A Multi-Dimensional Classification for Users of Security Patterns

Michael VanHilst[1], Eduardo B. Fernandez[1] and Fabrício Braz[2]

[1]Florida Atlantic University, Department of Computer Science and Engineering
777 Glades Road, Boca Raton, Florida 33431, U.S.A.

[2]Universidade de Brasília, Departamento de Ciência da Computação, Campus
Universitário - Asa Norte, 70910-900 Brasília - DF - Brazil

Abstract. This paper presents a classification for security patterns that addresses the needs of users. The approach uses a matrix defined by dividing the problem space along multiple dimensions, and allows patterns to occupy regions, defined my multiple cells in the matrix. It supports filtering for narrow or wide pattern selection, allows navigation along related axes of concern, and identifies gaps in the problem space that lack pattern coverage. Results are preliminary but highlight differences with existing classifications.

1 Introduction

Computer application security is no longer an option; it is a necessity. Our well being depends on an interconnected and ever growing web of information systems and applications. Threats to those systems are real with the potential for dire consequences. Threats come from all directions – external, internal, accidental, intentional, and highly organized. Any point can be exploited; comprehensive security is not easy to achieve.

Security must be addressed for all components in all activities in all phases of the software lifecycle. For complex applications, a comprehensive approach to security – top-to-bottom, beginning-to-end, and everywhere-in-between – is a requirement. The problem is vast. Inspection and testing is only a small part. Building secure software involves more than plugging holes.

The security challenge is compounded by the way real world applications are built. Consider the issues of component source and standards compliance. Components can come from new code, legacy application, commercial-off-the-shelf, open-source, remote web service, reuse library, runtime script, wizard code generation, model transformation, and outsourced development. It is a rare and inefficient project that doesn't leverage more than a couple of these sources. Development, design, and deployment must comply with multiple standards, including FISMA [10], HIPAA [13], Sarbanes/Oxley [24], and Graham-Leach-Bliley [11]. Amidst this complexity, the focus on security must be maintained.

In our work, we have come to the conclusion that a single end-to-end methodology, by itself, cannot realistically address all the security concerns in every situation and variation a developer is likely to face. While a systematic approach is essential, it must be augmented by the delivery of focused elements of security knowledge when and where they are needed.

We address the problem of secure software development through the use of patterns. Patterns are well known solutions to common problems in given contexts. Patterns make it possible for developers looking at a specific problem or task to benefit from the experience of others in a conveniently packaged form. While the best known patterns are design patterns, we develop security patterns for a wide range of concerns and many different phases of the software life cycle [4, 6, 23].

There are several groups developing security patterns [27, 28]. Many books on security also package material in the form of patterns [28]. With so many patterns now available, it ought to be possible for software developers to find patterns for just about any phase, activity, and need. But how do developers find patterns that are relevant to what they are doing? Are all concerns covered, and if not, where are the gaps?

Traditional pattern classification serves pattern collectors' need to determine the extent to which a new pattern is the same as other patterns in the collection. Such classifications are typically hierarchical [8, 9, 12, 30], and based on the solution being presented rather the problems being addressed [12]. Existing classifications mimic classification in biology, where hierarchical classification enables identification by following a decision tree (e.g. Does it have a backbone?) and conforms to a view of. speciation by mutation from a parent gene. But biological classification is not intended for the consumer (e.g. Does it offer taste and protein to complement my menu?), and, not surprisingly, yields little value on a trip to the grocery store. Similarly, traditional pattern classification is ill suited to the needs of developers.

The remainder of the paper is classified as follows: Section 2 gives an overview of our approach, describing the matrix of concerns, its theoretical foundation, and how it differs from existing classifications based on hierarchies and tagging. Sections 3 and 4 provide more details about the dimensions of the matrix and their use. Section 5 gives some preliminary observations and an example of its use, while Section 6 discusses related work.

2 Matrix of Concerns

We propose to address the problem of pattern classification and problem coverage through the use of a multi-dimensional matrix of concerns. Each dimension of the matrix is a list of concerns along a single continuum, defined by a common set of distinctions. A single dimension thus addresses a simple logical concept. The categories within the dimension should be easily understood and represent widely used and accepted classifications within that concept. For example, one dimension would be a list of life-cycle activities, covering domain analysis, requirements, problem and solution analysis, design, implementation, integration, deployment (including configuration), operation, and maintenance. The list of component source types, described earlier, forms another dimension. Types of security response could form yet a third dimension, covering avoidance, deterrence, prevention, detection, mitigation, recovery, and investigation (or forensics).

