A M -D s o a C ass a o o U s o S Pa s

M a V H s a E a o B. F a

Florida A lan ic Univer i y, Depar men of Compu er Science and Engineering
777 Glade Road, Boca Ra on, Florida 33431, USA
{mike, ed @c e.fau.edu
h p://www.c e.fau.edu/ {mike,ed

F o B a

Univer idade de Bra lia, Depar amen o de Engenharia El rica
Campu Univer i rio A a Nor e, 70910-900 Bra lia DF Bra il
fabraz@unb.br

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 by 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.

ACM Classification: D.2.1., K.6.3., K.6.5.

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, and 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, open-source, runtime script, model transformation, wizard code generation, legacy application, reuse library, outsourced development, commercial-off-the-shelf, and remote web service. 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 (Federal Information Security Management Act, 2007), HIPAA (United States Department of Health & Human Services, 2003), Sarbanes/Oxley (One Hundred Seventh Congress of the United States of

Manuscript received: 23 April 2008
Communicating Editor: Eduardo Fernandez-Medina Paton

America, 2002), and Graham-Leach-Bliley (Federal Trade Commission, 1999). 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.

Our group addresses 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 (Fernandez and Yuan, 2000; Fernandez, Larrondo-Petrie, Sorgente and VanHilst, 2006; Palaez, Fernandez, and Larrondo-Petrie, 2007).

Other groups also develop security patterns (Blakeley, Heath and members of the Open Group Security Forum, 2004; Schumacher, Fernandez, Hybertson, Buschmann and Sommerlad, 2006; Steel, Nagappan and Lai, 2005), and many books on security also package material in the form of patterns. 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 (Fernandez, Washizaki, Yoshioka, Kubo and Fukazawa, 2008; German and Cowan, 2000; Hafiz, Adamczyk and Johnson, 2007; Trowbridge, Cunningham, Evans and Brader, 2004), and based on the solution being presented rather the problems being addressed. 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 organized 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 pattern classification and problem coverage through the use of a multi-dimensional matrix of concerns. Each dimension of the matrix is a distinct list of concerns along a single axis, with a simple concept and a set of distinctions that define the categories. The categories along an axis or dimension should be easily understood and represent widely used and accepted classifications with respect to that concept. For example, one dimension would be a list of life-cycle activities, covering domain analysis, requirements, problem analysis, design, implementation, integration, deployment (including configuration), operation, maintenance and disposal. 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. Information is not secure if it can be compromised at any point in any way. 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 extend to every language, platform, and domain. Security must be addressed in depth, with all forms of response. 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. Thus we are concerned not only with patterns that exist, but also to identify gaps where patterns do not exist.

Our approach starts with a complete problem space, and then carves it into different concerns along different dimensions. The idea of dividing up psychological space can be traced back to Euclid's elements. Its use here builds on the ideas of George Kelly (Kelly, 1955). In Kelly's personal construct theory, a construct is a reference axis of two opposing poles. Wealth, for example is an axis of rich and poor. The space between the poles defines a "range of convenience" which gains further relevance with additional planes of distinction. "A construct is a dichotomous reference axis. It defines a family of planes orthogonal to it that divide the space" (Shaw and Gaines, 1992). In our case, we are not as interested in the planes of distinction, which create the separations, as with spaces between two planes, which provide a convenience of classification. Kelly described a matrix of concepts that embodies a person's intentions and shapes their response. He called it a "role repertory grid." A more formal treatment of the division of psychological space can also be found in Brown (1971) "Laws of Form."

Here we report on our initial effort to define a matrix of concepts to embody the intention to secure information, and to use that matrix to classify patterns. In defining dimensions or axes for classification, each axis should correspond to a single logical construct. In Kelly's model, a construct is defined by dichotomous poles. We strive to do the same. Each axis or dimension is further divided into regions of concern. The regions can be loosely defined and be hierarchical, disjoint or overlap. Regions or classifications of concern within a dimension should be based on distinctions that are easily or well understood by target users – in our case, computer scientists and security practitioners. Defining the regions is much like defining concerns in top-down decomposition. In logic, a distinction defines both that which is included, and that which is not (Brown, 1971). 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.

The target users of this work cover a range of security stake holders. In addition to software developers, we also consider the needs and perspectives of information security managers with more of a systems perspective, patterns writers looking for gaps to fill and a broader context within which to view their patterns, and students looking to broaden their understanding of a range of security issues and perspectives.

