Usablity and Security Trade-Off: A Design Guideline

Yasser M. Hausawi
Florida Institute of Technology
Melbourne, FL, USA
yhausawi@my.fit.edu

William H. Allen
Florida Institute of Technology
Melbourne, FL, USA
wallen@fit.edu

ABSTRACT

Requirements engineering and design are the first two phases of the Software Development Life-Cycle. Considerable research has addressed the requirements phase and a number of well-regarded tools exist to assist with that process. The design phase can also make use of a wide range of tools, including design principles, activities, best practices, techniques, and patterns, to improve the incorporation of requirements into the software design documents. However, the process of selecting the appropriate design tools to support each requirement is a complex task that requires considerable training and experience. It is also possible that design tools selected for different requirements can conflict with each other, reducing their effectiveness, increasing complexity, impacting usability or potentially causing security vulnerabilities. In this paper, we propose guidelines for selecting appropriate design tools to support the integration of usability and security requirements in the software design phase and to resolve conflicts between those tools. We demonstrate this approach with a case study that illustrates the design tool selection and analysis process.

Keywords

Software Design, Security, Usability, Usable-Security, Patterns, Best Practices.

1. INTRODUCTION

The second phase of the Software Development Life-Cycle (SDLC) addresses the design of a system and makes use of a number of tools that aid in interpreting the non-technical textual documents produced during the requirements phase into a technical document that can be used to construct a software product [15]. A wide variety of tools have been developed for use during the design phase such as design principles, activities, best practices, techniques, and patterns [2].

ACMSE '14 March 28-29, 2014 Kennesaw, GA USA.

ACM 978-1-4503-2923-1/14/03 $15.00.
http://dx.doi.org/10.1145/2638404.2638483 .

It is important that decisions made during the design phase address specific requirements and this often requires feedback from the design phase to the requirements engineering process. Unfortunately, security and usability are often regarded as non-functional quality attributes. As a result, they may not be considered until near the end of the development process or when negative security or usability issues appear in software products after deployment. There is a body of research on the inclusion of security best practices and patterns during the design phase [15, 19, 24, 10]. Similarly, usability techniques and patterns for the design phase have received some attention [4, 5, 6] and research has also shown the benefits of integrating security and usability into usable-security to create a balance between those two attributes that enhances both. Additionally, design tools for usable-security have been introduced [21, 9].

The process of linking the available design tools to security and usability requirements needs further research to ensure that those attributes are properly integrated into the software development process. Design tools (such as patterns) are not usually identified and selected during the requirements phase [2]. Delaying those choices until the beginning of the architectural design process places constraints on the design of the system [6]. Therefore, it is usually better to make those decisions as early as possible in the design phase.

The work proposed here provides systematic guidance for linking the usability, security, and usable-security requirements to appropriate design tools (i.e. principles, practices, patterns, and techniques) that can be used during the architectural design stage. The remainder of this article is structured as follows: Section 2 reviews security, usability, and usable-security tools that are used during the SDLC in general, and during the design phase in particular. Section 3 introduces the proposed guidance. Section 4 presents a case study to illustrate and demonstrate the guideline in details, and finally, Section 5 concludes this article and describes future directions for developing the proposed guidelines.

2. BACKGROUND

2.1 Security Practices and Patterns

Security practitioners have created a wide range of tools that can be used during the Software Development Life-Cycle to integrate security into software products. Some of the tools, such as software patterns, have been adapted from other disciplines while others were developed specifically to address software security issues. The Software Assurance Forum for Excellence in Code (SAFECode) pub-

lished 30 fundamental best practices for secure software development [24]. McGraw [19] developed seven software security practices, called Touchpoints, to be used during the software development life-cycle. Among the seven practices, two are specifically used during the design phase. At Microsoft, Howard and Lipner [15] developed a seven-phase Security Development Life-cycle based on the traditional software development life-cycle. The Microsoft SDL is a collection of 17 security activities applied to the software development process to produce more secure software. Three of these activities are specifically used during the design phase. Alkussayer, et al., [2] developed a framework to integrate security patterns and best security practices into the SDLC. They collected 23 well-known security practices and activities from different sources, such as [24, 19, 15]. Among these practices and activities, four are used during the design phase. They integrated security patterns into the SDLC as well.

