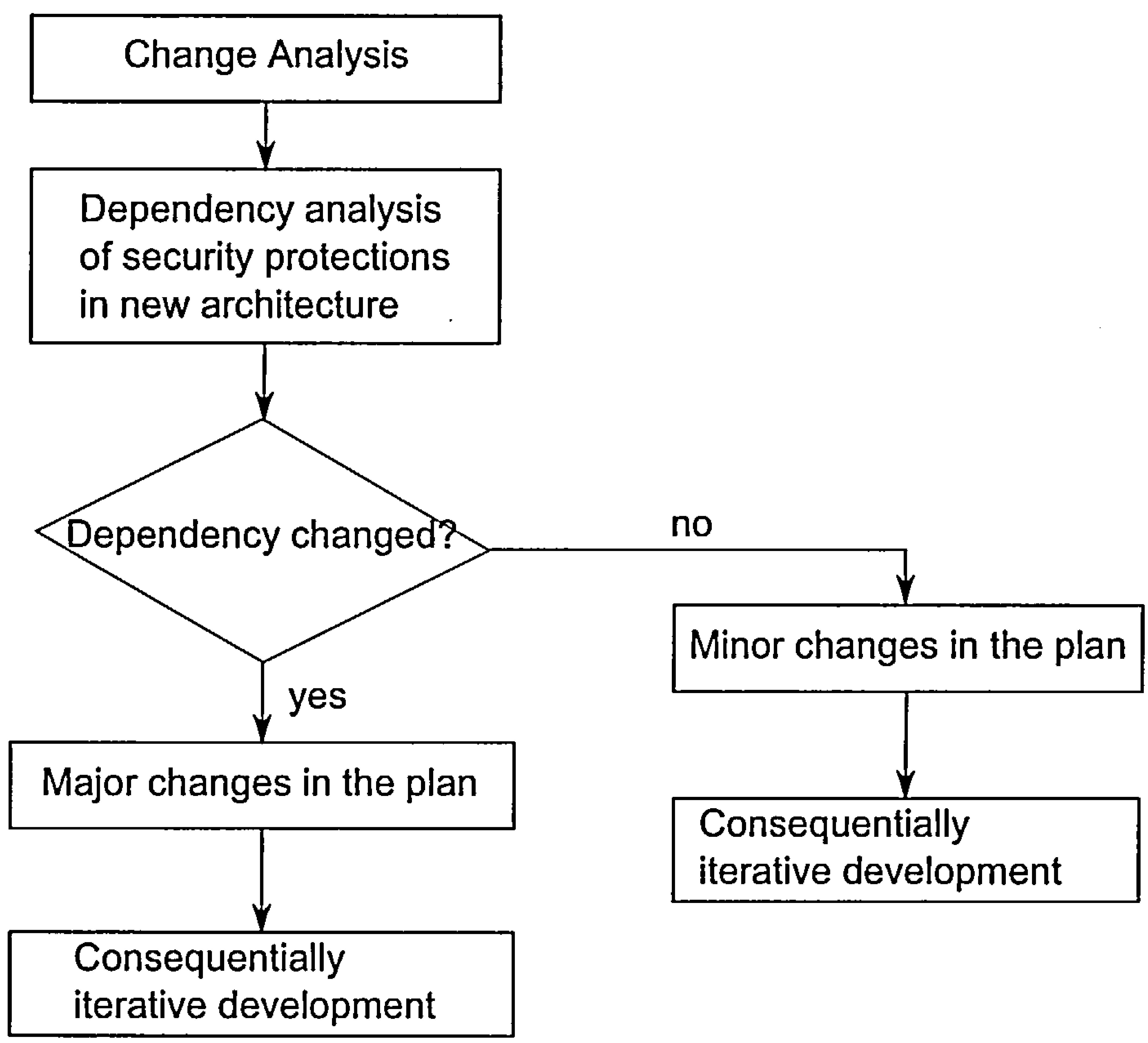



**Figure 8.2:** Framework of Change Analysis

The analysis process when there is a new requirement is straightforward: the developer considers a new architecture; then analyses the dependency of security protections in the new architecture; then compares the new and old architecture. If the dependency relationships are not changed, there will be a minor change in the plan and following development. But if there are significant changes in the dependency relationships, there will be a major change in the plan and the development process will be revised.

## 8.3 Limitations Related to Team Work

Developing a software system is usually team work. The plain FDD [3] defines 15 roles in the development team, including project manager, chief architect, development manager, chief programmer, class owner, domain expert, domain manager, release manager, language guru, code programmer, tool builder, system administrator, tester, deployer, technical writer. In sFDD, roles are added for security aspects, such as security expert, security tester, and security manager. Therefore, there are at least 18 roles in the development team, excluding the on-site customer.

Among the sFDD roles, more than one can be taken by each team member. For example, the chief programmer can be a class owner and language guru. However some of the roles really need to be taken by different people. For example, the tester and code programmer should be different people for the reason of the quality of development and testing. Ideally, security roles should not be shared, to maximise the oversight of security in the team.

In this research, the author was the only person working on the case study, and thus had to take all the team roles. This has a significant potential impact on the quality of the application developed in the case study, and weakens the evaluation of the case study. To attempt to mitigate the impact of this limitation, a trial security evaluation of the application against TCSEC is given in Chapter 7. The aim of such a retrospective evaluation (TCSEC guidelines were was not considered at the beginning of the case-study development) is to use a set of accepted criteria to evaluate the quality of the software product, and to identify missing requirements of sFDD.

## 8.4 Challenges From Other Layers

In Chapter 2 it is shown that security is an emergent properties in all layers in the system architecture. Chapter 4 identified that plain FDD encourages developers to consider the architecture of the system in advance. In fact, plain FDD suggests partitioning an application into layers that include (some of) a User Interface layer (UI), a Problem Domain (PD) ("business logic" layer), a System Interface (SI) layer and a Data Management (DM) layer.

The research and the case study focus on the security development of the business logic, which is in the PD layer. To build a secure system, it is not enough to consider only the security of a particular layer. Issues from other layers impact on the security of the PD layer. For example, it is well-known that flaws or bad designs in the UI can cause security problems for application logic.