Cells at the intersections of two or more dimensions represent a concern that is more specific than would be expressed by the list of classifications in any one dimension. For example, with two dimensions we can target security patterns for requirements when using COTS or for outsourced components. Similarly, we can target security patterns for analysis and design with web services, and of those, more specifically, patterns that address detection or recovery.

The design of the matrix is motivated by a notion of coverage of concerns. For security, coverage must be comprehensive. Security must be addressed in every activity of every role in every phase. Security must be addressed in every component at every level of architecture. Security must also be addressed at every transition or interface. Security must be appropriate to every language, platform, and domain. Security must be addressed in depth, with all forms of response. Security is more than a few choices – it applies to everything. We express coverage as a grid or matrix of concerns, where comprehensive coverage would mean there is something for every cell at every intersection of every dimension.

Our approach starts with a complete problem space, and then carves it into different concerns along different dimensions. Each dimension should be logically cohesive with simple, well understood distinctions over a single continuum or whole. By treating classification as a partitioning, we never loose sight of the whole. In logic, the drawing of a distinction defines both that which is included, and that which is not. In contrast, when classification starts with a known, but unstructured collection of items, and puts them into groups, there is no way to know what is missing. We view coverage of concerns as a complement to separation of concerns in top-down decomposition.

Developers can identify their current focus or concern by choosing the applicable element, or range of elements, along each dimension, and then looking for patterns that fall into the intersection of all. In this sense, it is no different than tagging patterns with identifier keys and using the tags to search. But there are several important distinctions. First, by defining tags along continua in an n-dimensional space, the developer can navigate to adjoining, and thus related, regions of the space for related patterns, added context, and deeper understanding. Second, by looking at the number of cells a pattern covers, and the region they represent, developers gain insight into not only the degree of generality, but also the type of generality the pattern entails. Third, by looking at regions and cells where “tagged” patterns are missing, pattern developers can identify gaps that have yet to be addressed.

The use of a concern matrix is not specific to any methodology. Developers in any method can use concern matrices to better identify tasks and concerns and to locate patterns relevant to a specific concern. A chosen method dictates the sequence and timing of certain tasks, while the matrix provides guidance to more specific knowledge or ideas for how to perform the tasks. This approach is consistent with, for example, McGraw’s notion of security “touch points” [19].

The matrix can be easily extended to adapt to new technologies and concerns. Entries can be added along existing dimensions, or new dimensions added entirely. In essence, there is one comprehensive matrix with all dimensions and all elements of all dimensions. All other matrices are subsets of the one, collapsed or filtered along different dimensions. Extending the matrix does not obsolete earlier matrices – they just don't include as many distinctions.

2.1 Primary Dimensions

Primary dimensions are generally useful and should be considered in any classification. We also consider secondary dimensions which identify or bring focus to additional concerns that may be useful in some cases, but need not be in focus for others. Later on, we also introduce a concept of auxiliary dimension. An auxiliary dimension presents a list of concerns that may be useful for ranking or scoring patterns, like a checklist, but do not divide the space along a continuum, for concern navigation or subdividing tasks.

Earlier, we introduced three dimensions for classifying security patterns: lifecycle stage, component source, and security response type. In our work so far, we have found six primary dimensions to be of value. The remaining three dimensions are domain, architecture layer, and level of constraint.

Application domain can provide an important differentiator or filter to narrow the field of applicable knowledge. Some solutions are specific to a particular domain or application type. Ubiquitous computing, e-commerce, and SCADA (sensing and control) all pose distinct security challenges. For example, security for 3-Tier business applications may differ from solutions for SCADA systems of sensors and controls. Organizations can create patterns for their own domain, following a template, as a form of knowledge capture. The concern matrix provides a useful and easy way to organize such patterns and position them within the context of other patterns.