Developers using the grid can identify their current focus or concern by choosing the applicable element, or range of elements, along each dimension, and then look 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 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" (McGraw, 2006).

The matrix can be easily extended to adapt to new technologies and concerns, or focus on a particular need. Entries can be added along existing dimensions, or new dimensions added entirely. Extending the matrix does not obsolete earlier matrices – they just don't include as many distinctions.

3. PRIMAR DIMENSIONS

Primary dimensions are generally useful and should be considered in any classification. 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 it useful to include three more primary dimensions: architecture layer, constraint level, and domain. These six dimensions are shown graphically in Figure 1. The "all" classification is used when the distinctions along that axis are not meaningful to a pattern. 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.

Classifications on the axis for lifecycle stage are ordered on the dichotomy of beginning and end. It actually starts with a pre-beginning or potential, where domain analysis falls, and could end post-end with disposal. The categories of component source are organized around a dichotomy of internal vs. external control. In new code, the developer has complete control over all details. With a remote service, there is little or no control or even knowledge of details, and limited ability to test. The response axis based on whether or not and attack happens and the extent, from not happening at all (avoidance), to completely happened and in the past (forensics).

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. We fit these concepts on an axis from meaningless bits to end-task semantics. Combining elements from several domains, and allowing for overlap between views, we chose the following distinctions: network, transport, distribution (including gateways and brokers), platform and operating system, data, business logic, and client. A simpler notion of application spans the last three. Network, transport, and distribution may also be grouped as communication. Since patterns can be placed in more than one cell, there is no real need for exact or disjoint classification.

