Managing Security Requirements Patterns using Feature Diagram Hierarchies

Rocky Slavin[1], Jean-Michel Lehker[1], Jianwei Niu[1], Travis D. Breaux[2]

Department of Computer Science[1]
University of Texas at San Antonio
San Antonio, Texas, USA
rocky.l.slavin@ieee.org, rpl599@my.utsa.edu,
jianwei.niu@utsa.edu

Institute for Software Research[2]
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
breaux@cs.cmu.edu

***Abstract*—Security requirements patterns represent reusable security practices that software engineers can apply to improve security in their system. Reusing best practices that others have employed could have a number of benefits, such as decreasing the time spent in the requirements elicitation process or improving the quality of the product by reducing product failure risk. Pattern selection can be difficult due to the diversity of applicable patterns from which an analyst has to choose. The challenge is that identifying the most appropriate pattern for a situation can be cumbersome and time-consuming. We propose a new method that combines an inquiry-cycle based approach with the feature diagram notation to review only relevant patterns and quickly select the most appropriate patterns for the situation. Similar to patterns themselves, our approach captures expert knowledge to relate patterns based on decisions made by the pattern user. The resulting pattern hierarchies allow users to be guided through these decisions by questions, which introduce related patterns in order to help the pattern user select the most appropriate patterns for their situation, thus resulting in better requirement generation. We evaluate our approach using access control patterns in a pattern user study.**

***Index Terms*—Security, requirements, patterns, feature diagram.**

I. Introduction

A *requirements pattern* is a structure that engineers can use to generate one or more requirements for a recurring situation [1]. Based on design patterns [2], each requirements pattern describes a recurring problem as well as a core solution which can be used repeatedly, but not necessarily in the same way every time. Because situations vary, a requirements pattern must be parameterized to selectively control for those effects that vary across problem spaces. For example, an engineer may choose between single sign-on [3], where users need only one username and password for logging into different systems, as means to maximize usability, where they prefer more complex role-based access control (RBAC) [4] requirements to maximize confidentiality by compartmentalizing access to information. In this example, the engineer perceives constraints differently and desires different levels of control: more control over different systems to manage logins, or more control over classification of resources into roles.

Requirements patterns incorporate engineering knowledge into the solution space of a pattern in order to provide a basis for requirements elicitation and generation. For example, patterns can itemize pre-conditions to indicate when a pattern applies to a given scenario, or questions whose answers direct the engineer from one solution to another (e.g., from single sign-on to RBAC). Such knowledge reuse allows software engineers to solve problems in a more effective manner. That is, patterns consist of tried-and-tested solutions that have been shown to be effective when applied in the correct context. Consequently, patterns serve as a common language for software engineers to document their design decisions [5].

Security requirements patterns are a special case of requirements that address security risks in a system. Historically, security is dealt with using a penetrate-and-patch approach [6], wherein security problems are identified and addressed in response to penetration testing of the fully-functional system, sometimes in a post-deployment situation. If the system failures are intrinsic to the design, then significant rework is required [7]. Alternatively, it is more cost effective to identify security flaws early in the requirements and design stages of development, which is the focus and intent of security requirements patterns [8].

In the wild, security requirements patterns appear mostly in isolation, either in small sets of related patterns or repositories and related only by a common topic or theme [9, 10]. Combined with the lack of guidance for pattern selection [11], engineers lack the structure needed to holistically address security and balance complex forces or quality attributes (e.g., usability and confidentiality). As the number of security patterns continues to grow [12], engineers face an increased challenge in recognizing what patterns to select [13]. The contribution of our approach is as follows: (1) we propose a new security requirements pattern format that aims to organize requirements knowledge into a canonical form that itemizes this knowledge into inter-dependent questions that investigate the problem space; (2) we demonstrate that these patterns can be linked into a hierarchy to make problem space trade-offs more salient and to connect related patterns to comprise holistic solution spaces; and (3) we evaluate this new format and hierarchy in a user study to measure speed and correctness in selecting patterns to apply to example scenarios.

We evaluate our approach in a user study consisting of graduate and undergraduate students at the University of Texas at San Antonio (UTSA). The study examined our method's ability to deduce the most appropriate set of patterns by having novice users with limited security knowledge and experience use a pattern hierarchy to select patterns for a scenario. We then compared the results with selections that experts chose for the same scenario. Based on our analysis, we found that novice users not only were more accurate in their pattern selections, but were able to select patterns more quickly than users without a hierarchy. From these results and further observations we assert that the use of pattern hierarchies can improve the usability of security requirements by providing a means for easier access and faster selection as well as better documentation of related patterns.

The remainder of this paper is organized as follows: in Section II, we review the theoretical foundations and background upon which we based our approach; in Section III, introduce our requirements pattern template and in Section IV, we present our pattern hierarchy; in Section V, we present our empirical study design with our results and observations presented in Section VI; finally, we present related work in Section VII, with future work in Section VIII and our conclusion in Section IX.

II. Theoretical Foundations

Our approach is based on prior work in pattern languages, feature diagrams and the requirements elicitation, which we now discuss.

A. *Pattern Languages*

Alexander et al. introduced the earliest notion of pattern as a structured device for reusing knowledge in seminal work on building architecture patterns [2]. More recently, there has been substantial work on object-oriented design patterns [14], requirements patterns [9, 15] and security patterns [10, 12, 16]. A security requirements pattern provides a software engineer with a reusable set of requirements to solve common security problems. To be effective, a requirements pattern should incorporate an engineer's knowledge about their system context to select the most appropriate requirement. For example, many systems use passwords to restrict access to sensitive resources (e.g., healthcare data, bank account data, and purchase histories) as well as to associate unique identifiers with individual users (e.g., forums or social networking sites). The risk to individual users of having their password retrieved through a security attack is different depending on the harms of a data breach. A breach to a bank account could be more expensive than the cost of a breach to web forum, where the attacker could at best post derogatory comments. Therefore, an engineer needs a means to tailor reusable knowledge to their situation to yield requirements that balance complex and sometimes competing forces (e.g., performance, security, usability, etc.). Despite this need, current security patterns are not configurable or linkable to enable engineers to tailor security requirements to their needs.