Architectural layer provides another useful dimension, since problems and their solutions in different layers of the architecture differ, yet all are important. Roughly the same architecture continuum has been divided in different ways for communication protocols, business systems, and execution environments, but always with an ordering from low to high level of abstraction, and from network to platform to application. Combining elements from each, and allowing for overlap between views, we chose the following distinctions: network, transport, distribution (including gateways and brokers), operating system, data, business logic, and client. The simpler notion of an application layer spans the last three layers in our model. Since patterns can be placed in more than one cell, there is no real need for exact or disjoint classification.

Leveson defines four levels of constraint: regulatory, organizational, operator/developer, and mechanism. In Leveson's work on system safety [17], each level of constraint plays an important role in safety failures and their prevention. By extension, we use the same levels for security. The Common Criteria has functional requirements that apply at the level of mechanisms [1]. But it also has assurance requirements that concern organizational processes to document actions taken. The development of a configuration management plan is a Common Criteria assurance requirement that applies at the organizational level in the lifecycle stage of domain analysis. The Common Criteria and other standards such as SOX and SSE-CMM [29] themselves belong at the regulatory level.

2.2 Secondary and Auxiliary Dimensions

We did not include testing in the lifecycle stages since testing is an orthogonal concern that applies to all stages. But the lifecycle stages, in turn, can all be subdivided by the application of another dimension, with elements for preparation, execution, validation/verification, and transition (to another stage). This added granularity not only creates a placeholder for testing and formal verification, but, for the purpose of verifying coverage, it elevates all of the distinctions in this secondary dimension, and their concerns, within each of the software lifecycle stages.

Collections of security practices often include a list of security principles, like the principle of least privilege. Viega and McGraw [31], for example, use a list of 10 security principles, while in [28] there are 12. OWASP [21] lists 15 principles, as well as 10 secure coding principles, 20 weaknesses or vulnerabilities and 12 countermeasures. Such lists do not really divide up the problem space. But they could provide an auxiliary dimension to rank solution patterns, based on how many and which principles they apply or address.

A danger in composing point solutions occurs at the interface between components of the solution. For example, in a heterogeneous system, some parts may be .NET while others are J2EE. New code may interface to a legacy system or interchangeable web services. On a different dimension, subsystems may be formed by combining outsourced with legacy code. Unique security issues may occur at the interface, where the two interact or coexist. The matrix can be used to isolate and document interface issues by replicating an axis, orthogonal to itself. The resulting 2-dimensional slice is analogous to a mileage chart, with lists of cities on both axes. Chart elements represent interfaces between corresponding components: outsource to legacy, legacy to web service, .NET to J2EE, etc.

3 Usage Experience

We are currently applying our matrix to a classification of security patterns for Service Oriented Architecture. The results, however, are preliminary and will be presented in a later, separate paper. Here, we discuss only a few observations.

Classification is proving to be quite easy. The classifications were chosen to be easily understood. Experience bears this out. Patterns are mapped in the same way they would be tagged. Each dimension is considered separately. Patterns are identified at the point in each dimension, and thus the matrix, where their content affects decisions that will be made. If the distinctions of a dimension are not relevant, for example, if the pattern is not specific to any domain but applicable to all, then its classification on that dimension is "all."

Fig. 1 illustrates a mapping of design patterns to lifecycle phase and different levels of architecture. Only a small sample of patterns is shown. While all of the patterns in the figure are applicable to Service Oriented Architecture, some apply more generally to other domains, as well. We grouped the patterns within Design along a secondary dimension with Access Control and Authentication. A developer might navigate to adjoining Analysis phase cells (not shown) and look for general patterns on Access Control and Authentication. In the figure, we show patterns from the Domain Analysis phase, where the developer would find patterns that explain the domain standards and technologies later used in the design phase. While the patterns are found in these locations in the matrix, understanding their role in a system, and how they relate to one another, still requires a pattern language diagram and other tools and methods for pattern application.