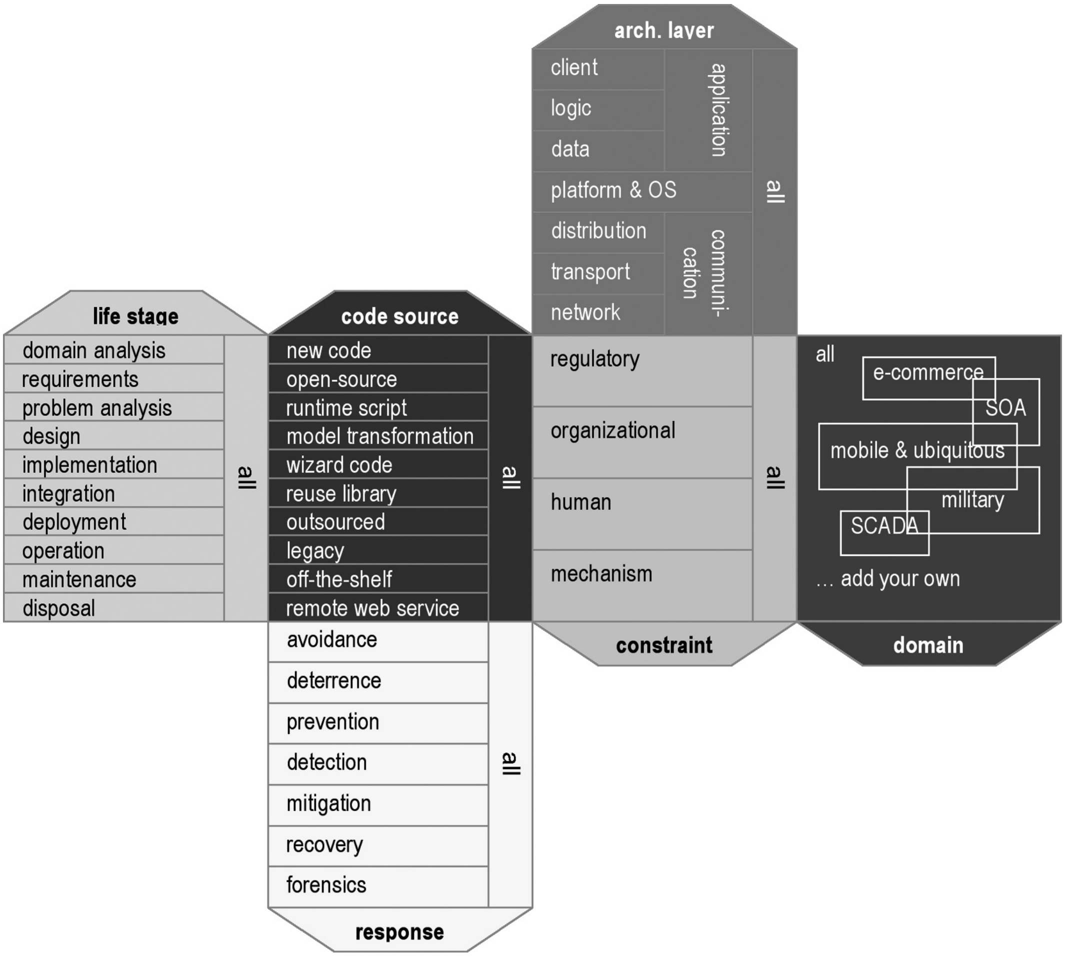


Figure 1: Six primary dimensions of classification (shown as a pseudo-hypercube)

Leveson defines four levels of constraint: mechanism, human (operator or developer), organizational, and regulatory. In Leveson's work on system safety (Leveson, 2004), each level of constraint plays an important role in safety failures and their prevention. By extension, we use the same levels for security with an axis with levels from thing to society. While most security patterns describe mechanisms, the National Training Standard for Information Systems Security (INFOSEC) Professionals (National Security Agency, 1994) is mostly concerned with practices, policies, and regulations. The Common Criteria has functional requirements that apply at the level of mechanisms (Common Criteria Sponsoring Organizations, 2006). 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 (Systems Security Engineering - Capability Maturity Model, 2003) themselves belong at the regulatory level.

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 (Supervisory Control and Data Acquisition) all pose distinct security challenges. For example, security for 3-Tier business applications may differ from solutions for SCADA systems of sensors and controls. This axis is an exception in that it does not have a dichotomy or ordering – the space is freely defined. Organizations can create patterns for their own domain as a form of knowledge capture.

4. SECONDAR AND AU◀ILIAR 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). The added granularity of this secondary dimension not only creates a placeholder for testing and formal verification, but, for purposes of verifying coverage, it elevates all of the distinctions in this dimension, and their concerns, in 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 (2001), for example, use a list of 10 security principles, while in Steel *et al* (2005) there are 12. OWASP (Open Web Security Application Project, 2008) 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.

5. USAGE

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 initial observations.

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 "any" or "all." As an example, the anti-patterns in Kis (2006) apply at the developer level of constraint in the requirements phase. Some patterns refer to legacy components, while component source is not relevant to others. In a test with different members of our team independently classifying the same six patterns from the Open Group, and Anwar, Yurcik, Johnson, Hafiz, and Campbell (2006), the results showed only minor differences in five of the primary axes. Domain was the exception where it proved harder to assign ranges of applicability.

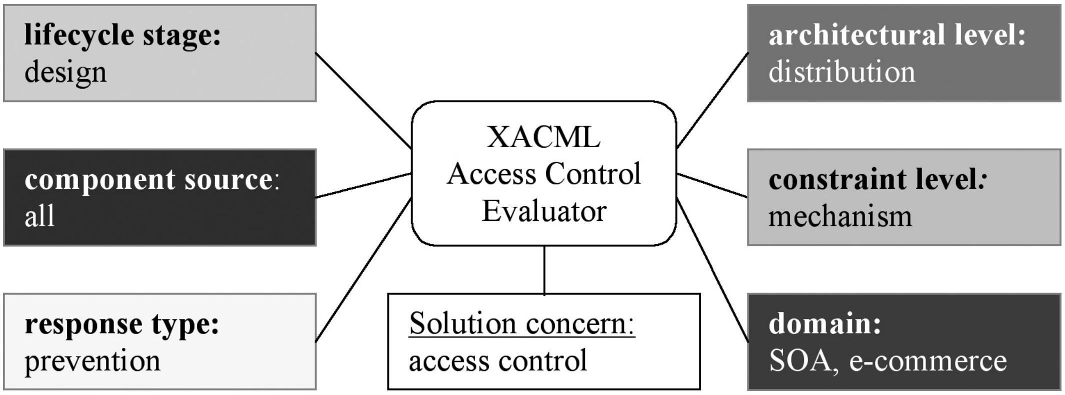


Figure 2: Classifications of the XACML Access Control Evaluator

Figure 2 shows the classifications of the XACML Access Control Evaluator pattern. The pattern describes a mechanism that is applied in the design stage of development. The type of response is prevention since the attack happens and is turned away. The control is used in the distribution level of the system architecture. Component source is not relevant as it is applicable to all classifications