The use of patterns to enhance design was first introduced in 1975 by the construction architect Christopher Alexander in his book *The Oregon Experiment* [1, 10]. Coplien [14] defines a pattern as, "a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to solve themselves". Computer scientists, software engineers and architects adapted the pattern concept to software design and programming in 1992 when Ralph Johnson [17] proposed 10 patterns for designing graphical editors. Johnson's contribution was further developed in collaboration with Gamma, Helm and Vlissides [8], a group known as the Gang-of-Four (GoF). Subsequently, Hanmer [11] published 63 general software patterns in his book *Patterns for Fault Tolerant Software*.

Reflecting on the proven benefit of using patterns in software development, security practitioners began to incorporate the use of patterns into the development of secure software . Yoder and Barcalow [25] developed their first set of security patterns in 1997. They devised seven security patterns for application development. In 2006, Schumacher et al. [23] introduced 46 enterprise-oriented security patterns for integrating security with systems engineering to address enterprise-wide issues. In 2011, Hafiz et al. [10] described a mechanism for enhancing a security pattern language through cataloging, classifying, and relating the patterns. They presented 96 security patterns to aid in software development.

2.2 Usability Techniques and Patterns

As with security, usability specialists have produced a set of tools that can be used during the Software Development Life-Cycle to ensure that usability will be properly incorporated into software products. Some of the tools have been adopted from other disciplines, such as patterns, while others have been developed to address particular usability issues. Folmer and Bosch [6] define the term *Usability Pattern* as, "a technique or mechanism that can be applied to the design of the architecture of a software system in order to address a need identified by a usability property at the requirements stage." They have developed a framework that bridges usability and software architecture through generating 19 usability patterns, all of which are used during the design phase. Ferre [4] reviewed the usability engineering literature to identify published usability techniques. He collected 82 well-known techniques, but he showed that only 51 of those usability techniques meet the criteria for inclusion into the software development process. In [5] Ferre et al. extended the work of [4], by collecting 95 usability techniques from the HCI literature. They reduced this collection to 35 techniques that they used in a framework for integrating usability practices into the software development process based on six criteria. Among the selected HCI techniques, seven are used during the design phase.

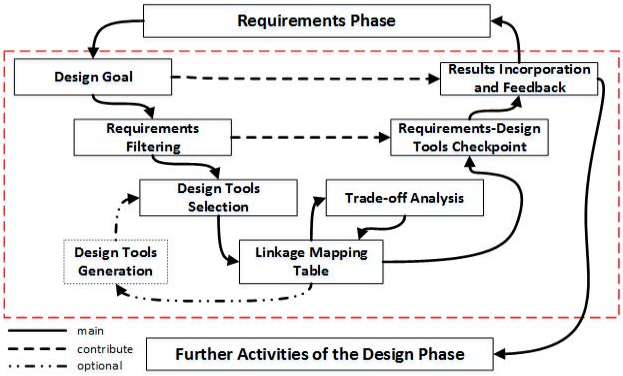

Figure 1: The proposed usable-security design.

2.3 Usable-Security Principles and Patterns

Garfinkel [9] provided a collection of principles and generated patterns for integrating both security and usability into systems development during the design phase. He stated that "usability and security can be made synergistic by redesigning systems with specific principles and through the adoption of well-defined patterns" [9]. To reach this goal, he used six design principles as a foundation for generating 21 patterns and showed how these principles and patterns can be used together to achieve usable-security in computer system design.

3. GUIDELINES FOR USABLE-SECURITY

Figure 1 shows the proposed process for selecting appropriate design tools to support the requirements specified in the first phase of the SDLC. There are eight steps in this process: 1) selection of priority sensitivity, 2) requirements filtering, 3) selection of appropriate design tools, 4) creation of a linkage mapping table, 5) generation of new design tools, 6) analysis of the trade-off matrix, 7) a check on how well requirements and design tools match, and 8) incorporation of the results (if step 7 is successful) or feedback to the requirements phase (if the requirements should be modified).

3.1 Selection of priority sensitivity

The first step is to determine an important goal of the design phase by clearly stating the targeted priority levels for security, usability, and usable-security. This step is directly related to the priority levels assigned to requirements in the previous phase. There are two different strategies which could be followed: either identifying a unified priority target that applies to all the requirements (generalization), or identifying separate target priority levels for each requirement individually (specialization). In both cases, design-

ers must determine whether they intend to provide a high, medium or low level of sensitivity for the priority of the stated requirements when assigning the design tools. These sensitivity levels are described below.