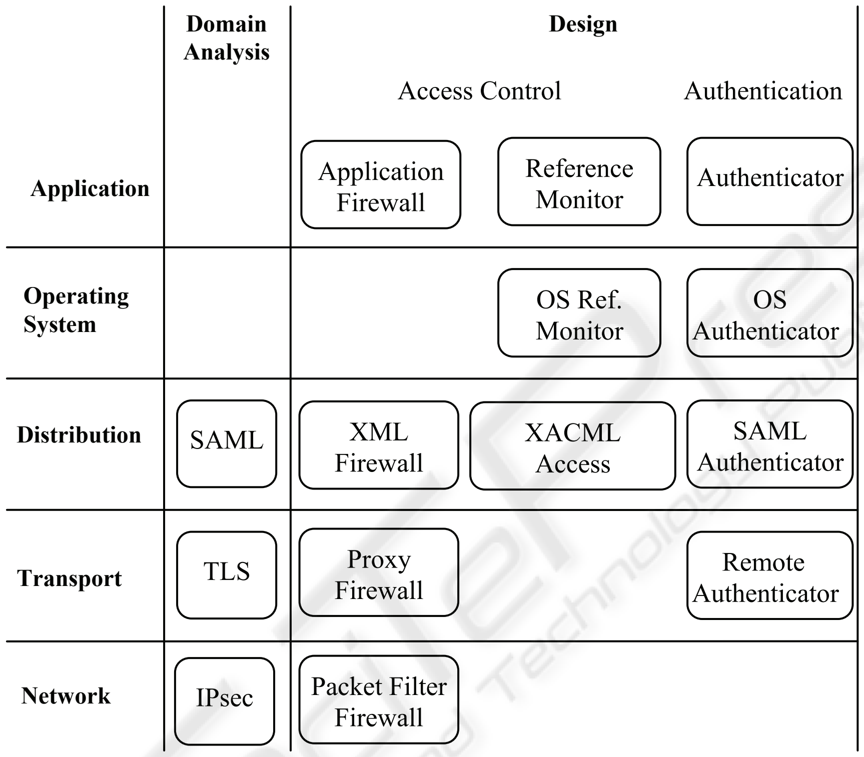

Fig. 1. A small sample showing patterns found in a 2D snippet of the matrix.

Many of the patterns we have looked at appear also in [28]. Steel et al. grouped their patterns only according to the layers in a 4-tier architecture, while we, of course, have applied more distinctions. A number of differences can be observed. By forcing each pattern to occupy only one cell, important information was lost or distorted. For example, in [28] Yoder and Barclow's Checkpoint was identified as a Client (Web) pattern, and merged with the Open Group's Checkpointed System pattern as one of a group of checkpoint patterns in the same cell. In our classification, Checkpoint is an analysis phase pattern applicable to all three application layers and addresses a general approach to prevention, while Checkpointed System is a design phase pattern applicable to the client layer and addresses a mechanism for recovery. The two patterns are quite different.

The Open Group presents a number of security patterns together with a methodology for their application [2]. But they classify their patterns into only two groups: Available Systems Patterns, and Protected Systems Patterns. Their approach assumes that the domain and problem analysis are already understood. Our matrix shows that their approach offers quite good coverage with design solutions for components and low level code, and several of the security response types. But it still leaves large parts of the space of developer concern unaddressed, particularly in early and late lifecycle phases.

Most of the security patterns we find in other work are design patterns describing mechanisms for prevention. Our own work has long been motivated by the needs of developers and the development process. This concern has driven us to pioneer a number of new pattern types and cover a larger variety of concerns. For example, our semantic analysis patterns [4] provide domain models and context, extending coverage along the lifecycle axis to domain analysis. Our standards patterns explain the security aspects of interchange languages and network infrastructure protocols [3, 7], covering more of the dimension of architectural levels. Lastly we developed forensics patterns for methods and mechanisms of attack investigation [22], covering more of the dimension for types of security response.

4 Related Work

Lists are often used in security. DoD and NIST maintain lists of checklists for securely configuring various software applications, while CERT and NIST list 24,000+ known software exploits. The Common Criteria [1] and SSE-CMM (ISO/IEC 21827) [sse] both include lists of assurance areas that must be documented to satisfy certification. Hoglund and McGraw list 49 types of software attacks [14]. Microsoft has produced lists of flaws, attack trees, and a secure development process [15, 16, 18]. In contrast to these long lists of heterogeneous concerns, each dimension of our matrix is comparatively easy to understand with generally recognizable partitions.