During the research and the development of case study, the challenges from other layers, such as UI, DM, are considered. They have not been emphasised in writing up the case study because the focus of the research and the case study is the security of application, and this is complicated enough.

## 8.5 Lack of Coverage of Software Development Life Cycle

One identified shortcoming of plain FDD is that plain FDD focus on design and implementation; other approaches are needed to cover the full span of the SDLC. Figure 2.5 in Chapter 2 relates common security artefacts to a SDLC. In sFDD, the goal is to integrate one of them (architectural risk analysis, from the design part of the SDLC) into plain FDD, in order to balance the security of the resultant software system and the agility of the development process. This does not mean that other security artefacts could not be integrated into an agile process, although artefacts such as security requirements and penetration testing are outside of the scope of plain FDD.

Building a secure system is a project of system engineering. Only applying the sFDD approach is not enough; it has to be used in the context of other supporting approaches to cover the entire SDLC.

## 8.6 Conclusion

This chapter discussed some of the limitations of the research and the case study. Agile software development and security are both wide research areas; they both raise issues for information technology and social engineering. The research and the case study only consider the information technology side of their integration. There are further limitations, due to the broad domain of agile and security development; and the nature of the research and limitation of agile process itself.

The next chapter presents an overall evaluation of the research presented in this thesis, and provides indications for some interesting directions of future work.

# Part III

# Conclusion and Future Work

# 9 Evaluation of Research

*The major contribution of this thesis is a development process that integrates security and agile software development. The evaluation of it focuses on the quality of this new proposed process.*

## 9.1 Framework of Research

As explained in Chapter 1, the research presented in this thesis was carried out following three steps:

1. Find a suitable Agile method and set of security practices,
2. Study how to integrate them,
3. Demonstrate they work well.

The hypothesis of this thesis is that:

**It is possible to develop an acceptably secure Web application by adopting sFDD which is an agile method.**

The previous chapters have already explained the process of the research and given an example of how a Web application is developed using secure FDD. The major contribution of this research is secure FDD, which is an integration of security and agile development. The proposal for sFDD is based on a thorough analysis of the security development and agile development. Secure FDD has three distinct characteristics: 1) it has an explicit and straightforward process; 2) it builds overall models in advance; 3) it considers the security concerns (e.g, security architecture, requirements) up front. The case study supports the hypothesis: a Web application has been built using sFDD, and has been shown to be secure enough (i.e, it satisfies all security requirements).

The key part of evaluation concerns an assessment of the quality of sFDD process. It will come from two angles: agility and quality of the process.

## 9.2 Discussion of Agility

The Manifesto for Agile Software Development [25] defines the fundamental principles that an Agile method should support. It is the only accepted criteria to answer the questions of what is an Agile method and what is not. The principles are defined to satisfy value priorities. The values of agile software development are:

- **Individuals and interactions,** against *heavyweight processes or tools*
- **Working software,** against *comprehensive documentation*
- **Customer collaboration,** against *closed-house development*
- **Responding to change,** against *strictly following a plan*

The Manifesto for Agile Software Development says that there is more value in the items on the left (in **bold**); the author has added comparisons on the right (in *italic*).

The following paragraphs assesses that the secure FDD method against the above four values. This aims to show that sFDD embodies the values in the Manifesto, and is thus an agile process.

- **Individuals and interactions,** against *heavyweight processes or tools*

  The development of a software system involves team work. Although the roles involved in a project of secure FDD were not demonstrated in detail, the roles in a plain FDD project are applicable to secure FDD. Cooperativeness and good communication are also encouraged in the development process. Secure FDD emphasises software architecture, but the architecture also focuses on the shape, not the content of the system. Therefore, secure FDD is not a heavyweight process. A tool, like FDDPMA, can be used in the project, but secure FDD does not rely on automated tools. Secure FDD includes iterations and feedback in the process (see Figure 6.1).

- **Working software,** against *comprehensive documentation*

  Secure FDD does have up-front designing and modelling, but it also aims to deliver working releases quickly, normally every one to two weeks. A reasonable amount of documentation is required in the process. Some documents, such as overall models, architecture, timetables and risk assessment reports, are necessary to keep the project monitored and running smoothly, and for satisfying security requirements.

- **Customer collaboration,** against *closed-house development*

  The development of a secure FDD project is not a closed-house development process. It encourages collaborations and interactions with customers, for example security decision making and risk assessment judging. Again the importance of customer involvement was not demonstrated in detail, but the basis of secure FDD (i.e, plain FDD) does directly address this issue, and nothing introduced in sFDD changes this.

- **Responding to change,** against *strictly following a plan*

  Secure FDD has up-front planning. However, the Agile Manifesto [25] does not forbid an initial plan; it suggests that responding to change is more valuable than strictly following a plan. Secure FDD considers the plan a required part of any project. However, strictly following the plan without being able to adapt and respond to change is a pattern for failure, and is certainly not part of sFDD. As a result, secure FDD accepts changes of the plan. Any changes of the plan are treated as new features to be built.

In short, the secure FDD method embodies all the values that an Agile method should, according to the Agile Manifesto [25]. In conclusion, secure FDD can be categorised as an Agile method.

## 9.3 Overall Quality of sFDD Process

[42] introduces seven requirements for a software development process, which are **effectiveness, maintainability, predictability, repeatability, quality, improvement,** and **tracking.** These seven factors are used to evaluate the overall quality of the sFDD process.

- **Effectiveness** Effectiveness for a process means that the process helps to produce the right product. The requirements of effectiveness for sFDD is 1) the sFDD process helps to develop a working application (i.e, the application satisfies all the functional requirements); and 2) the sFDD process helps to develop a secure application (i.e, the application satisfies all security requirements). In the case study, a Web application is built using sFDD process. The case study has already demonstrated that the product of the sFDD process is a working and secure application. Therefore, the sFDD process is effective.
- **Predictability** Predictability for a process means the business value of the process is predictable. A development plan is produced at the end of first stage of sFDD process. In the plan, the deliverable of each development iteration has been described clearly. So the product of each iteration is predictable.
- **Maintainability** Maintainability refers to how a process adapts to post-deployment changes. The case study concretely demonstrates how sFDD process adapts to a post-development change. However, the sFDD process only covers a short part of the life-span of an information system. As such, maintainability is not easily assessed for sFDD (nor FDD either).