B. *Feature Diagrams*

A *feature diagram* is a graphical notation to describe how products can be composed from multiple features [17]. A *feature* is an increment of functionality, usually with a coherent purpose [18]. Features may be linked together in a feature diagram to describe common and variable requirements in a system composition [19]. The links between features may be mandatory or optional, and groups of features can be organized using logical connectives such as inclusive or exclusive or. Figure 1 shows an example including the most common notations used in a feature diagram as well as their meanings.

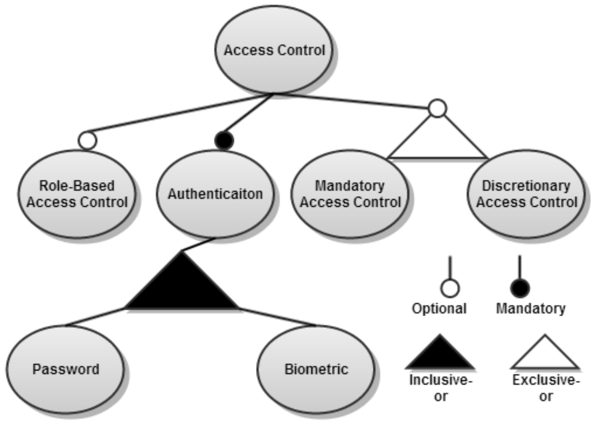

Fig. 1. Feature diagram notation.

Figure 1 describes an example. If we were to describe access control requirements, a mandatory feature would be authentication [12]. An optional feature could be the inclusion of a role-based access control policy. We can further break down some features into groups of features. Here, we can make a decision between the optional mandatory access control (MAC) [20] and discretionary access control (DAC) [21]. We could choose one or the other, but not both, denoted by exclusive-or. Similarly, we must choose between using passwords or biometrics for authentication. In this case, inclusive-or, it is possible to have both, but at least one must be included. It is important to note that the resulting hierarchy does not represent implementation flow. Instead, child nodes represent patterns to be considered if the parent is used.

We felt that feature diagrams would be a suitable notation for visually representing pattern hierarchies because security concepts have mandatory and optional features. By breaking down the features of security concepts, we can construct a diagram using logical connectives which can help a software engineer understand which features are most important for their specific application. Once features associated with the security concept are identified, requirements generation can be customized to the user's specific situation.

C. *Inquiry-Cycle*

The *Inquiry Cycle Model* (ICM) by Potts et al. [22] is a requirements elicitation method for refining requirements by generating discussion through questions. This model is used to analyze requirements of a system through questions and answers which describe the solutions. We adapt this idea of using ICM questions by using the answers to "how", "why", "what", and "when" questions to direct users to the requirements pattern that will provide them with the resources needed to generate complete requirements. If a user was applying access control, they would start with the Access Control pattern and answer a question about how access to a resource is decided. If the user answers something like "the resource's owner decides who should have access to it," they would find the DAC pattern most suitable. Their answer to this question helped them select the correct type of pattern for their needs. This is a very simple example and not all situations will be so clearly outlined.

The ICM assists in the creation of pattern hierarchies to help users map their needs to relevant patterns in the context of their analysis. Section V describes how we were able to build and refine our hierarchy by interviewing security experts with the ICM. We encouraged experts to ask questions when dealing with hypothetical situations involving security. The experts continued to refine the requirements for the scenarios by asking questions for clarification. By "forcing" them to use an approach similar to the ICM, we were able to elicit the kinds of questions that would be used to connect patterns within hierarchies.

III. Pattern Template

The presentation of pattern content varies depending on the pattern author. In requirements engineering formal and semi-formal methods, such as Tropos [23] and Problem Frames [24], structure requirements knowledge into pattern-like representations. However, many approaches that principally identify themselves as patterns use natural language templates (NLT) to illustrate patterns [25, 26]. These NLT approaches tend to base their template on the so-called *Gang-of-Four* (GoF) [14] book which outlines the following essential pattern elements:

- *Pattern Name* – A simple phrase to describe the problem
- *Problem* – A description of when to apply the pattern
- *Solution* - A general arrangement of the elements used to solve the problem
- *Consequence* – The results and trade-offs of applying the pattern

Our approach includes two additional important features: *forces*, which determine quality attributes that are impacted (maximized, minimized, etc.) by the pattern, and *relations* or links to other patterns using an inquiry-based approach. For example, the access control requirements pattern may contain such forces as generalizability, flexibility, and modifiability since they would be particularly relevant to its application and a question such as, "How are authorization privileges bound to actors and resources?" may lead to an authentication pattern.

By balancing forces using questions which invoke discussion and refinement of security requirements, we aim to minimize the negative consequences or liabilities of the pattern (e.g., access control may require many rules and increase the system's complexity) as well as capture the applicability of the pattern. The use of an approach based on the ICM provides a way to connect existing security requirements patterns as well as refine system requirements themselves through questions that surface hidden assumptions about the system and its environment. The approach works on two levels: (Level 1) questions relevant to the problem can either draw out specific information from the situation which can be applied to generic requirements in the template in order to generate specific requirements, or (Level 2) they relate the pattern to other patterns relevant to the situation as a means to increase requirements coverage by identifying dependencies or to introduce alternative requirements that balance forces in alternative ways that may be better suited to a particular problem or stakeholder needs. Pattern hierarchies are constructed using the second level, which further supports eliminating unnecessary or unfitting requirements to select the most appropriate pattern.

To catalog patterns, we used a standardized pattern template consisting of six elements. The template we used is an elaboration of ideas devised at RePa'12 and our previous work on the Standardized Pattern Format for deriving characterizations and boundaries of patterns [27]. Using this pattern template is what enables us to construct a pattern hierarchy. To illustrate the different aspects of the template, we use access control as an example.

The template consists of the following elements:

- *Name* – a unique name that is limited to the scope of the pattern
- *Problem* – a statement of the problem to be addressed or a high-level goal to be achieved
- *Context* – domain assumptions that must be true in conjunction with the generated requirements
- *Forces* – the non-functional quality attributes that create trade-offs for the application of the pattern
- *Solution* – a set of questions which refine requirements and guide the pattern user to related patterns
- *Management* – additional information related to the pattern including examples and known uses

We now describe the use of each of the attributes using access control.