	Domain Analysis	**Design**		
		Filtering	Access Control	Authentication
Application		Application Firewall	Reference Monitor	Authenticator
Operating System			OS Ref. Monitor	OS Authenticator
Distribution	SAML XACML	XML Firewall	XACML Access Control Evaluator	SAML Authenticator
Transport	TLS	Proxy Firewall		Remote Authenticator
Network	IPsec	Packet Filter Firewall		

Figure 3: A sample of patterns in a focused and flattened (2.5D) snippet of the matrix.

along this axis. The pattern is specific to the domain of e-commerce and Service Oriented Architecture, where XACML is used. As it describes a component of the solution architecture, a classification on an additional axis for solution concern is also useful.

Figure 3 illustrates a mapping of design patterns in two lifecycle phases and at 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 Filtering, 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. A developer might also navigate to adjoining Analysis phase cells (not shown) to look for general patterns on Filtering, Access Control and Authentication. 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.

Many of the patterns we have looked at appear also in Steel *et al* (2005). Steel *et al* grouped their patterns only according to the layers in a 4-tier architecture, while we, 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 Steel *et al* (2005) Yoder and Barcalow'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.

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. As a result, our own patterns cover a larger variety of concerns. For example, our semantic analysis patterns (Fernandez and Yuan, 2000) 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 (Delessy and Fernandez, 2005; Fernandez, VanHilst, and Palaez, 2007), covering more of the dimension of architectural levels. Lastly we developed forensics patterns for methods and mechanisms of attack investigation (Pelaez, J. and Fernandez, 2006), covering more of the dimension for types of security response.

In this work, we do not specifically address issues of visualization. The usual methods of reducing multi-dimensional information such as narrowing, flattening, and projection – as in spreadsheets and OLAP (online analytical processing) – can be applied. In their work on dividing the enterprise architectural space (Trowbridge *et al*, 2004), members of Microsoft's Patterns and Practices group constrained themselves to two dimensions because of the visualization issue. We are more concerned with coverage of the problem space and about the potential for loss of information – especially loss of information about gaps.

6. RELATED ORK

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 (Common Criteria Sponsoring Organizations, 2006) and SSE-CMM (ISO/IEC 21827) (Systems Security Engineering – Capability Maturity Model, 2003) both include lists of assurance areas that must be documented to satisfy certification. Hoglund and McGraw (2004) list 49 types of software attacks. Microsoft has produced lists of flaws, attack trees, and a secure development process (Howard and LeBlanc, 2003; Howard and Lipner, 2006; Lipner and Howard, 2005). In contrast to these long lists of heterogeneous concerns, each dimension of our matrix covers a more cohesive and concise range of concepts with generally recognizable partitions.

Schumacher has defined a methodology for secure systems design using security patterns (Schumacher, Ackermann and Steinmetz, 2000; Schumacher *et al*, 2006). Like us, they also propose applying security at all stages of the software lifecycle. They propose using a vulnerability database to keep track of possible attacks and countermeasures. But they do not provide details on how to apply security to all the development stages, nor how to verify coverage of concerns.

Trowbridge *et al* (2004) describe an "organizing table" to organize patterns and identify gaps, and include a discussion of identifying relationships by exploring adjacent cells. But they limit their classification to two heterogeneous dimensions for viewpoints and interrogatives. Their five viewpoints are business, integration, application, operation, and development, each then subdivided into architect, designer, and developer. The 110 patterns classified fall into only two categories: integration and application, and three concerns: data, function, and network.

Hafiz *et al* (2007) 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 followed by threat. The collector perspective was apparent in their effort to uniquely place each pattern and their 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." We reject this notion, and begin with the space to be covered.

Fernandez *et al* (2008) classify patterns based on architectural levels and concerns. For example, access control can be defined in the application and reflected to the database and to the operating system. Architecture levels and security concerns are two possible dimensions in the proposed matrix.

German and Cowan (2000) 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 not related to security. It is clear from their example 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 *et al* (2004), Hafiz *et al* (2007), and even German and Cowen, (2000) 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.