- **Repeatability** Repeatability depends on the scope of sFDD adoption. In practice, sFDD is suitable for a project which is going to deliver small and non-high-integrity application like the application developed in the case study. In this context, and because of the precisely determined steps and tasks of sFDD, I argue that sFDD is repeatable.
- **Quality** Secure FDD does not fully cover the stage of software testing. But I claim it improves security by considering it in advance. The case study has demonstrated that it is feasible to integrate several security artefacts with an agile software development process. Secure FDD is thus a systematic approach to agile software development which emphasises the requirements of security. As such, I consider it a high quality process in this domain.
- **Improvement** A good software development approach should not be a rigid process. It should have possibility for improvement, especially when advanced principles or techniques emerge. Secure FDD has several steps of development which define the contents (e.g, tasks) of the development work), rather than concrete techniques to be adopted. For example, sFDD does not define which technique should be used in the step of **security risk assessment**. The developers can decide on any particular technique. This leaves space for future improvement of the process.
- **Tracking** A good process is a traceable process; it is an essential requirement of project management. The research of this thesis did not produce any tool for secure FDD to support project management. Instead it customised a tool for plain FDD. In sFDD, there are several milestones for project management. By using a tool, the case study demonstrates that the process of sFDD is indeed traceable.

Overall sFDD is a quality software process because it meets many of key requirements. But this does not means sFDD is perfect. There are several drawbacks of sFDD.

- The software development life-cycle is not fully covered. It has been suggested that security development should cover all phases of SDLC [18]. Plain FDD does not cover the entire SDLC. Secure FDD is an integration based on plain FDD so it cannot cover all phases of SDLC. But within the stages of SDLC that plain FDD covers, sFDD introduces essential security activities to emphasise the importance of security, while attempting to not lose the values of an agile development method.
- No quantitative measurement. There is no quantitative measurement of how much effort should be put in the front parts of the development process. Security development (especially the approach of security evaluation criteria) needs comprehensive up-front analysis, but too much up-front analysis makes the process less agile. Secure FDD thus aims to be in the middle of security development for critical systems (e.g, systems have to be evaluated against higher levels of security evaluation criteria) and agile software development (e.g, extreme programming), but there is yet no quantitative measurement to clarify how much up-front analysis is needed.

# 10 Conclusions and Future Work

*This chapter concludes how the work proposed in this thesis supports the thesis proposition, summarises the overall conclusions of the research and addresses some directions for further research.*

## 10.1 Overall Conclusions

Overall, the objective of the research was to analyse the feasibility of adapting an Agile software development method in order to improve Web application security. The work presented here provides substantiation for this objective.

The novel contributions of this research are:

- Analysis of conflicts between established risk assessment methods and Agile methodologies;
- Development of an Agile method for the purpose of building secure (Web) applications; more precisely an integration of standard FDD which explicitly considers the security of the software system while maintaining the agility of the development process producing the software system; and
- Demonstration of the use of this adapted FDD method, and an overall evaluation of the agility and quality of the method.

In the research, risk assessment is integrated into standard FDD sub-processes. It is demonstrated that secure FDD is an Agile method because it embodies all the values of an Agile method. By performing a risk assessment process before and during the development iterations, the method can help to improve the security of the target software system. There are therefore good reasons to believe that integrating security development with an FDD process is a *"right first"* step towards agile security engineering.

## 10.2 Future Work

There are several potential courses for furthering or complementing the research reported in this dissertation, some of which are listed below:

- Automating the risk assessment process and achieving security to standard levels. At the moment, the risk assessment process has to be done manually so far in the method. Although the risk assessment is a model-driven process, it would be helpful if there was a tool to help. In addition, security has been standardised in several levels in most international security standards. Fully addressing these standards in sFDD would be useful.
- Targeting other Agile methodologies (e.g, XP). FDD was the focus of this research because of its characteristics, particular up-front architecture. There are many other established Agile methods, and the development method is selected based on the nature of the project, such as the size of project, the familiarity of developers etc. It would be useful to study the security artefacts in the development process of other Agile methods, such as XP. Some preliminary work on this in the domain of safety engineering has been carried out [178].
- Applying the methodology to case studies of larger scope. The systems used for the verification and validation of the methodology were intentionally selected to be small in scope, as the focus has mostly been on the practicability of the approach. In order to test the scalability of the methodology, larger systems should be considered for testing.

# Part IV

# Appendix

# A Description of Use Case

| Use Case | Searching properties |
|---|---|
| **Brief description** | This use case allows an Internet user to browse properties. |
| **Actor** | Internet User |
| **Pre-condition** | User open an Internet browser, and point to the Agency's home page. This page displays several items that allow the user to select the criteria of query. |
| **Main Flow** | The use case begins when the User browses the agency's Web site. The system waits the user to select some options from the list, including post code, how many bedrooms, the range of the property price. Then the user submits the criteria. The application will varify the parameters and prepare the execution of SQL query. When the query is executed without any exceptions, the application send the results back in a list. |
| **Exceptions** | There are two kinds of possible exceptions: HTTP exceptions and database exceptions. The major HTTP exception will be the exception if the contents of HTML page are not correct. That may be because the results of database operations are not parsed correctly. The database exceptions are occured when there is something wrong in SQL sentenses or database. |
| **Security Requirements** | The inputs of Internet users are validated. The Internet users can only have an access of some information of properties, including the description, address, asking price, and the number of bedrooms. |