A. Name

For our running example, we name the pattern "Access Control" to describe what the pattern covers. In addition, it can be uniquely incorporated into the pattern hierarchy.

B. Problem

The problem shall be expressed either as the security objectives that need to be achieved or the threats that must be mitigated. For our example, we define the problem as "the confidentiality and integrity of resources shall be protected by regulating access to the resources based on different factors." Here, confidentiality and integrity, [28], two fundamental security attributes, are addressed.

C. Context

A pattern addresses a generic problem in a specific context [12], which shall describe the nature of the situation. This includes any domain assumptions as well as expectations of the system and its environment. For access control we describe the context as "any computer-driven environment in which the access to the resources needs to be regulated" and we list the following assumptions:

- The administrator involved in implementing the authorization system shall be trusted.
- The actors involved in granting or denying authorization shall have the ability to restrict access to the resources being protected.
- Actors shall not assume the identity of other actors with different rights.

We also include one expectation:

- Actors will not circumvent the system to gain access to resources.

D. Forces

Lists of forces are useful for identifying and balancing potentially conflicting or complementary aspects of a system that are imposed by the environment or the requirements [29].

For access control we list three forces:

- Generalizability – The authorization structure must be independent of the type of resources.
- Flexibility – The authorization structure shall be flexible enough to accommodate different types of principals (users or subsystems), objects, and rights.
- Modifiability – It should be easy to modify the rights of a principal in response to changes in his or her or its duties or responsibilities.

E. Solution

To solve the problem and balance the forces, the solution includes a series of ICM-inspired questions that help elicit responses from a user or a subject matter expert. A set of requirements templates with variable parts may be configured by the developer and may correspond directly to the questions. Answers to questions may be pre-conditions to additional questions, pieces to fill in the requirements templates, or guides to other patterns. The questions for our example are:

- Which entities (principals) exist in the system and what resources do they need to access?
- Can two entities access the same resource at the same time?
- How are the resources intended to be accessible?
- Can users of the system be categorized into roles that will have different access privileges? *See Role-Based Access Control pattern.*
- Can resources be assigned labels by the system so users can be given clearance to access resources based on levels of clearance? *See Mandatory Access Control pattern.*
- Can access to resources be regulated by the owner of the resource? *See Discretionary Access Control pattern.*
- How are entities to be authenticated in order to gain access to the system? *See Authentication pattern.*

The requirements template for our example consists of:

- <list of entities> shall <be permitted | not be permitted> to access <resource> simultaneously.
- <list of entities> shall access <resource> through the use of <action>.
- An entity shall be granted or denied privileges to <resource> based on <authorization criteria>

Italicized text refers a pattern-user to a corresponding pattern for the question (as seen in Figure 1). The template also lists generic requirements which can be customized based on the answers to some of the questions. For these requirements, words surrounded by "<" and ">" are mandatory and words surrounded by "[" and "]" are optional. In some cases, a choice of possible answers are provided separated by the "|" character.

F. Management

Finally, a pattern should include any information regarding the source of the pattern, the version, known uses, or any information relating to how the pattern was derived. For this pattern we note that it was adapted from the security design pattern from [12] and modified for requirements based on Withall's work on access control requirements [9].

IV. Pattern Hierarchy

Pattern relations are managed with the use of feature diagrams, which connect patterns in places where a question might lead a user to another pattern. The resulting hierarchy partitions a larger problem or security area into smaller problems (i.e. patterns). We believe such a hierarchical structure would reduce the time and difficulty needed to select an appropriate pattern and security requirements for a complex situation. Furthermore, the hierarchy enables users to easily discover related patterns which could be implemented to enhance their system security. Feature diagrams are not necessary for the end use of pattern hierarchies, as apparent in our user study. The diagrams serve as a means for visualizing and managing the structure. When knowledge of the feature diagram notation is not present, users can simply be guided with the pattern questions as long as the hierarchy has been constructed correctly.

Based on the solution space of the template, a pattern hierarchy can be derived as seen in Figure 1. The hierarchical relations between patterns appear within each pattern template as the questions that help the pattern user make decisions on which pattern to include. For the Access Control pattern in Section III, the question, "how are entities to be authenticated in order to gain access to the system?" is associated with the Authentication pattern. Figure 1 shows this connection with a link between the Access Control and Authentication patterns. Our approach uses inquiry-based discussion between stakeholders in order to revise previous iterations of requirements and introduce relevant patterns. Our approach begins by highlighting existing challenges for the problem (i.e. the forces) and uses questions to elicit security requirements as responses to those challenges.

We continue to use access control as an example to illustrate a hierarchy due to its large amount of related work including various aspects and specialized models [30, 31, 32]. Not only is it a commonly understood aspect of security, but it also includes many facets which allow us to demonstrate the different benefits of using feature diagrams. In Section III we described a single Access Control pattern from which we extend our example hierarchy as seen in Figure 1. The following subsections describe the use of each notation for our hierarchy design using the same example.

A. Optional

The notation for optional components is denoted by an unfilled circle on the end of the connection closest to the optional pattern. Such components are directly related to the inquiry-based approach. In Section III-E, our example included the question, "Can users of the system be categorized into roles that will have different access privileges?" which related the Access Control pattern to the Role-Based Access Control pattern. For such an instance where the answer to the question determines whether or not the related pattern is relevant, an optional connection would be made.

B. Mandatory

Mandatory components are denoted by a filled circle on the end of a connection closest to the mandatory pattern. In Figure 1, the Access Control pattern includes the Authentication pattern which is commonly viewed as a necessary component of access control [9] as a mandatory feature. Mandatory components do not stem from “yes” or “no” questions as such questions would imply a non-compulsory relation. Instead, such relations are made through “how” questions. For instance, we asked, “*How* are actors to be authenticated in order to gain access to the system?” Here, the word “how” poses the question so that it requires an answer. On the contrary, if the question were posed, “Are actors to be authenticated in order to gain access to the system?”, one could simply answer “no”, thus avoiding the mandatory component. Here, the phrasing requires that there must be a way for the actors to be authenticated. This question includes a reference to the Authentication pattern and is connected within the diagram as a mandatory component.