7. CONCLUSION

In the past, we proposed patterns-based methodologies (Fernandez *et al*, 2006). 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. This organization is useful for any development methodology.

Although we have only begun to use this tool, preliminary results are promising. Some examples were presented from our work with security patterns for Service Oriented Architecture.

8. ACKNO LEDGEMENTS

This work is partially supported by a grant from the US Department of Defense' Defense Information Systems Agency. We wish to acknowledge Paul Black of NIST for his contribution to the initial discussion of using a matrix of concerns for developing secure systems.

REFERENCES

ANWAR, Z., YURCIK, W., JOHNSON, R.E., HAFIZ, M. and CAMPBELL, R.H. (2006): Multiple design patterns for voice over IP (VoIP) security. In *Proceedings of the IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2006. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1629443

BLAKELEY, B., HEATH, C. and MEMBERS OF THE OPEN GROUP SECURITY FORUM (2004): Technical guide: Security design patterns http://www.opengroup.org/bookstore/catalog/g031.htm

BROWN, G.S. (1971): *Laws of Form*, George Allen & Unwin.

COMMON CRITERIA SPONSORING ORGANIZATIONS (2006): *Common Criteria for Information Technology Security Evaluation Part 2: Security Functional Components, Version 3.1 Rev 1.*

DELESSY, N. and FERNANDEZ, E.B. (2005): Patterns for the eXtensible access control markup language. In *Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2005),* Monticello, Illinois, USA, 7-10. http://hillside.net/plop/2005/proceedings/

FEDERAL INFORMATION SECURITY MANAGEMENT ACT (2007): March 18. http://iase.disa.mil/fisma/index.html.

FEDERAL TRADE COMMISSION (1999): Gramm-Leach-Bliley Act. http://www.ftc.gov/privacy/glbact/glbsub1.htm

FERNANDEZ, E.B. and YUAN, X. (2000): Semantic analysis patterns. In *Proceedings of the 19th Int. Conf. on Conceptual Modeling, ER2000*, 183-195. Also available from: http://www.cse.fau.edu/~ed/SAPpaper2.pdf

FERNANDEZ, E.B., LARRONDO-PETRIE, M.M., SORGENTE, T. and VANHILST, M. (2006): A methodology to develop secure systems using patterns. Chapter 5 in *Integrating security and software engineering: Advances and future vision.* MOURATIDIS, H. and GIORGINI, P. (eds.). IDEA Press, 2006, 107-126.

FERNANDEZ, E.B., VANHILST, M. and PELAEZ, J.C. (2007): Patterns for WiMax security. In *Proceedings of EuroPLoP*. http://hillside.net/europlop/home.html

FERNANDEZ, E.B., WASHIZAKI, H., YOSHIOKA, N., KUBO, A. and FUKAZAWA, Y. (2008): Classifying security patterns. In *Proceedings s. of the 10th Asia-Pacific Web Conference (APWEB'08),* Springer LNCS 4976/2008, 342-347. http://www.neu.edu.cn/apweb08/

GERMAN, D. and COWAN, D. (2000): Towards a unified catalog of hypermedia design patterns. In *Proceedings of 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii.

HAFIZ, M., ADAMCZYK, P. and JOHNSON, R.E. (2007): Organizing security patterns. *IEEE Software* 24(4):52-60.

HOGLAND, G. and MCGRAW, G. (2004): *Exploiting Software: How to Break Code*, Addison-Wesley.

HOWARD, M. and LEBLANC, D. (2003): *Writing secure code*. (2nd Ed.), Microsoft Press.

HOWARD, M. and LIPNER, S.(2006): *The security development lifecycle*. Microsoft Press.

KELLY, G.A. (1955): *The Psychology of Personal Constructs*, Norton.

KIS, M. (2002): Information security antipatterns in software requirements engineering. In Proceedings of the 9th Pattern Languages of Programs Conference (PLoP2002). http://jerry.cs.uiuc.edu/%7Eplop/plop2002/final/mkis_plop_2002.pdf

LEVESON, N. (2004): A new accident model for engineering safer systems. *Safety Science* 42(4):237-270.

LIPNER, S. and HOWARD, M. (2005): The trustworthy computing development lifecycle. http://msdn2.microsoft.com/en-us/library/ms995349.aspx

MCGRAW, G. (2006): *Software Security: Building Security In*, Addison-Wesley.