**Table A.1:** Use case: Searching properties

| Use Case | User Authentication |
|---|---|
| **Brief description** | This use case requests user's username and password, then authenticates user. |
| **Actor** | Internet User |
| **Pre-condition** | User's browser points to *SignIn* page. |
| **Main Flow** | This use case begins when a user types in his username and password, then submits these to the server. The application checks the username and password: if they are matched with the record in the database, the application sends back a authentication token to user; and the application redirects the user's request to previous page. |
| **Exceptions** | There are two kinds of exceptions:<br>• if the username/password is not matched, an error message is returned to the user.<br>• if there is any exception thrown during the database operation, the application will catch it and display the cover-up page. |
| **Security Requirements** | The inputs of Internet users (i.e, username and password) are examined that there is no special characters in the text field. The authentication token is encrypted and well managed in sessions. The operation is audited. |

**Table A.2:** Use case: User Authentication

| Use Case | Submitting Offer |
|---|---|
| **Brief description** | This use case allows user to submit an offer to the server; then the offer will be recorded in the database. |
| **Actor** | Internet User |
| **Pre-condition** | The user is identified and authorised to have an access to this use case. |
| **Main Flow** | The application receives the request of an offer; then the application connects the database and inserts a record in the database. |
| **Exceptions** | The major exceptions are from database operations, including database connection, database insertion. |
| **Security Requirements** | The inputs of users (i.e, the offer price) is type checked; and the integrity of the offer is checked (in a reasonable range). The operation is audited. |

**Table A.3:** *Use case: Send message*

# Bibliography


[1] R. Baskerville, "Information systems security design methods: implications for information systems development," *ACM Computing Surveys*, vol. 25, no. 4, pp. 375–414, 1993.

[2] J. Meier, A. Mackman, S. Vasireddy, M. Dunner, R. Escamilla, and A. Murukan, "Improve web application security: Threats and countermeasures," tech. rep., Micorsoft, 2003.

[3] S. R. Palmer and J. M. Felsing, *A practical guide to feature-driven development.* Prentice Hall, 2002.

[4] S. Khramtchenko, "A project management application for feature driven development," Master's thesis, Harvard Univeristy, June 2005.

[5] S. W. Ambler, "Survey says ... agile has crossed the chasm." http://www.ddj.com/artichitect/200001986, July 2007.

[6] L. Barnett, "Agile survey results: Solid experience and real results." http://www.agilejournal.com/articles/from-the-editor/agile-survey-results%3a-solid-experience-and-real-results.html, September 2006.

[7] G. Booch, A. Cockburn, and A. Pyster, *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, 2004.

[8] "Trusted computer systems evaluation criteria (orange book)." DoD 5200.28-STD, Washington D.C., Department of Defence, December 1985.

[9] "Common criteria for information technology security evaluation, version 2.1." ISO/IEC 15408, 1999.

[10] "Information technology security evaluation criteria." Department of Trade and Industry, London, June 1991.

[11] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," Tech. Rep. ESPOO 2002, VTT Publication 478, Technical Research Centre of Finland, 2002.

[12] X. Ge, R. F. Paige, F. A. Polack, H. Chivers, and P. J. Brooke, "Agile development of secure web applications," in *Proceedings of the 6th international conference on Web engineering ICWE '06*, (New York, NY, USA), pp. 305–312, ACM, 2006.

[13] X. Ge, H. Chivers, F. Polack, and R. Paige, "Adapting security risk analysis to the design of database-centric web-based information system," in *18th International Conference of Software and System Engineering and Their Applications (ICSSEA)*, (CNAM, Paris), November 2005.

[14] X. Ge, F. Polack, and R. Laleau, "Secure databases: an analysis of Clark-Wilson model in a database environment," in *Advanced Information Systems Engineering. The 16th International Conference on Advanced Information Systems Engineering (CAiSE04)* (A. Persson and J. Stirna, eds.), pp. 234–247, LNCS 3084, Springer-Verlag, 2004.

[15] X. Ge, F. Polack, and R. Laleau, "Secure database development and the Clark-Wilson security model," in *Atelier SSI'04 Securite des Systemes d'Information, INFORSID 2004*, (Biarritz, France.), 2004.

[16] A. Apvrille and M. Pourzandi, "Secure software development by example," *IEEE Security & Privacy*, vol. 3, pp. 10–17, July/August 2005.

[17] "Information resources management." The Department of Justice USA, Systems Development Life Cycle Guidance Document http://www.usdoj.gov/jmd/irm/lifecycle/table.htm, January 2003.

[18] T. Grance, J. Hash, and M. Stevens, "Security considerations in the information system development life cycle," tech. rep., National Institute of Standards and Technology (NIST), Special Publication 800-64, October 2003. (revision 1 released June 2004).

[19] R. J. Anderson, *Security Engineering: a guide to building dependable distributed systems*. Wiley, 2001.

[20] G. B. W. Eric A. Fisch, *Secure Computers and Networks: Analysis, Design, and Implementation*. CRC Press, 2000.

[21] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing*. Prentice Hall, 3rd ed., 2003.

[22] M. T. Siponen, *An Analysis of the Recent IS Security Development Approaches: Descriptive and Prescriptive Implications*, ch. 8, pp. 101–124. Idea Group Publishing, 2001.

[23] S. Ramadorai, "Towards the internet era," *The Economic Times*, November 3 2000.

[24] S. Goldman, R. Nagel, , and K. Preiss, *Agile Competitors and Virtual Organizations*. Wiley, October 1994.

[25] "Agile manifesto." http://agilemanifesto.org, May 2007.

[26] "RSA conference 2006." http://2006.rsaconference.com/us/, February 2006.

[27] H. Chivers, R. F. Paige, and X. Ge, "Agile security using an incremental security architecture," in *Proceeding of the Sixth International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2005)*, Spring-Verlag LNCS 3556, (Sheffield, UK), pp. 57–65, 2005.

[28] R. Baskerville, "Agile security for information warefare: a call for research," in *Proceedings of the European Conference on Information System, ECIS2004*, (Turku, Finland), June 2004.

[29] R. Paige, J. Cakic, X. Ge, and H. Chivers, "Towards agile reengineering of dependable grid applications," in *Proceeding of 17th International Conference of*

*Software and System Engineering and Their Applications (ICSSEA)*, (CNAM, Paris), November 2004.

[30] M. Siponen, R. Baskerville, and T. Kuivalainen, "Integrating security into agile development methods," in *38th Hawaii Internetional Conference on System Sciences*, 2005.