C. Inclusive-Or

Some questions may require a child pattern to be included, but allow for multiple children to be included simultaneously. Our example includes two forms of credential verification patterns: Password and Biometric. These patterns are included as an inclusive-or decision since it is acceptable to use multiple credential verification mechanisms and at least one is required. These relations have a filled triangle spanning the edges between the nodes in the decision. Cardinality is represented with brackets in the form [n, m] where n is the lower bound and m is the upper bound. A single number implies an exact amount.

D. Exlusive-Or

Similarly to the inclusive-or case, the exclusive-or relation allows for decisions between different patterns. However, this relation is used when only one choice may be made. These relations have an unfilled triangle spanning the edges between the nodes in the decision. The Access Control node illustrates the use of an exclusive-or decision between the MAC and DAC patterns. An unfilled triangle joins their edges to signify that only one can be chosen if this optional branch is selected.

V. Empirical Study Design

We believe that pattern hierarchies can improve users’ ability to select appropriate patterns and requirements faster and more effectively than unassisted pattern selection. Furthermore, the use of pattern hierarchies should allow novice software engineers with limited experience to benefit from expert knowledge embedded in such patterns and hierarchies.

To evaluate our approach we conducted a study to compare the pattern selection of two groups: one with a pattern hierarchy and one without a pattern hierarchy. The first step was to gather existing security patterns into a repository [33]. This assessment included design, architectural, and requirements patterns so we could better understand how different security concerns were addressed at different development stages among the pattern landscape. Sources included a literature review of textbooks and scientific publications on security patterns. A total of 143 security patterns, 30 of which we classified as security requirement patterns, were collected. We then gathered a set of patterns relating to access control from the repository and mapped them to the template described in Section III in order to create an initial draft of a pattern hierarchy.

Next, we reached out to security experts from the local security community with the intention of revising our hierarchy and preparing for the evaluation of the hierarchy with novice users. Two banking scenarios [37] regarding access control were presented to these experts and used as a means to gather more information and possible patterns for our existing hierarchy example. From transcriptions of expert interviews conducted using the ICM and feedback directly from the experts, we further refined the Access Control hierarchy. For each of the two scenarios we were also able to assemble a subset of patterns in the hierarchy which were most applicable. These subsets were used later in evaluation with novice users.

Finally, we evaluated the Access Control hierarchy comprised of patterns chosen by the experts by observing what patterns novices selected from it. This evaluation consisted of a comparison of novices both with and without the use of the hierarchy so as to test our hypothesis that the hierarchy would improve users’ ability over unassisted pattern selection.

A. Scenarios

The two scenarios used in our study involved access control in some way due to the common use of access control as an example for security [32]. This allowed for less time to be taken in training since it would be more likely for both experts and novices alike to have some background knowledge in the domain. The scenarios involved a fictional credit union and bank which were partnering with each other. This partnership required new software to be implemented to accommodate shared data. Our description of the entities described the number of employees, their jobs, and features from each of the entities’ existing software infrastructures (e.g., instant messaging system and administrative tools).

Scenario 1 described the ATM system currently in place for both institutions and gave the new requirements for the partnership. These requirements outlined the location, hours of operation, and usage fees for customers of each institution. A general set of auditing requirements were provided as well as the stipulation that only a subset of the bank’s ATMs would be accessible to credit union customers.

Scenario 2 described a computerized banking system with online banking. The system would need to be implemented only for the credit union, but with possibility for relevant data to be transferred to the bank. Specifications pertaining to user access, financial transaction services, customer services, and browser support were described.

These scenario descriptions were intended to include only high-level descriptions of the institutions’ requirements so as to elicit questions and discussion from the experts. This would allow us to infer more about the thought process of the experts interviewed. For situations where clarification on the scenarios was requested, a third-party trained as a domain expert representing the financial institutions was provided. The domain expert explained the different features required for

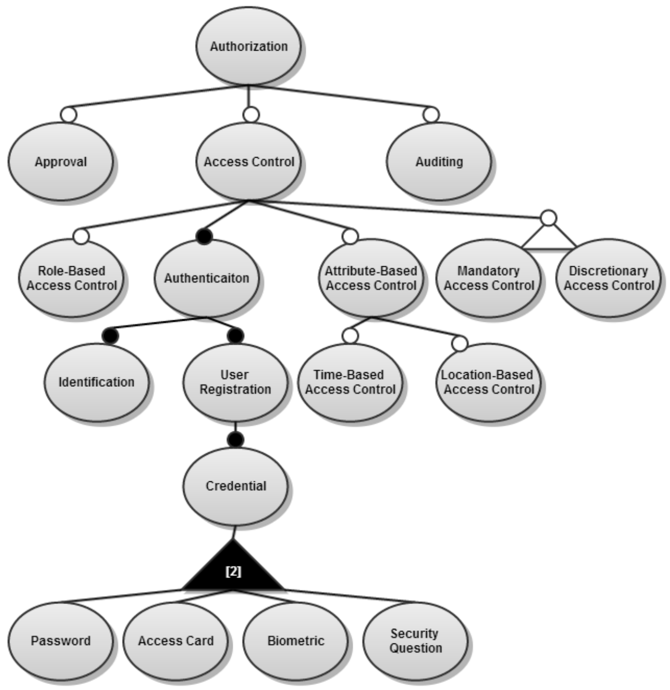

Fig. 2. Access control requirements hierarchy.

employees based on their roles. For any other questions, we required that the domain expert be consistent with his responses so as to provide a constant experience.

B. Sampling

We gathered expert participants with experience relating to computer security from academia and industry. We were able to recruit two members of a local security forum in San Antonio. Members of this forum were industry security specialists in the financial, utility, and local government sectors. We also enlisted participation from a member of a local application security consulting firm, a researcher and professor in the area of access control UTSA, a senior security staff member at a multinational computer technology corporation, and two post-doctoral researchers in the areas of privacy and security policies. In total, we interviewed seven industry and academic experts. Our goal was not to provide a complete representation of the security community's opinion, but instead to create an example to test the use of a hierarchy. For this reason, the representation given by our small sampling of security experts would be sufficient.