Schumacher has defined a methodology for secure systems design using security patterns [25, 26]. He analyzed social and technical weaknesses impeding the development of secure systems and proposed applying security at all stages of the software lifecycle. He originally proposed a knowledge management approach involving a Security Improvement Feedback Loop and a vulnerability database to keep track of possible attacks and countermeasures. In our experience, vulnerability databases focus largely on low level code exploits to be addressed in implementation. We believe that by providing a rich framework for organizing and reasoning about patterns, the same model of improvement and knowledge feedback can be applied at a higher level, addressing weaknesses at a system level.

Trowbridge et al. [30] describe an “organizing table” to organize patterns and identify gaps, and includes a discussion of identifying relationships by exploring adjacent cells. But they limited their classification to two heterogeneous dimensions for viewpoints and interrogatives. Their 5 viewpoints were business, integration, application, operation, and development, each then subdivided into architect, designer, and developer. The 110 patterns classified fell into only two categories: integration and application, and three concerns: data, function, and network. Their viewpoints roughly correspond to our lifecycle stages, though the distinctions are less clear. Without a method of connecting classification to the whole, identification of gaps can only be ad hoc.

Hafiz et al. [12] identify four potential classification dimensions: protection type, application context, threat, and Trowbridge's viewpoints. In the end they proposed a hierarchy based on application context (core, perimeter, exterior), followed by threat. Developer contexts like lifecycle stage, domain, and component source, and levels of response are not considered. The collector perspective was apparent in their effort to uniquely place each pattern and their expressed concern that too many patterns fell in too few cells. In the paper, they state, "Any organization effort must begin by collecting the items to be organized." For our purposes, we reject this notion, and begin with the space to be covered.

E.B. Fernandez et al. also discuss classifying patterns for designers [8]. The paper describes the importance of architecture layer and how to use architectural layer classifications and concerns to make systems more secure. For example, access control can be defined in the application and reflected to the database and operating system layers. The paper also discusses pattern diagrams and their important relationship to classifications. The ideas in that paper complement those presented here.

The work of German and Cowan [9] classify user interface patterns in a deep hierarchy of domain, phase, elaboration, general to specific, and value (a kind of Google ranking). Their ordered elaboration categories make sense for user interface design, but are unrelated to security. Issues in that paper make it clear that our matrix would have to be extended if we wanted to classify patterns in domains other than security.

It should be noted that a multidimensional space can be aligned with cell divisions in Trowbridge, Hafiz, and even German and Cowen without hierarchical grouping. From the users' perspective, reducing orthogonal classifications to a single hierarchy achieves little and hinders the exploration of relationships along different dimensions. Moreover, if patterns are meaningful in multiple places, then they should be found in multiple places.

5 Conclusions

In the past, we proposed patterns-based methodologies [5]. In this paper we propose a method of organizing patterns to address pattern coverage, and facilitate pattern discovery, exploration, and navigation by the application developer. The approach uses a multi-dimensional matrix where each dimension divides the problem space into generally understood concerns. The combination of concerns from multiple dimensions allows patterns to be precisely associated with regions of applicability, and supports developer navigation to find related patterns with relevant information.

Although we have only begun to use this tool, preliminary results are promising. The matrix is proving quite easy to use, and provides insights relevant to the understanding and selection of patterns not found in other classification schemes. Some examples are presented from our work with security patterns for Service Oriented Architecture.

Acknowledgements

We wish to acknowledge Paul Black of NIST for his participation in the initial discussion of using a matrix of concerns for developing secure systems.