[31] B. W. Boehm and T. DeMarco, "Software risk management," *IEEE Software*, vol. 14, pp. 17–19, May-June 1997.

[32] G. Stoneburner, A. Goguen, and A. Feringa, "Risk management guide for information technology systems," tech. rep., National Institute of Standards and Technology (NIST), Special Publication 800-30, July 2002.

[33] R. L. Murphy, C. J. Alberts, R. C. Williams, R. P. Higuera, A. J. Dorofee, and J. A. Walker, *Continuous Risk Management Guidebook.* Software Engineering Institute (SEI), 1996.

[34] R. C. Williams, J. A. Walker, and A. J. Dorofee, "Putting risk management into practice," *IEEE Software*, vol. 14, pp. 75–82, May/June 1997.

[35] A. Gemmer, "Risk management: Moving beyond process," *IEEE Computer*, vol. 30, pp. 33–43, May 1997.

[36] E. H. Conrow and P. S. Shishido, "Implementing risk management on software intensive projects," *IEEE Software*, vol. 14, pp. 83–89, May/June 1997.

[37] "An introduction to computer security: The NIST handbook." http://csrc.nist.gov/publications/nistpubs/800-12/handbook.pdf, October 1995.

[38] "Information security mangement part 2: Specification for information security management systems," tech. rep., British Standards Institution BS 7799-2:1999, 1999.

[39] "Operationally critical threat, asset, and vulnerability evaluation (OCTAVE)," tech. rep., Software Engineering Institute,CERT Coordination Centre, http://www.cert.org/octave/, 2003.

[40] "Information security mangement part 1: Code of practice for information security management systems," tech. rep., British Standards Institution BS 7799-1:1999, 1999.

[41] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in software quality," tech. rep., National Technical Information Service NTIS ADA049-014(Volume I),015(Volume II),055(Volume III), 1977.

[42] S. Tyrrell, "The many dimensions of the software process." http://www.acm.org/crossroads/xrds6-4/software.html#MCCALL, 2000.

[43] J. Eckstein, *Agile Software Development in the Large - Diving into the Deap.* Dorset House Publishing, 2004.

[44] T. Kahkonen, "Agile methods for large organizations - building communities of practice," in *Proceedings of the Agile Development Conference (ADC'04)*, (Washington, DC, USA), pp. 2–11, IEEE Computer Society, 2004.

[45] D. J. Reifer, F. Maurer, and H. Erdogmus, "Scaling agile methods," *IEEE Software*, vol. 20, pp. 12–14, July/August 2003.

[46] G. Stoneburner, "Underlying technical models for information technology security," tech. rep., National Institute of Standards and Technology (NIST), Special Publication 800-33, December 2001.

[47] J. Viega and G. McGraw, *Building Secure Software.* Addison-Wesley, 2002.

[48] S. R. W. Terry Mayfield, J.Eric Roskos and J. M. Boone, "Integrity in automated information system." National Computer Security Center NCSC C TECHNICAL REPORT 79-91 Library No. S-237.254, September 1991.

[49] C. P. Pfleeger, *Security in Computing.* Prentice Hall, 2nd ed., 1997.

[50] D. Woodward, "Smart security," *The British Journal of Administrative Management*, vol. 18, pp. 22–23, 2000.

[51] R. Malan and D. Bredemeyer, "Defining non-functional requirements," tech. rep., Bredemeyer Consulting, Whitepaper 8/3/01, 2001.

[52] D. Bennett, *Designing Hard Software: the Essential Tasks.* Prentice-Hall, 1997.

[53] H. Chivers, "Security and systems engineering," Tech. Rep. YCS378, University of York, June 2004.

[54] G. McGraw, *Software Security: Building Security In.* Addison-Wesley Software Security Series, Addison Wesley Professional., January 2006.

[55] "Common criteria for information technology security evaluation, version 2.5." ISO/IEC 18405, 2005.

[56] R. Kissel, "NIST glossary of key information security terms." http://csrc.nist.gov/publications/nistir/NISTIR-7298_Glossary_Key_Infor_Security_Terms.pdf, April 2006.

[57] J. Meier, A. Mackman, S. Vasireddy, M. Dunner, R. Escamilla, and A. Murukan, "Security engineering explained," tech. rep., Micorsoft, 2005.

[58] B. Schenier, *Beyond Fear: Thinking Sensibly About Security in an Uncertain World.* Copernicus Books, 2003.

[59] M. Howard and D. LeBlanc, *Writing Secure Code.* Microsoft Press, December 2002.

[60] "CRAMM." http://www.cramm.com/.

[61] M. Graff and K. van Wyk, *Secure Coding, Principals, and Practices.* O'Reilly, 2002.

[62] "Automated security self-evaluation tool." The National Institute on Standards and Technology (NIST), http://csrc.nist.gov/asset/, 12 2004.

[63] "Control objectives for information and related technology (COBIT) 4.1." Information Systems Audit and Control Association (ISACA), http://www.isaca.org/cobit/, April 2007.

[64] R. Shirey, "Internet security glossary." http://www.faqs.org/rfcs/rfc2828.html, May 2000.

[65] "A guide to building secure web applications and web services." The Open Web Application Security Project (OWASP) http://prdownloads.sourceforge.net/owasp/OWASPGuide2.0.1.pdf?download, July 2005.

[66] J. D. Moffett and B. A. Nuseibeth, "A framework for security requirements engineering," Tech. Rep. YCS2003-368, Computer Science Department, University of York, Augest 2003.

[67] A. Dardenne, A. van Lamsweerde, and S.Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, pp. 3–50, 1993.

[68] R. Darimont, E. Delor, PhilippeMassonet, and A. van Lamsweerde, "GRAIL/KAOS: a requirements engineering environment," in *ICSE19*, pp. 612–613, ACMP, May 1997.

[69] E. Amoroso, *Fundamentals of Computer Security Technology*. Prentice & Hall, 1994.

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

[71] P. P. Chen, "The entity-relationship model - toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, pp. 9–36, March 1976.

[72] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. ADDISON-WESLEY, 1998.