3.1.1 Higher sensitivity to requirement priorities.

At the higher level of sensitivity, all of the requirements are associated with appropriate design tools (e.g., principles, practices, patterns, or techniques), regardless of the priority assigned in the requirements phase. If any requirement can't be associated with an appropriate design tool, it cannot be processed through the design phase until this issue is addressed. This will be reported to the requirements engineers as it may require adjustments to the requirements.

3.1.2 Medium sensitivity to requirement priorities.

The medium level of sensitivity ensures that appropriate design tools are associated with all of the medium to high priority requirements. If any of these requirements can't be associated with an appropriate design tool, they cannot be processed through the design phase until this issue is addressed. This will also be reported to the requirements engineers. Low priority requirements may optionally be associated with design tools, but a failure to associate any of them will not require corrective action.

3.1.3 Lower sensitivity to requirement priorities.

The lower level of sensitivity only expects that design tools are associated with the high priority requirements. Specifying tools for medium or low priority requirements is optional. To achieve this goal, all high priority requirements must be assigned to appropriate design tools. If any of the high priority requirements can't be associated with an appropriate design tool, they cannot be processed through the design phase until this issue is addressed and this failure will be reported to the requirements engineers.

3.2 Requirements filtering

Based on the target levels of sensitivity specified in the previous step, the requirements are filtered to determine which of them must be associated with appropriate design tools. It should be emphasized that all requirements will be included in the design phase, this filtering process is intended to separate the requirements that must be associated with design tools from those requirements for which design tools are optional. The outcome of the filtering is used to ensure that requirements with the specified level of priority are associated with appropriate design tools.

3.3 Selection of appropriate design tools

As mentioned above, there are a number of tools available for enhancing the software development process and some are particularly useful during the design phase. The most common tools are: design principles, best practices, techniques, activities, and patterns. Among them, patterns are possibly the most widely used because they provide reusable, pre-determined solutions for well-known problems that occur frequently during and after software development [18]. However, suitable patterns have not been developed to address all of the problems that can arise in software development. Therefore, incorporating best practices, techniques, and activities during the design phase helps to fill the gap caused by the lack of patterns for some important tasks [2].

Table 1: Linkage Mapping Table

Requirements				Design Tool		
Reqs.	Pro_1	Pro_2	Pro_m	Pro_1	Pro_2	Pro_m
Req_1	Low	Low	High			DT_x
Req_2	High	High	Low	DT_y	DT_z	
Req_n	High	High	Low	DT_y	DT_z	

Unfortunately, the utilization of these design tools and techniques varies among software developers and not all developers are able to use the available tools properly [2]. Thus, guidance in the selection of suitable design tools is important to supplement those developers' skills and experience.

3.4 Creation of a linkage mapping table

After specifying the desired requirement sensitivity levels, filtering the requirements based on the specified sensitivity levels, and selecting the suitable types of design tools to be used, the design tools are assigned to the appropriate requirements, where the relationship between the requirements and the design tools is a many-to-many mapping. The mapping idea comes from the SALUTA APT [7], where scenarios and usability are linked in an Attribute Preference Table (APT). We adapted the APT to create a Linkage Mapping Table (LMT) to link requirements, attribute properties, and design tools. Table 1 shows an example of the linkage mapping table, where *Req* represents a requirement, *Pro* represents attribute properties (i.e., *Low*, *Medium* or *High* priority), and *DT* represents a specific design tool. Table 1 shows the mapping between selected requirements and the design tools that will aid in meeting those requirements during the design phase.

3.5 Generation of new design tools

It is possible that no design tools are available to support a particular requirement that is entered into the LMT. The diagram in Figure 1 shows a solution to this problem, the (optional) *Design Tools Generation* loop. This option provides an opportunity to generate a new design tool, such as new patterns, design activities, techniques, or practices, for use by the *Design Tools Selection* component. This step may also cause a refinement to the linkage mapping table.

3.6 Analysis of the trade-off matrix

Working through quality attributes that have conflicting objectives (e.g. security and usability) to make them synergistically integrated is difficult [3]. However, dealing with such cases is increasingly common in software development projects. Unfortunately, many of the existing design tools were developed with a narrow focus and conflicts between two or more tools are possible. During this step we address this issue by creating a two-dimensional trade-off matrix that lists all the design tools in the linkage mapping table to allow us to identify potential conflicts between these tools. Table 2 depicts the trade-off matrix which is used to refine the linkage mapping table (see Figure 1).