References

1. Common Criteria, http://www.commoncriteriaportal.org/
2. Blakley, B., Heath, C., members of The Open Group Security Forum: Technical Guide: Security Design Patterns. The Open Group, UK, April 2004.
3. Delessy, N., Fernandez, E.B.: Patterns for the eXtensible Access Control Markup Language. Proc. 12th Pattern Languages of Programs Conference, Monticello, Illinois, USA, (2005) http://hillside.net/plop/2005/proceedings/
4. Fernandez, E.B., Yuan, X.: Semantic Analysis Patterns. Proc. 19th Int. Conf. on Conceptual Modeling (2000), 183-195 http://www.cse.fau.edu/~ed/SAPpaper2.pdf
5. Fernandez, E.B., Larrondo-Petrie, M.M., Sorgente, T., VanHilst, M.: A Methodology to Develop Secure Systems Using Patterns. In: Mouratidis, H., Giorgini, P. (Eds.): Integrating Security and Software Engineering: Advances and Future Vision. IDEA Press (2006) 107-126
6. Fernandez, E.B., VanHilst, M., Larrondo Petrie, M.M., Huang, S.: Defining Security Requirements through Misuse Actions. In: Ochoa, S.F., Roman, G.-C. (Eds.): Advanced Software Engineering: Expanding the Frontiers of Software Technology, International Federation for Information Processing, Springer (2006) 123-137
7. Fernandez, E.B., VanHilst, M., Pelaez, J.C.: Patterns for WiMax Security. Proc. EuroPLoP (2007) http://hillside.net/europlop/home.html
8. Fernandez, E.B., Washizaki, H., Yoshioka, N., Kubo, A., Fukazawa, Y.: Classifying Security Patterns. Proc. 10th Asia-Pacific Web Conference, Shenyang, China, April 26-28 (2008)
9. German D., Cowan, D.: Towards a Unified Catalog of Hypermedia Design Patterns. Proc. 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, (2000)
10. Federal Information Security Management Act (FISMA), March 18, 2007, http://iase.disa.mil/fisma/index.html
11. Senate Banking Committee: Gramm-Leach-Bliley Act, Monday, November 1 (1999) http://www.senate.gov/~banking/conf/fincon.pdf
12. Hafiz, M., Adamczyk, P., Johnson, R.E.: Organizing Security Patterns. IEEE Software, 24(4), July/August (2007) 52-60
13. United States Department of Health and Human Services, Office of Civil Rights: Health Insurance Portability and Accountability Act of 1996. http://www.hhs.gov/ocr/hipaa/
14. Hoglan, G., McGraw, G.: Exploiting Software: How to Break Code. Addison-Wesley (2004)
15. Howard, M., LeBlanc, D.: Writing Secure Code, (2nd Ed.). Microsoft Press (2003)
16. Howard, M., Lipner, S.: The Security Development Lifecycle. Microsoft Press (2006)
17. Leveson, N.: A New Accident Model for Engineering Safer Systems. Safety Science, 42(4), April (2004) 237-270
18. Lipner, S., Howard, M.: The Trustworthy Computing Development Lifecycle, http://msdn2.microsoft.com/en-us/library/ms995349.aspx, March (2005)
19. McGraw, G.: Software Security: Building Security. Addison-Wesley (2006)

20. Nagaratnam, N., Nadalin, A., Hondo, M., McIntosh, M., Austel, P.: Business-Driven Application Security: from Modeling to Managing Secure Applications. IBM Systems Journal, 44(4) (2005) 847-867
21. The OWASP Testing Project. http://www.modsecurity.org/archive/OWASPTesting_PhaseOne.pdf
22. Pelaez, J.C., Fernandez, E.B.: Network Forensics in Wireless VoIP Networks, Proc. 4th Latin American and Caribbean Conference for Engineering and Technology. Mayaguez, Puerto Rico, (2006)
23. Pelaez, J.C., Fernandez, E.B., Larrondo-Petrie, M.M., Wieser, C.: Attack Patterns in VoIP. Proc. 14th Pattern Languages of Programs Conference, Monticello, Illinois, USA, (2007)
24. One Hundred Seventh Congress of the United States of America: Sarbanes-Oxley Act of 2002. http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf
25. Schumacher, M., Ackermann, R., Steinmetz, R.: Towards Security at All Stages of a System's Life Cycle. Proc. Int. Conf. on Software, Telecommunications, and Computer Networks (2000) 11-19
26. Schumacher, M., Roedig, U.: Security Engineering with Patterns. Proc. 8th Pattern Languages of Programs Conference (2001)
27. Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns: Integrating Security and Systems Engineering. Wiley (2006)
28. Steel, C., Nagappan, R., Lai, R.: Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management. Prentice Hall (2005)
29. Systems Security Engineering – Capability Maturity Model, http://www.sse-cmm.org
30. Trowbridge, D., Cunningham, W., Evans, M., Brader, L., Describing the Enterprise Architectural Space. MSDN (2004) http://msdn2.microsoft.com/en-us/library/ms978655.aspx
31. Viega, J., McGraw, G.: Building Secure Software: How to Avoid Security Problems the Right Way. Addison-Wesley (2001)