[73] P. Stevens, *Using UML software engineering with objects and components*. Addison-Wesley, updated edition ed., 2000.

[74] J. Jürjens, "UMLsec: Extending UML for secure systems development," in *Proceeding of 5th International Conference of Unified Modeling Language: Model Engineering, Languages, Concepts, and Tools. UML 2002* (J.-M. Jézéquel, H. Hussmann, and S. Cook, eds.), vol. 2460 of *LNCS*, (Dresden, Germany), pp. 412–425, Springer Verlag, September/October 2002.

[75] J. Jürjens, "Towards development of secure systems using UML," in *Proceeding of 4th International Conference of Fundamental Approaches to Software Engineering FASE 2001 (Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001)* (H. Hußmann, ed.), vol. 2029 of *Lecture Notes in Computer Science*, (Genova, Italy), Springer Verlag, April 2001.

[76] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security," in *Proceeding of 5th International Conference of Unified Modeling Language: Model Engineering, Languages, Concepts, and Tools. UML 2002* (J.-M. Jézéquel, H. Hussmann, and S. Cook, eds.), vol. 2460 of *LNCS*, (Dresden, Germany), pp. 426–441, Springer Verlag, September/October 2002.

[77] D. A. Basin, J. Doser, and T. Lodderstedt, "Model driven security for process-oriented systems.," in *8th ACM Symposium on Access Control Models and Technologies SACMAT 2003*, (Villa Gallia, Como, Italy), pp. 100–109, June 2003.

[78] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Proceedings of the 15th Annual Computer Security Applications*

*Conference (ACSAC '99)*, (Phoenix, Arizona), p. 55, IEEE Computer Society, 1999.

[79] C. E. Landwehr, "Formal models for computer security," *ACM Comput. Surv.*, vol. 13, pp. 247–278, September 1981.

[80] S. Stepney and S. P. Lord, "Formal specification of an access control system," *Software—Practice and Experience*, vol. 17, no. 9, pp. 575–593, 1987.

[81] M. H. Cheheyl, M. Gasser, G. A. Huff, and J. K. Millen, "Verifying security," *ACM Computing Surveys*, vol. 13, no. 3, pp. 247–278, 1981.

[82] J. M. Wing, "A symbiotic relationship between formal methods and security," in *Workshops on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solution*, 1998.

[83] M. Tracy, W. Jansen, and M. McLamon, "Guidelines on securing public web servers," tech. rep., National Institute of Standards and Technology (NIST), Special Publication 800-44, September 2002.

[84] "Secure programming guidelines." http://archive.ncsa.uinc.edu/General/Grid/ACES/security/programming/, January 2003.

[85] "Security code guidelines." http://java.sun.com/security/seccodeguide.html, February 2000.

[86] G. Booch, "The architecture of web applications." http://www.ibm.com/developerworks/ibm/library/it-booch_web/, June 2001.

[87] G. Costagliola, F. Ferrucci, and R. Francese, "Web engineering: Models and methodologies for the design of hypermedia applications," *Handbook of Software Engineering and Knowledge Engineering*, vol. 2, pp. 181 – 199, 2002.

[88] J. Conallen, *Building Web Applications with UML*. Addison-Wesley Professional, 2nd ed., 2002.

[89] A. Ginige and S. Murugesan, "Web engineering: An introduction," *IEEE Multimedia, Special Issue on Web Engineering*, vol. 8, no. 1, pp. 14–18, 2001.

[90] M. Bieber and T. Isakowitz, "Design hypermedia applications," *Communications of the ACM*, vol. 38, no. 8, pp. 26–29, 1995.

[91] E.Marcos, P. Cáceres, B. Vela, and J. Cavero, "MIDAS/DB: a methodological framework for web database design," in *Proceedings of International Workshop on Data Semantics in Web Information Systems (DASWIS2001)*, vol. 2465 of *LNCS*, (Yokohama, Japan), Springer, November 2001.

[92] P. Cáceres, E. Marcos, and B. Vela, "A MDA-based approach for web information system development," in *Proceedings of workshop in Software Model Engineering (WiSME) in UML'2003*, (San Francisco, USA), http://www.metamodel.com/wisme-2003, October 2003.

[93] G. R. Lifia, H. Schmid, and F. Lyardet, "Engineering business processes in web applications: Modeling and navigation issues.," in *Proceedings of 3rd International Workshop on Web-Oriented Software Technologies IWWOST'03*, July 2003.

[94] F. Garzotto, P. Paolini, and D. Schwabe, "HDM — model-based approach to hypertext application design," *ACM Transactions of Information System.*, vol. 11, no. 1, pp. 1–26, 1993.

[95] J. Gómez, C. Cachero, and O. Pastor, "Conceptual modeling of device-independent web applications.," *IEEE MultiMedia*, vol. 8, no. 2, pp. 26–39, 2001.

[96] J. Gómez and C. Cachero, "OO-H method: extending UML to model web interfaces," *Information modeling for internet applications*, pp. 144–173, 2003.

[97] R. Hennicker and N. Koch, "A UML-based methodology for hypermedia design," in *Proceedings of 3rd International Conference on the Unified Modeling Language (UML2000)*, vol. 1939 of *LNCS*, (York, UK), pp. 410–424, Springer, October 2000.

[98] N. Koch and A. Kraus, "Towards a common metamodell for the development of web appliactions," in *Proceedings of International Conference of Web Engineering (ICWE03)*, vol. 2722 of *LNCS*, (Oviedo, Spain), pp. 497–506, Springer, July 2003.

[99] L. Baresi, F. Garzotto, and P. Paolini, "Extending UML for modeling web applications," in *Proceedings of 34th Annual Hawaii International Conference on System Sciences ( HICSS-34)*, vol. 3, (Maui, Hawaii, USA), January 2001.

[100] S. Ceri, P. Fraternali, and A. Bongio, "Web modeling language (WebML): a modeling language for designing web sites," in *Proceeding of 9th International World Wide Web Conference*, vol. 33, pp. 137–157, Computer Networks, 2000.