The second group consisted of university students with computer science backgrounds. We recruited participants in three undergraduate (junior and senior level) and one graduate-level computer science course. For each class, we gave a three minute explanation of what participants would be asked to do for our study and offered a $15 gift card as compensation for time. We were able to recruit 34 participants in this way. Due to the course requirements of the department, these students also had some knowledge of computer security. We chose students as subjects due to their fresh entry-level knowledge in access control with little industry experience. This would allow us to see whether pattern hierarchies are useful in relaying expert experience in order to make better and faster decisions.

C. Expert Interviews

We interviewed each expert with an existing hierarchy already created from patterns gathered from textbooks and our pattern repository [9, 12, 33, 35, 36]. Our goal was to see where our hierarchy needed expansion or revision as well as to find a unique subset of the patterns in the hierarchy that would apply to each scenario based on expert opinion. Such patterns subsets would be treated as the gold standard or "correct" set of patterns for evaluating novice performance. Furthermore, by interviewing experts, we sought to better understand how experts decide what security concepts are relevant to a particular system context. This information is fundamental to refining our pattern hierarchy.

The two banking scenarios were presented to each expert at the beginning of the interviews. We asked the participant to assume the role of a software analyst with the task of creating security requirements for the system. The actor playing the domain expert was introduced and the facilitator explained that he would be available for any questions or clarification that may be needed. The same domain expert was present during each interview to answer questions.

Interview questions were planned out according to a script [34] based on the ICM. We used this model as a means to encourage the expert to ask questions that could be used as the pattern-linking questions fundamental to the hierarchy. For example, when expert participants began to consider an access control policy they would ask the domain expert questions such as, "What kinds of users are there?" or "When do you want users to be able to access the system?". We looked for these kinds of cues to construct a hierarchy using the method described in Section IV.

After the interview was completed, we asked the participant to go through the existing list of security requirements patterns in the hierarchy and affirm whether or not the pattern should be regarded for the scenario along with an explanation for inclusion or exclusion. Experts were also encouraged to add anything else they thought was relevant. Transcriptions of the interviews were analyzed along with any notes the experts provided in order to create a final Access Control hierarchy as seen in Figure 2. This resulted in the Authorization pattern [9] being placed at the root of the hierarchy above Access Control. We continue to refer to the hierarchy as an Access Control hierarchy due to our scenarios being focused on access control as well as consistency.

A subset of the patterns in the final Access Control hierarchy was also compiled for each scenario based on expert responses to be used in the novice interviews. To do this, we tallied the selections made by the experts resulting in Figures 3 and 4. If at least five of the seven agreed on a pattern, we included it in the subset of correct patterns for the scenario. This resulted in 12 of 17 patterns for scenario 1 and 13 of 17 for Scenario 2.

D. Novice Interviews

Novice participants were interviewed individually and randomly placed into either a control or an intervention group. Members of both groups were asked to consider the same two scenarios given to the experts and were provided with the same domain expert present during the expert interviews. Participants for both groups were placed into the role of a

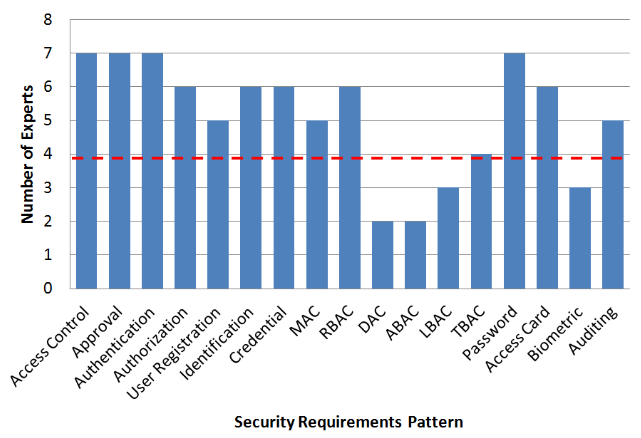

Fig. 3. Scenario 1 expert pattern selection.

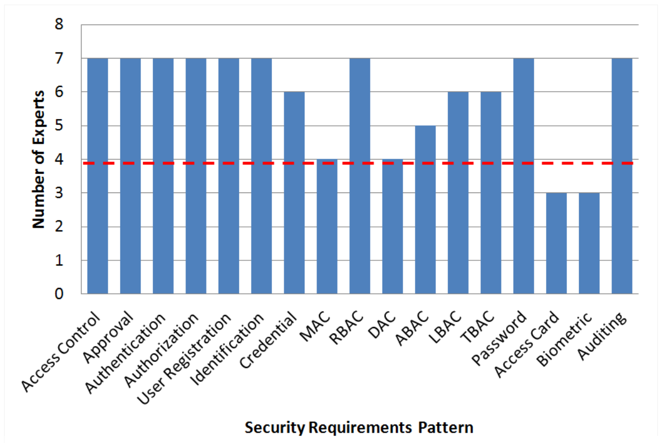

Fig. 4. Scenario 2 expert pattern selection.

software analyst and asked to select the most relevant patterns for each scenario from a list of the patterns from the Access Control hierarchy. Participants were informed that any information they provided would be made anonymous so as to encourage them to proceed naturally. The time to complete both scenarios was also recorded.

For the control group, only the list was given with no relations between the patterns as in the hierarchy. This group was provided with access to the Internet and asked to make the selections to the best of their ability. This gave us a baseline description of how well novices could do in pattern selection on their own.

The intervention group was presented with a questionnaire [34] representing the hierarchy. A questionnaire was used in order to forgo any confusion involving training novices in the use of feature diagrams. The questionnaire was created by organizing the pattern hierarchy questions so that they could correspond to the same checklist of patterns given to the control group. Instructions placed after each question both directed the participants to appropriate consecutive questions and had the participant check off appropriate patterns for the scenario based on the hierarchy.

E. *Threats to Validity*

Here we discuss both internal and external validity [37].

In order to make statistical comparisons between the two groups, we had to provide the control group with the same set of patterns to choose from as the intervention group. In the wild, it would be up to the user to select from all available patterns. By providing the control group with a list, we were forced to provide them with much of the work that would have been done by the hierarchy. Regarding external validity, we believe that the control group would have performed more poorly in both selection and speed without the provided list. This would actually further validate the use of pattern hierarchies.