3.7 Checking the match between requirements and design tools

After completion of the trade-off matrix and refining the linkage mapping table, the entries in the linkage mapping table are matched against the requirements to check the

Table 2: Design Tools Trade-Off Matrix

	DT_x	DT_y	DT_z
DT_x		X	
DT_y	X		X
DT_z		X	

compliance of the design phase with the requirements phase. This step reports any missing design tools or conflicts that indicate an inability to meet the requirements during the design phase. The results of this analysis are delivered to the final step to determine whether the design process is complete or needs further work.

3.8 Incorporation of results and feedback

The results from the previous step are analysed for completeness and consistency. If it is determined that the design phase is complete, the linkage mapping table is passed to the designers and architecture engineers to provide them with the essential design tools for the software product. If not, the results of the analysis are passed back to the requirements engineers for reassessment of the requirements.

4. A CASE STUDY: SECURE AND USABLE AUTHENTICATION

To better illustrate the advantages of our guidelines, we will use an authentication system that we are currently developing as part of our research in usable-security. Authentication is one of two areas receiving significant attention from the usable-security community [21]. The system under development is a demonstration of an authentication method that we call the Choice-Based Authentication Approach (CBAA). This method is based on two concepts. The first is to allow end users to select their preferred authentication method to enhance usability and universal access during the sign-up process [13]. The second concept is to increase the complexity for adversaries by displaying all of the possible authentication methods during the login process. Our aim is to demonstrate the usefulness of the guidelines that we proposed in this paper. Thus, a brief subset of the CBAA demonstration is presented here for each step of the guideline.

4.1 Selection of priority sensitivity

As described earlier, this step relies on the requirements engineering phase of the SDLC. In order to identify the target priority levels, we used the results of rating the requirements for the authentication system under development based on the Assessment Framework for Usable-Security (AFUS)[1] [12] (see Tables 3 and 4), where all rating values are selected for higher sensitivity, the rating values starting from 6 are selected for medium sensitivity, and the rating values starting from 7 are selected for the lower sensitivity level. We adopted the generalization strategy that applies one design goal over all the requirements and we used the medium level of priority sensitivity, which means that priority values starting from 6 should be associated with appropriate design tools. These parameters were specifically to present a typical demonstration of the guidelines.

[1]AFUS is an assessment methodology that we are developing based on well-known tools from the Decision Science field. AFUS rates the requirements via adapting two well-known assessment methodologies: the OWASP RRM [20] for security, and the SALUTA APT [7] for usability.

Table 3: Security Requirements Filtering

Code	Requirement	CONF	INT	AVA
$SReq_1$	The system shall provide one unique login interface.	3	1	6
$SReq_2$	The system shall show all authentication methods on the login interface.	7	1	3
$SReq_3$	The system shall meet the NIST 800-Series.	6	3	3
$SReq_4$	The system shall not store users' private information.	6	6	1
$SReq_5$	The system shall provide a separate sign-up interface.	1	1	3

4.2 Requirements filtering

Based on the sensitivity targets from the previous step (i.e., "medium" sensitivity), the requirements are filtered to identify any that are not expected to be associated with a design tool. Table 3 shows a list of security requirements, where **CONF, INT,** and **AVA** represent the security properties: confidentiality, integrity, and availability, respectively [22]. At least one of those three properties must meet the expected priority level to be assigned a design tool. For example, in Table 3, $SReq_5$ will be eliminated because it doesn't have rating values of 6 or higher and is considered to be a low priority requirement. Similarly, Table 4 shows a list of usability requirements, where **EFF1, EFF2,** and **SAT** represent the usability properties: effectiveness, efficiency, and satisfaction, respectively [16].

4.3 Selection of appropriate design tools

In our example, all of the available design tools are considered. Therefore, best practices, techniques, activities, and patterns can all be assigned to requirements during the design phase.

4.4 Creation of a linkage mapping table

After completion of the above steps of the guideline, a linkage mapping table can be created through associating each requirement with appropriate design tools. For example, the first security requirement $SReq_1$ on Table 5 mandates providing a unified individual login interface that all users should use during the login process. This requirement can be met through applying two well-known security patterns: *SINGLE ACCESS POINT* [25] and *FRONT DOOR* [6]. In the same way, usability requirements are associated with appropriate design tools as shown in Table 6. We provided examples of the design tools used to meet the high priority requirements of Tables 3 and 4.