[101] F. Ricca and P. Tonella, "Understanding and restructuring web sites with reweb," *IEEE MultiMedia*, vol. 8, no. 2, pp. 40–51, 2001.

[102] G. A. D. Lucca, A. R. Fasolino, and P. Tramontana, "Reverse engineering web applications: the ware approach," *Software Maintenance and Evolution: Research and Practice*, vol. 16, no. 1-2, pp. 71–101, 2004.

[103] N. Anquetil and T. C. Lethbridge, "Experiments with clustering as a software remodularization method," in *Proceeding of 6th Working Conference on Reverse Engineering*, (Atlanta, Georgia, USA), pp. 235–255, IEEE Computer Society, October 1999.

[104] C. Boldyreff and R. Kewish, "Reverse engineering to achieve maintainable WWW sites," in *Proceedings of 8th Working Conference on Reverse Engineering (WCRE'01)*, (Washington, DC, USA), pp. 249–257, IEEE Computer Society, 2001.

[105] H. Q. Nguyen, *Testing Applications on the Web: Test Planning for Internet-Based Systems*. Wiley, October 2000.

[106] F. Ricca and P. Tonella, "Analysis and testing of web applications," in *Proceedings of 23rd International Conference on Software Engineering (ICSE '01)*, (Washington, DC, USA), pp. 25–34, IEEE Computer Society, 2001.

[107] S. McClure, S. Shah, and S. Shah, *Web Hacking: Attacks and Defense*. Addison-Wesley, August 2002.

[108] E. Birkholz and S. McClure, *Special Ops: Host and Network Security for Microsoft, UNIX, and Oracle.* Syngress Publishing, February 2003.

[109] L. D. Stein, *Web Security: A Step-by-Step Reference Guide.* Addison-Wesley, December 1997.

[110] S. Garfinkel, *Web Security, Privacy and Commerce.* O'Reilly Media, 2nd ed., January 2002.

[111] S. Siddharth and P. Doshi, "Five common web application vulnerabilities." http://www.securityfocus.com/infocus/1864, April 2006.

[112] K. Raina, "Trends in web application security." http://www.securityfocus.com/infocus/1809, October 2004.

[113] "Sans infosec reading room - web servers." http://www.sans.org/reading_room/whitepapers/webservers/.

[114] "Security focus." http://www.securityfocus.com/.

[115] "UK payments association." http://www.apacs.org.uk/.

[116] "Secure software forum." http://www.securesoftwareforum.com/.

[117] "The top most critical web application security vulnerabilities." The Open Web Application Security Project (OWASP) http://sourceforge.net/project/showfiles.php?group_id=64424\&package_id=70827, January 2004.

[118] "Application vulnerability description language (AVDL) v1.0." OASIS Application Vulnerability Description Language (AVDL) TC, February 2004.

[119] "Threat classification." Web Application Security Consortium (WASC) http://www.webappsec.org, 2004.

[120] D. M. Kienzle and M. C. Elder, "Final technical report: Security patterns for web application development," tech. rep., Defense Advanced research Projects Agency (DARPA), 2001.

[121] S. Burbeck, "Applications programming in smalltalk-80: How to use model-view-controller (MVC)." http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html, 1992.

[122] B. McLaughlin, *Building Java Enterprise Applications, Vol. 1: Architecture.* O'Reilly Media, 2002.

[123] L. Constantine and L. Lockwood, *Software for Use.* Addison-Wesley, January 2000.

[124] N. Koch, H. Baumeister, R. Hennicker, and L. Mandel, "Extending UML for modeling navigation and presentation in web applications," in *Proceedings of 3rd International Conference on the Unified Modeling Language (UML2000)*, vol. 1939 of *LNCS*, (York, UK), October 2000.

[125] A. McDonald and R. Welland, "Agile web engineering (AWE) process," tech. rep., Department of Computer Science, University of Glasgow, UK, December 2001.

[126] http://www.adaptivesd.com/, April 2005.

[127] http://alistair.cockburn.us/crystal/crystal.html, March 2007.

[128] http://extremeprogramming.org/, July 2006.

[129] http://featuredrivendevelopment.com/, May 2007.

[130] http://www.controlchaos.com/, May 2007.

[131] M. Fowler, "The new methodology." http://martinfowler.com/articles/newMethodology.html, December 2005.

[132] K. Beck, "Embracing change with extreme programming," *IEEE Computer*, vol. 33, no. 10, pp. 70–77, 1999.

[133] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd ed., November 2004.

[134] R. Ramsin, *The Engineering of an Object-Oriented Software Development Methodology*. PhD thesis, Computer Science Department, University of York, April 2006.

[135] P. Coad, E. Lefebvre, and J. D. Luca, *Java Modeling In Color With UML: Enterprise Components and Process*. Prentice Hall, June 1999.

[136] S. Khramtchenko, "Comparing extreme programming and feature driven development in academic and regulated environments." http://www.featuredrivendevelopment.com/files/FDD_vs_XP.pdf.

[137] S. R. Palmer, "Feature-driven development and extreme programming." http://www.informit.com/articles/printerfriendly.asp?p=26055\&rl=1, March 2002.

[138] J. D. Luca, "FDD implementations." http://www.nebulon.com/articles/fdd/fddimplementations.html, May 2007.

[139] E. G. Aydal, R. F. Paige, H. Chivers, and P. J. Brooke, "Security planning and refactoring in extreme programming," in *Proceeding of the 7th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2006)*, Spring-Verlag LNCS 4044, (Oulu, Finland), pp. 154–163, June 2006.

[140] K. Beznosov, "Extreme security engineering: On employing XP practices to achieve 'good enough security' without defining it," in *Proceedings of First ACM Workshop on Business Driven Security Engineering BizSec*, (Fairfax, VA), October 2003.

[141] M. Fowler, "Is design dead?." http://www.martinfowler.com/articles/designDead.html, May 2004.

[142] J. Wäyrynen, M. Bodén, and G. Boström, "Security engineering and extreme programming: An impossible marriage?," in *Extreme Programming and Agile Methods - XP/Agile Universe 2004 (XP2004)* (C. Zannier, H. Erdogmus, and L. Lindstrom, eds.), vol. 3134 of *LNCS*, pp. 117–128, Springer, August 2004.