We chose not to use the feature diagram representation of the hierarchy for our study due to the time constraints involved for training. This presented a risk to internal validity. Even with the feature diagram notation, the same questions must be answered as in the questionnaire. The feature diagram itself is useful for visualizing the flow between patterns; however the user must be familiar with the notation. We felt that the questions alone were enough for the user to gain the guidance provided by the hierarchy, and a properly-trained user would have only performed better. By providing the feature diagram representation to participants unfamiliar with it, we would have risked user error due to misunderstanding of the notation.

VI. Results and Observations

Here we discuss the knowledge we were able to extract from the experts as well as how our hierarchy affected novice decisions.

A. *Expert Knowledge Goes Beyond Patterns*

An important contribution of patterns is the reuse of expert knowledge. Our research asks if expert knowledge could go beyond individual patterns and expand to pattern organization and selection through the use of pattern hierarchies. Based on our interaction with experts and what we were able to produce with that knowledge, we found that expert knowledge can be applied in a larger scale through such hierarchies.

We used 14 patterns from textbooks and research papers prior to our interaction with experts [9, 12]. From the seven interviews we were able to expand that number to 17 from the additions the experts included to our checklist of patterns. The additions (auditing, time-based access control (TBAC), location-based access control (LBAC), and security question) were not in the form of explicit patterns, but fit into the hierarchy as supplemental security topics [34] related to the existing patterns. We assert that this implies the potential for yet-to-be-created patterns.

The links between patterns which formed the pattern hierarchy were produced by integrating information already present in patterns (e.g., “Related Patterns” sections) and information gained from expert interviews. Generally, as experts described what kind of security requirements were important for the scenarios they would ask the domain expert questions for clarification. For example, when discussing attribute-based access control (ABAC) [38], the ICM urged us to pose the follow-on question: “when should users have access to the system?” Experts would often respond by asking the domain expert if there would be times when employees would not be at work and if they should have access during those times. Depending on the answers to these questions, a

TABLE I. Statistical Results

	Average Success Rate (%)		Increase	Standard Deviation (s)		Sample Size (n)		Significance	Effect Size
	Control ($\bar{x}_C$)	Intervention ($\bar{x}_I$)	($\bar{x}_I$ - $\bar{x}_C$)	Control	Intervention	Control	Intervention	(p-value)	(Cohen's d)
Scenario 1	60.78	82.92	22.13	14.70	12.41	17	16	0.00003	1.63
Scenario 2	72.08	82.75	10.66	11.21	7.09	16	17	0.002	1.16

need for a corresponding requirements pattern would be necessary. For this example, the domain expert indicated that there are parts of the day when no employees are at work and thus should not be able to access the system. This triggered the expert to begin considering requirements that would be part of a TBAC pattern. The resulting relation between ABAC and TBAC could now be represented with the question: "are there times when users should not have access to the system?" This is represented in Figure 2 by the optional connection between these two patterns. The question itself would become part of the solution space of the ABAC pattern.

B. *Pattern Hierarchies are Efficient and Usable*

Novice results were measured by correct pattern choice and time to completion. In both cases, the intervention group performed better than the control group.

Regarding correct pattern choice, novices were graded depending on if they made the same decision to include or exclude a particular pattern as defined by the gold standard pattern sets derived from expert decisions described in Section V-C. Points were not deducted for incorrect answers. A majority of the experts expressed the need for two-factor authentication [39] for both scenarios. To accommodate this, a correct choice of authentication patterns (password, biometric, access card, and security question) consisted of any two. If a participant chose only one of those patterns, they were not awarded points. This selection scheme is apparent in Figure 2 with the range of necessary patterns denoted within the exclusive-or arc with "[2]".

Before computing the results of our study, we removed outliers by using Iglewicz and Hoaglin's modified Z-score with median absolute deviation method (MAD) [40]. The MAD for each group was calculated with:

$$MAD = median(|x_i - \tilde{x}|)$$

Modified Z-scores (M_i) for each score were calculated with:

$$M_i = \frac{0.6755 * (x_i - \tilde{x})}{MAD}$$

Where x_i represents individual scores and $\tilde{x}$ is the median. 0.6755 is the multiplier used for outlier detection recommended by Iglewicz and Hoaglin for samples of our size. For this method, values of M_i greater than ($3.5 * MAD$) were removed as statistical outliers. This resulted in the removal of two participants.

Table I describes the quantitative results of the study regarding successful pattern choice by novice users. For both Scenario 1 and Scenario 2 novices were more successful at making the same choices as experts when using the pattern hierarchy with a success rate increase over the control group of 22.13% and 10.66% respectively. Regarding the lower increase for Scenario 2, we found that fewer users asked the domain expert questions for that scenario. We took this as an implication of misappropriated familiarity with the situation.

We used a standard t-test to measure the significance of our results. We tested the null hypothesis (H_0) that the mean population success rates for users without the hierarchy (μ_C) and users with the hierarchy (μ_I) were equal:

$$H_0\colon \mu_C - \mu_I = 0$$

The p-values produced by the t-tests indicate that the null hypothesis could be rejected for both scenarios and that the results of our study were statistically significant.

As an indicator of the strength of our sample size, we calculated effect size using Cohen's d [41]. This was done by taking the difference between the mean sample success rate for the control group ($\bar{x}_C$) and the mean sample success rate for the intervention group ($\bar{x}_I$) divided by the average standard deviation for both groups, s_{IC}:

$$d = \frac{\bar{x}_I - \bar{x}_C}{s_{IC}}$$

While the p-values explained in the previous paragraph give us indications of whether or not our null hypothesis can be rejected, and thus our hypothesis be accepted, effect size is an indicator of whether the result of our experiment is meaningful regardless of our sample size. Based on Cohen's conventions [41], the values for both Scenario 1 and Scenario 2 indicated a large effect size. This implied meaningful results regardless of our sample size.

Efficiency in terms of completion time was recorded as a total for both groups due to the design of our study. Table II shows an average decrease in selection time of more than 80% for the group using the pattern hierarchy. We attribute this increase in selection speed to the guidance the hierarchy provided. Without the hierarchy, the control group was forced to use their previous knowledge and the Internet to make decisions. Participants were able to make decisions faster while covering the same amount of patterns with the aid of the straightforward questions provided by the hierarchy. We calculated statistical significance and effect size with the same methods used for success rates. Both values implied statistical significance and meaningful results.