DT_1:SINGLE ACCESS POINT [25]
DT_2:FULL VIEW WITH ERROR [25].
DT_3:SECURITY POLICY ENFORCEMENT [10].
DT_4:ANONYMITY SET [10].

Table 4: Usablility Requirements Filtering

Code	Requirement	EFF1	EFF2	SAT
$UReq_1$	The system shall help the users to avoid making mistakes by providing clear steps to be followed.	9	1	1
$UReq_2$	The system shall make the users satisfied and want to use it by adopting user-friendly interfaces.	3	3	9
$UReq_3$	The system shall allow users to compete the required tasks in an acceptable amount of time.	3	9	7
$UReq_4$	The system shall not increase the users' cognitive load through providing consistent steps and interfaces.	6	6	6

Table 5: The Linkage Mapping Table for the Security Requirements

Security Requirements Rating				
Requirements		CONF	INT	AVA
$SReq_1$	**Rate**	Low	Low	High
	Design Tool			DT_1
$SReq_2$	**Rate**	High	Low	Low
	Design Tool	DT_2		
$SReq_3$	**Rate**	High	Low	Low
	Design Tool	DT_3		
$SReq_4$	**Rate**	High	High	Low
	Design Tool	DT_4	DT_4	

DT_5:*FORM/FIELD VALIDATION [6].*
DT_6:*PROGRESS INDICATION [6].*
DT_7:*MULTI-TASKING [6].*
DT_8:*EMULATION [6].*
DT_9:*ALERTS [6].*
DT_{10}:*SHORTCUTS [6].*
DT_{11}:*COGNITIVE WALK-THROUGH [4].*
DT_{12}:*ACTIONS FOR MULTIPLE OBJECTS [6].*
DT_{13}:*HISTORY LOGGING [6].*
DT_{14}:*PROTOTYPING [4].*
DT_{15}:*WIZARD [6].*

4.5 Generation of new design tools

As stated above, this step is an optional one and added to the guideline steps in order to make the guideline flexible enough for design tools development and generation. However, there are enough design tools that meet the requirements of our authentication system. Therefore, there is no need to generate new tools in this case.

4.6 Analysis of the trade-off matrix

Table 6: The Linkage Mapping Table for the Usability Requirements

Usability Requirements Rating				
Requirements		EFF1	EFF2	SAT
$UReq_1$	**Rate**	High	Low	Low
	Design Tool	DT_5		
$UReq_2$	**Rate**	Low	Low	High
	Design Tool			DT_6
$UReq_3$	**Rate**	Low	High	High
	Design Tool		DT_7	DT_8
$UReq_4$	**Rate**	High	High	High
	Design Tool	DT_{11}	DT_9	DT_{10}
$UReq_5$	**Rate**	Low	High	High
	Design Tool		DT_{13}	DT_{14}
$UReq_6$	**Rate**	High	Low	High
	Design Tool	DT_{12}		DT_{15}

Table 7: Design Tools Trade-Off Matrix for the CBAA Demonstration

	DT_2	DT_{12}	DT_3	DT_{11}
DT_2		X		
DT_{12}	X			
DT_3				X
DT_{11}			X	

The trade-off matrix analysis is used to identify any conflicts between the design tools selected to address the requirements' needs during the design phase. Table 7 shows examples of conflicting cases where the adopted design tools work against one another. For instance, the *FULL VIEW WITH ERROR* pattern [25] conflicts with the usability activity for avoiding dangling alternatives and double vision that is *ACTIONS FOR MULTIPLE OBJECTS* [6]. Another conflicting case appears when the *SECURITY POLICY ENFORCEMENT* pattern [10] is used to ensure applying security policy, where it may conflict with the *Cognitive Walk-through* [4] usability technique. Therefore, these conflicting design tools have been identified and analysed to allow us to address the conflicts. In the case of our authentication, the conflict between design tools is resolved by adopting usable-security design tools such as *Interaction Design* activities [4], and *LEAST SURPRIZE / LEAST ASTONISHMENT* and *PROVIDE STANDARDIZED SECURITY POLICIES* [9].

4.7 Checking the match between requirements and design tools

After refining and completing the linkage mapping tables, a matching check is performed to make sure that the requirements have been completely covered by appropriate design tools and there are no missing tools, as missing tools indicate that the design phase is limited and therefore unable to properly meet the needs of all of the high priority requirements. In our demonstration, we could successfully cover all the requirements through assigning each requirement with the appropriate design tools. Moreover, conflicts have been resolved by applying usable-security principles and patterns [9].