[143] S. W. Ambler, "Agile/lean documentation: Strategies for agile software development." http://www.agilemodeling.com/essays/agileDocumentation.htm, December 2007.

[144] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices.* Prentice Hall, 1 ed., October 2002.

[145] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New directions on agile methods: A comparative analysis," in *the 25th International Conference on Software Engineering ICSE 2003*, (Portland, Oregon, USA), pp. 244–254, May 2003.

[146] IEEE, "IEEE recommended practice for software requirements specifications," in *Software Requirements Engineering* (R. Thayer and M. Dorfman, eds.), ch. 5, IEEE Computer Society Press, 1993.

[147] G. Kotonya and I. Sommerville, "Requirements engineering with viewpoints," *Software Engineering Journal*, vol. 11, pp. 5–18, January 1996.

[148] A. van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in *Proceedings of 22nd International Conference on Software Engineering (ICSE2000).*, (Limerick), ACM Press, June 2000.

[149] P. T. Devanbu and S. G. Stubblebine, "Software engineering for security: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering ICSE '00*, (Limerick, Ireland), pp. 227–239, ACM Press, 2000.

[150] D. G. Firesmith, "Engineering security requirements," *Journal of Object Technology*, vol. 2, pp. 53–68, January/February 2003.

[151] R. Crook, D. Ince, L. Lin, and B. Nuseibeh, "Security requirements engineering: When anti-requirements hit the fan," in *Proceedings of IEEE International Requirements Engineering Conference (RE'02)*, 2002.

[152] J. Rushby, "Security requirements specifications: How and what?," in *Symposium on Requirements Engineering for Information Security (SREIS)*, (Indianapolis), March 2001.

[153] G. Wimmel and A. Wißpeintner, "Extended description techniques for security engineering," in *Proceedings of IFIP TC11 16th International Conference on Information Security (IFIP/Sec'01)* (M. Dupuy and P. Paradinas, eds.), (Paris, France), pp. 470–485, Kluwer Academic Publishers, June 2001.

[154] J. D. Moffett, "Requirements and policies," in *Policy Workshop*, (Bristol, UK), HP-Laboratories, 1999.

[155] L. Chung, "Dealing with security requirements during the development of information systems," in *Proceedings of 5th International Conference Advanced Information Systems Engineering, CAiSE 93* (C. Rolland, F. Bodart, and C. Cauvet, eds.), pp. 234–251, 1993.

[156] P.-J. Fontaine, "Goal-oriented elaboration of security requirements," MSc. thesis, Department of Computing Science, University of Louvain, June 2001.

[157] I. Sommerville, *Software Engineering.* Addison-Wesley, 6th ed., 2001.

[158] R. A. Caralli and W. R. Wilson, "The challenges of security management." CERT http://www.cert.org/archive/pdf/ESMchallenges.pdf, 2004.

[159] R. A. Caralli, "Managing for enterprise security," tech. rep., CERT, Software Engineering Institute (SEI), CMU/SEI/2004-TN-046, December 2004.

[160] D. J. Anderson, *Agile Management For Software Engineering: Applying the Theory of Constraints for Business Results*. Coad Series, Prentice Hall PTR., 2003.

[161] S. Khramtchenko, "FDDPMA user guide." http://fddpma.sourceforge.net/help/fddpma_user_guide.pdf, June 2005.

[162] P. Kumar, *J2EE Security: for Servlets, EJBs, and Web Services*. Prentice Hall PTR, April 2004.

[163] "Authentication, authorization, and access control." Apache HTTP Server Version 1.3 Document http://httpd.apache.org/docs/1.3/howto/auth.html, January 2004.

[164] B. Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21(5), pp. 61–72, 1988.

[165] M. Lehman, "Programs, life cycles, and laws of software evolution," in *IEEE special issue on software engineering*, pp. 1060–1076, September 1980.

[166] M. Lehman, "Software engineering: the software processes and their support," *IEEE Software Engineering, special issue on software environment and factories*, pp. 243–258, September 1991.

[167] M. Lehman, *Encyclopedia of Software Engineering*, ch. Software Evolution, pp. 1202–1208. 1994.

[168] B. Boehm, "Improving software productivity," *IEEE Computer*, vol. September, pp. 43–57, 1987.

[169] E. Swanson and C. Beath, *Maintaining Information Systems Organizations*. John Wiley and Sons, 1987.

[170] J. Collofello and J. Buck, "Software quality assurance for maintenance," *IEEE Software*, pp. 46–51, 1987.

[171] D. Weiss and V. Basili, "Evaluating software development by analysis of changes," *IEEE Transactions of Software Engineering*, pp. 157–168, Feberary 1985.

[172] S. A. Bohner, *A Graph Traceability Approach to Software Change Impact Analysis*. PhD thesis, George Mason Unversity, Fairfax, VA, USA, 1995.

[173] J. Ferrante, K. Ottenstein, and J. Warren, "The program dependence graph and its use in optimization," *ACM Transaction of Programming Languages and Systems*, pp. 319–349, July 1987.

[174] J. Loyall and S. Mathisen, "Using dependence analysis to support software maintenance," in *Proceeding of Conference of Software Maintenance*, (Los Alamtos, Califonia), pp. 282–291, 1993.

[175] A. Podgurski and L. Clark, "A formal model of program dependencies and its implications for software testing, debugging, and maintenance," *IEEE Transaction of Software Engineering*, pp. 965–979, September 1990.

[176] T. Smith, "System development and requirements management," tech. rep., Paramax Systems Corporation (now Loral Corporation), Reston, VA, January 1992.

[177] B. Nejmeh and T. Dickey, "Traceability technology at the software productivity consortium," Tech. Rep. SPC-1.0-881130-40-00-N, Software Productivity Consortium, Herndon, VA, November 1988.

[178] R. F. Paige, R. Charalambous, X. Ge, and P. J. Brooke, "Towards agile engineering of high-integrity systems," in *Proceeding of the 27th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2008)*, (Newcastle upon Tyne, UK), September 2008.