VII. Related Work

Researchers have been documenting security patterns for decades, and there have been similar efforts to increase the usability of patterns [16]. We now review and discuss the similarities of related efforts in the security pattern domain, and how they differ from our own work.

Romanosky extends the work done by Yoder and Barcalow [42] by introducing an additional eight security patterns [43]. Romanosky adopts the standardized pattern template originally developed by the GoF, which includes: *Problem*, *Forces*, *Solution*, and *Consequences*. Their template also uses the following additional elements not included in our approach: *Alias*, *Motivation*, *Example*, and *Related Patterns*.

TABLE II. EFFICIENCY

Average Completion Time (minutes)		Decrease	Speedup	Standard Deviation (s)		Sample Size (n)		Significance	Effect Size
Control ($\bar{x}_C$)	Intervention ($\bar{x}_I$)	($\bar{x}_C$ - $\bar{x}_I$)		Control	Intervention	Control	Intervention	(p-value)	(Cohen's d)
28.56	15.82	12.74	1.81	11.85	4.95	16	17	0.00037	1.52

Although many of the element titles are identical in both templates, we assign slightly different meanings. For example, Romanosky's *Solution* section is a textual description of a step or series of steps that, when applied, can mitigate the problem. In contrast, our Solution is represented as a series of questions derived using the ICM, to guide users to other patterns and to generate requirements based on answers to these questions as described in Section III. Additionally, questions in Romanosky's approach do not direct readers to related patterns nor do they provide a customized solution based on their answers. Moreover, our Forces section is used to influence the types of questions contained within the Solution, by surfacing trade-offs that arise from other quality attributes.

Based on the analysis of 220 security patterns, Heyman et al. investigate possible improvements that can be made to increase their usability [44]. Their major conclusions include that the quality of patterns would greatly benefit by the adoption of a common documentation template, and that the construction of a "pattern inventory" would make pattern selection a less daunting task. As opposed to having a general inventory, our approach aims to contextualize each pattern in relation to other relevant patterns through the ICM and Forces. In this way, the pattern user discovers these other patterns as they become relevant and while the pattern user iterates to build a solution from each pattern.

Kienzle and Elder address some of the same problems we address with the security patterns landscape that Heyman et al. documented [45]. Specifically, they address the creation of a common pattern template and pattern repository. Like our template, their common pattern template was derived from the GoF template. Kienzle and Elder augmented the GoF template by including additional security-specific elements. These additions make the patterns somewhat lengthy and, as discussed in the previous paragraph, our Solution section provides more than just a textual description of a procedure. Using their template, they have documented a total of 26 security patterns in a patterns repository.

Hafiz et al. introduce an approach for organizing and describing the relationship between patterns using directed acyclic graphs [16]. Their approach uses a hierarchical scheme based on threat models in order to classify security patterns. Patterns which target similar problems are grouped together similarly to our hierarchical approach. With these grouped patterns, the researchers analyzed the order in which they would be applied to describe their relationships and position in the hierarchy. Dotted line arrows and solid line arrows show relationships between patterns for preventing 'tampering and DoS (denial of service)' and 'Escalation of Privilege', respectively. The authors admit that their pattern language is still a work in progress, is hard to read, and can be "large and intimidating." Based on the results from our empirical study, we believe that the use of feature diagrams provides a more usable method of pattern selection for novice users.

Nonfunctional requirements (NFRs) are an important part of requirement specification. Existing work on NFRs defines them as attributes of or constraints on a system [46]. We include forces as a representation of quality attributes in our pattern template. This provides a means for NFRs to be incorporated into the pattern selection process.

The representation and organization of requirements exists in other forms besides feature diagrams. Giorgini et al. use *goal models* to qualitatively relate goals with other goals [47]. This is done with "+" and "-" relationships which signify positive and negative contributions between goals. Similarly to feature diagrams, goal models also incorporate AND and OR compositions. We chose to use feature diagrams due to their expressiveness compared to other goal-oriented modeling techniques which do not allow for the representation of cardinality [48]. Furthermore, the distinction between positively and negatively relating patterns is not necessary for the goal of pattern hierarchies. Hierarchical relationships are strictly positive. Goal models also use binary relations meaning associative compositions are required to create n-ary relations. Feature diagrams include n-ary operations in their most fundamental form.

VIII. CONCLUSION AND FUTURE WORK

Security requirements patterns can greatly reduce the time spent in the requirements elicitation phase of software design as long as the pattern user identifies the correct pattern. Our comparison of pattern selection with and without the hierarchy showed statistically significant and meaningful results. This upholds our hypothesis that pattern hierarchies can be created to allow engineers to make more expert-like decisions in efficient time. Using this method, software engineers can be more confident in the completeness of their requirements specifications. We feel that, based on our results, incorporation of pattern hierarchies with existing patterns can better facilitate knowledge transfer in the requirements engineering domain.

In the future, we plan to modify our digital repository of security patterns to incorporate pattern hierarchies and provide public access in order to both benefit software engineers and gain feedback to further our research. We will also work to mine new patterns for the instances where experts in our case study indicated yet unimplemented patterns (e.g., Auditing and TBAC). Finally, we will work to formalize a method for hierarchy creation including an outline for generating questions used in the hierarchy. In doing so, we hope to provide a means for the creation of more hierarchies in different domains.

ACKNOWLEDGMENT

We thank Daniel Sass for his contribution to the design of our user study and Hanan Hibshi, Maria Riaz, and Laurie Williams for their insightful comments. We thank Daniel Amyot, Xavier Franch and other RePa'12 participants for their contributions in the initial brainstorming on requirements patterns. This research was funded by Army Research Office (Award #W911NF-09-1-0273). Jianwei Niu and Rocky Slavin have been supported in part by NSF award CNS-0964710.

References

[1] S. Konrad and B. H. Cheng, "Requirements patterns for embedded systems," *RE'02*, pp. 127-136, 2002.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, A pattern language, Oxford University Press, 1977.

[3] M. Lin and H. Guo, "Present situation and development of single sign-on tehnology," Journal of Computer Applications, pp. 248-250, 2001.