NATIONAL SECURITY AGENCY (1994): *NSTISSI-4011, National Training Standard for Information Systems Security (INFOSEC) Professionals*. http://www.nsa.gov/ia/academia/cnsstesstandards.cfm

ONE HUNDRED SEVENTH CONGRESS OF THE UNITED STATES OF AMERICA (2002): Sarbanes-Oxley Act of 2002. http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf

OPEN WEB SECURITY APPLICATION PROJECT (2004): The OWASP testing project.http://www.modsecurity.org/archive/OWASPTesting_PhaseOne.pdf

PELAEZ, J. and FERNANDEZ, E.B. (2006): Network forensics in wireless VoIP networks. In *Proceedings s. of the 4th Latin American and Caribbean Conference for Engineering and Technology (LACCEI 2006)*. Mayaguez, Puerto Rico, June 21-23.

PELAEZ, J., FERNANDEZ, E.B., LARRONDO-PETRIE, M.M. and WIESER, C. (2007): Attack patterns in VoIP. *Proceedings of the 14th Pattern Languages of Programs Conference (PLoP2007)*. http://hillside.net/plop/2007/index.php?nav=program.

SCHUMACHER, M., ACKERMANN, R. and STEINMETZ, R. (2000): Towards security at all stages of a system's life cycle. In *Proceedings of Int. Conf. on Software, Telecommunications, and Computer Networks (Softcom)*. 11-19. http://www.ito.tu-darmstadt.de/publs

SCHUMACHER, M., FERNANDEZ, E.B., HYBERTSON, D., BUSCHMANN, F. and SOMMERLAD, P. (2006): *Security Patterns: Integrating security and systems engineering*. Wiley.

SHAW, M.L.G. and GAINES, B.R. (1992): Kelly's "Geometry of psychological space" and Its Significance for Cognitive Modeling, The New Psychologist, 23-31, October, 1992 http://citeseer.ist.psu.edu/11830.html

STEEL, C., NAGAPPAN, R. and LAI, R. (2005): *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall.

SYSTEMS SECURITY ENGINEERING – CAPABILITY MATURITY MODEL (2003): http://www.sse-cmm.org

TROWBRIDGE, D., CUNNINGHAM, W., EVANS, M. and BRADER, L. (2004): Describing the enterprise architectural space. MSDN. http://msdn2.microsoft.com/en-us/library/ms978655.aspx

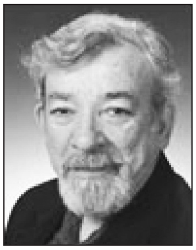
UNITED STATES DEPARTMENT OF HEALTH & HUMAN SERVICES (2003): Summary of the HIPAA privacy rule.

VIEGA, J. and McGRAW, G. (2001): *Building secure software: How to avoid security problems the right way*. Addison-Wesley.

BIOGRAPHICAL NOTES

Michael VanHil .

Michael VanHilst is an assistant professor in the Department of Computer Science and Engineering at Florida Atlantic University. His research interests include software development methods, process improvement, and computer security. He has 20 years of industry experience including for HP, IBM, and NASA. He was recently VP of Technology for a firewall development company. Dr. VanHilst received his PhD in Computer Science from the University of Washington, and holds three prior degrees from MIT. He is a member of the ACM and IEEE Computer Society.


Eduardo B. Fernandez

Eduardo B. Fernandez (Eduardo Fernandez-Buglioni) is a professor in the Department of Computer Science and Engineering at Florida Atlantic University. He has published numerous papers and four books, the most recent on security patterns. He has lectured all over the world at both academic and industrial meetings, and is an active consultant for insdustry. His current interests include security patterns and security and fault tolerance for web services. He holds an MS in Electrical Engineering from Purdue University and a Ph.D. in Computer Science from UCLA. He is a Member of the ACM and IEEE. More details can be found at http://www.cse.fau.edu/~ed

Fabricio Braz

Fabricio Braz is a PhD student in the Department of Electrical Engineering of the University of Brasilia (Brazil). He participates in the Security System Engineering Group at Florida Atlantic University. He is also a member of the Centre for Research in Information Architecture (CPIA) at the University of Brasilia. His research activities include security requirements engineering and security system design using patterns.