4.8 Incorporation of results and feedback

After a successful analysis, the completed linkage mapping tables are ready to be passed to the architecture engineers so they can use them to select the appropriate design tools during the architecture design activities.

5. CONCLUSION

In this paper, we proposed guidelines to improve the selection of the appropriate design tools for security, usability, and usable-security to ensure that the design process meets the requirements' needs. Moreover, our guidelines help to identify trade-offs in the selection of design tools by discovering conflicts and helping to overcome contradictions in design decisions. We recognize that more work is needed to determine how well our approach enhances the development of software products in terms of both linking the requirements to the design tools, and integrating security and usability synergistically for better usable-security.

Acknowledgment

The authors would like to thank the Institute of Public Administration (IPA) in Saudi Arabia for their support of this work

6. REFERENCES

[1] C. Alexander. *The Oregon Experiment*, volume 3. Oxford University Press, USA, 1975.

[2] A. Alkussayer and W. H. Allen. The ISDF Framework: Integrating security patterns and best practices. *Advances in Information Security and Its Application*, pages 17–28, 2009.

[3] C. Braz, A. Seffah, and D. MâĂŹRaihi. Designing a trade-off between usability and security: A metrics based-model. In *Human-Computer Interaction–INTERACT 2007*, pages 114–126. Springer, 2007.

[4] X. Ferre. Integration of usability techniques into the software development process. In *International Conference on Software Engineering (Bridging the gaps between software engineering and human-computer interaction)*, pages 28–35, 2003.

[5] X. Ferre, N. Juristo, and A. Moreno. Framework for integrating usability practices into the software process. *Product focused software process improvement*, pages 202–215, 2005.

[6] E. Folmer and J. Bosch. Usability patterns in software architecture. In *Proceedings of the 10th International Conference on Human-Computer Interaction (HCI 2003)*, pages 93–97, 2003.

[7] E. Folmer, J. van Gurp, and J. Bosch. Scenario-based assessment of software architecture usability. In *ICSE Workshop on SE-HCI*, pages 61–68. Citeseer, 2003.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Elements of reusable object-oriented design, 1995.

[9] S. Garfinkel. *Design Principles and Patterns for Computer Systems that are Simultaneously Secure and Usable*. PhD thesis, Massachusetts Institute of Technology, 2005.

[10] M. Hafiz, P. Adamczyk, and R. Johnson. Growing a pattern language (for security). In *Proceedings of the 18th conference on pattern languages of programs (PloP)*, 2011.

[11] R. Hanmer. *Patterns for Fault Tolerant Software*. Wiley Publishing, 2007.

[12] Y. M. Hausawi and W. H. Allen. An assessment framework for usable-security based on decision science. In *Human Aspects of Information Security, Privacy, and Trust*, pages 33–44. Springer, 2014.

[13] Y. M. Hausawi, W. H. Allen, and G. S. Bahr. Choice-based authentication: A usable-security approach. In *Universal Access in Human-Computer Interaction. Design and Development Methods for Universal Access*, pages 114–124. Springer, 2014.

[14] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen. An analysis of the security patterns landscape. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, page 3. IEEE Computer Society, 2007.

[15] M. Howard and S. Lipner. *The Security Development Lifecycle*. Microsoft Press, 2009.

[16] W. ISO. 9241-11. ergonomic requirements for office work with visual display terminals (VDTs). *The international organization for standardization*, 1998.

[17] R. E. Johnson. Documenting frameworks using patterns. In *ACM Sigplan Notices*, volume 27, pages 63–76. ACM, 1992.

[18] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns repository version 1.0. *DARPA, Washington DC*, 2002.

[19] G. McGraw. The security lifecycle-the 7 touchpoints of secure software-just as you can't test quality into software, you can't bolt security features onto code and expect it to become hack-proof security. *Software Development*, 13(9):42–43, 2005.

[20] OWASP. Risk rating methodology, 2013.

[21] B. D. Payne and W. K. Edwards. A brief introduction to usable security. *Internet Computing, IEEE*, 12(3):13–21, 2008.

[22] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*. Prentice Hall PTR, 2006.

[23] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*, volume 7. Wiley, 2006.

[24] S. Simpson. Fundamental practices for secure software development: A guide to the most effective secure development practices in use today, 2011.

[25] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. *Urbana*, 51:61801, 1998.