[4] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Compute 29(2)*, pp. 38-47, 1996.

[5] K. Beck, R. Crocker, G. Meszaros, J. Vlissides, J. O. Coplien, and L. Dominick, "Industrial experience with design patterns," *ICSE'96*, pp. 103-114, 1996.

[6] G. McGraw, "Testing for security during development: why we should scrap penetrate-and-patch," *IEEE T. Aero. Elec. Sys.* 13(4), 1998.

[7] B. Boehm, "Software engineering economics," *TSE10*, pp. 4-21, 1984.

[8] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Progress in Informatics*, No.5, pp. 35-47, 2008.

[9] S. Withall, Software Requirements patterns, Microsoft Press, 2007.

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

[11] M. A. Jalil and S. A. M. Noah, "The difficulties of using design patterns among novices: an exploratory study," *ICCSA'07*, 2007.

[12] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, Security patterns: integrating security and systems engineering, John Wiley & Sons, 2006.

[13] M. Weiss and H. Mouratidis, "Selecting security patterns that fulfill security requirements," *16th ICSE'08,* pp. 169-172, 2008.

[14] E. Gamma, R. Helm, R. Johnnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1994.

[15] T. D. Breaux, H. Hibshi, A. Rao, and J-M. Lehker. "Towards a framework for pattern experimentation: understanding empirical validity in requirements engineering patterns." *RePa'12,* pp. 41-47, 2012.

[16] M. Hafiz , P. Adamczyk , R. E. Johnson, "Growing a pattern language (for security)," *Onward!'12* pp.139-158, 2012.

[17] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie Mellon Univ., CMU/SEI-90-TR-021, 1990.

[18] P. Zave. (2004). *FAQ sheet on feature interactions* [Online]. Available: www.research.att.com/~pamela/faq.html

[19] D. Batory, "Feature models, grammars, and propositional formulas," *9th International Conference on Software Product Lines,* pp. 7-20, 2005.

[20] D. Bell and L. LaPadula. Computer Security Model: Unified Exposition and Multics Interpretation, MITRE Corp., ESD-TR-75-306, 1975.

[21] G. Graham and P. Denning. "Protection-principles and practice," *AFIPS Spring Joint Computer Conference,* pp. 417–429, 1972.

[22] C. Potts, K. Takahashi, and A. I. Antón, "Inquiry-based requirements analysis," *IEEE Software,* pp. 21-32, 1994.

[23] J. Mylopoulos and J. Castro, "Tropos: a framework for requirements-driven software development," *Information Systems Engineering: State of the Art and Research Themes*, pp. 261-273, 2000.

[24] M. Jackson, Problem frames; analyzing and structuring software develoment problems, Addison-Wesley, 2001.

[25] C. Palomares, C. Quer, X. Franch, C. Guerlain, and S. Renault, "A catalogue of non-technical requirement patterns," *RePa'12,* pp. 1-6, 2012.

[26] D. Dietrich and J. M. Atlee, "A pattern for structuring the behavioral requirements of features of an embedded system," *RePa'12,* pp. 1-7, 2012.

[27] R. Slavin, H. Shen, and J. Niu, "Characterization and Boundaries of Security Requirements patterns," *RePa'12,pp. 48-53,* 2012.

[28] J. Anderson, "Information security in a multi-user computer environment," in *Advances in Computers (12),* pp. 1-35, 1973.

[29] S. Lauesen, *Software Requirements: Styles and Techniques*, Pearson Education Limited, 2002.

[30] R. Crook, D. Ince, and B. Nuseibeh, "On modelling access policies: relating roles to their organisational context," *RE'05,* pp. 157-166, 2005.

[31] M. Koch, F. Parisi-Presicce, "Formal access control analysis in the software development process," *FMSE'03,* pp. 67-76, 2003.

[32] H. Hibshi, R. Slavin, J. Niu, and T.D. Breaux, "Rethinking security requirements in RE research," Tech. Rep. Report CS-TR-2014-001, Univ. Texas at San Antonio, 2014.

[33] J-M. Lehker. (2014). *Security Pattern Repository* [Online]. Available: http://sefm.cs.utsa.edu/repository/patterns/

[34] R. Slavin, J-M Lehker, J. Niu, and T. D. Breaux, "Managing security requirements patterns using feature diagram hierarchies," Tech. Rep. CS-TR-2014-002, Univ. Texas at San Antonio, 2014.

[35] S. S. Council. Payment Card Industry (PCI) Data Security Standard, 2nd ed., 2010.

[36] D. Kim, P Mehta, and P Gokhale. "Describing access control models as design patterns using roles,"*PLoP'06*. 2006.

[37] R. K. Yin. *Case study research,* 4th ed. In Applied Social Research Methods Series, V.5. Sage Publications, 2009.

[38] L. Wang, D. Wijesekera, S. Jajodia, "A logic-based framework for attribute based access control," *FMSE'04,* pp. 45-55, 2004.

[39] M.L. Das, "Two-factor user authentication in wireless sensor networks," *TWC'09*, pp. 1086-1090, 2009.

[40] B. Iglewicz and D. C. Hoaglin (1993), "How to detect and handle outliers", in *The ASQC Basic References in Quality Control: Statistical Techniques*, vol. 16, ASQC Quality Press, 1993.

[41] J. Cohen, "A power primer," *Psychological Bulletin*, 1992, pp. 155-159, 1992.

[42] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," *PLoP'97,* pp. 1–37, 1997.

[43] S. Romanosky. Security design patterns part 1. http://citeseer.ist.psu.edu/575199.html, Nov 2001.

[44] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen, "An analysis of the security patterns landscape," *SESS '07*. p. 3, 2007.

[45] D. M. Kienzle and M. C. Elder, "Security patterns for web application development, final technical report", 2003.

[46] M. Glinz, "On non-functional requirements," *RE'07,* pp. 21-26, 2007.

[47] P. Giorgini, J. Mylopoulos, and E. Nicchiarelli, "Reasoning with goal models," *ER'02*, pp. 167-181, 2002.

[48] C. Borba and C. Silva. "A comparison of goal-oriented approaches to model SPLs variability," *ER Workshops '09*, pp. 244-253, 2009.