Towards a Framework for Pattern Experimentation

Understanding Empirical Validity in Requirements Engineering Patterns

Travis D. Breaux, Hanan Hibshi
Institute for Software Research
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
breaux@cs.cmu.edu

Ashwini Rao
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
arao@cmu.edu

Jean-Michel Lehker
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas, USA
j.lehker@gmail.com

***Abstract*— Despite the abundance of information security guidelines, system developers have difficulties implementing technical solutions that are reasonably secure. Security patterns are one possible solution to help developers reuse security knowledge. The challenge is that it takes experts to develop security patterns. To address this challenge, we need a framework to identify and assess patterns and pattern application practices that are accessible to non-experts. In this paper, we narrowly define what we mean by patterns by focusing on requirements patterns and the considerations that may inform how we identify and validate patterns for knowledge reuse. We motivate this discussion using examples from the requirements pattern literature and theory in cognitive psychology.**

Keywords-requirements engineering; patterns; pattern-mining; abstraction

I. INTRODUCTION

Information security has received increasing attention with several recent high profile attacks, including the compromised RSA SecureID database [7], the Stuxnet attack [40], and the recent data breach at Global Payments [36], a credit card payment processor. Yet, security and the associated risks are not new. For decades, standards bodies, such as the U.S. National Institute of Standards and Technology (NIST) and International Organization for Standardization (ISO), have routinely published guidance on how to secure information systems. Despite these efforts, government and industry often fall short of achieving a "reasonable" level of security [6]. Based on our prior research [6], we believe this shortfall is due to a lack of accessible knowledge to developers, or the inability of developers to map security best practices to problem spaces[1]. Thus, we are interested in how patterns and associated abstractions can facilitate an analyst's ability to acquire, codify, transfer and apply security requirements knowledge to information system problem descriptions. While our primary interest is in security requirements, we believe our interest in patterns can be applied to other domains as well, such as accessibility, privacy, or reliability.

In this paper, we define what we mean by pattern, including other software and requirements engineering uses of the term, before exploring our vision for studying this problem empirically.

II. WHAT DO WE MEAN BY PATTERN?

The term pattern has appeared in several contexts in software engineering. Gamma et al. introduced several design patterns that describe how to refactor object-oriented source code to achieve certain code quality attributes, such as maintainability. For example, the *singleton pattern* provides a mechanism for allocating program variables and functions in a single class, as opposed to duplicating these variables across multiple classes and maintaining concurrency. Thus, a developer need only use one or more method calls to reach across one or more classes to access a shared variable.

In software architecture, quality attributes have been addressed by appealing to patterns, styles and tactics. Architectural patterns and styles consist of typed components and connectors and the constraints that dictate how instances of these types can be combined into a design [35]. For example, the *pipe and filter pattern* describes how data is transferred via pipes and transformed via filters, which improves modifiability and extensibility by allowing designers to more easily reconfigure data flows by replacing filters or adding new pipes. Architectural tactics describe specific strategies for achieving quality attributes [3]. For example, to improve performance, one might employ the tactic *reduce operational overhead*, which generally highlights that execution time is manifest in the management of the execution itself (e.g., queuing, scheduling, etc.) As abstractions, these constructs can be reused to frame solutions across multiple problem spaces.

Although patterns in programming and architecture are insightful, we are primarily interested in requirements patterns as a means to share domain knowledge in the early stages of software development and design. Thus, we conducted a small literature review as follows: (1) we searched for full research papers in the IEEE Requirements Engineering Conference Proceedings and the two predecessor conferences using only the keyword "pattern," which resulted in 64 papers; (2) we excluded posters, short papers and position papers, which reduced the results to 32 papers; finally, (3) we excluded papers that did not include at least one pattern in the main body of the paper or contributions such as in the abstract or when describing related work, which yielded a final result of 21 papers. Then we conducted a two-step classification procedure. First, three

[1] We refer to the problem spaces in the context of Michael Jackson's seminal work [18]

investigators (the second, third, and fourth authors) reviewed each paper and answered the following three questions:

- ***Goals****: What is the goal of the pattern?* For example, some patterns are used to improve requirements education, while others are used to improve elicitation.
- ***Sources****: What is the source of knowledge that the pattern is derived from?* For example, an elicitation pattern may be derived from an interview or meeting transcript, whereas an i* pattern may be derived from an i* diagram.
- ***Representations****: How is the pattern described?* For example, a pattern may be described in a table, a context-free grammar, or in natural language text.

Second, we reconciled the answers to these questions into three taxonomies (goals, sources, representations), which were each critiqued by a fourth reviewer (the first author). We now report our results of this two-stage process to help us define the term "requirements patterns."

A. The Goals of Requirements Patterns

We categorized the patterns by identifying what goal each pattern sought to achieve. The identified goals are:

1) ***Improve Requirements Acquisition*** – Requirements may be acquired from a variety of sources. Elicitation, which consists of soliciting requirements from stakeholders, is supported by improving communication among requirements engineers [27, 28], between requirements engineers and stakeholders [10, 31, 32], or between requirements engineers and other developers (testing, design, implementation, etc.) [24]. In addition, Rawsthorne proposes a pattern to capture functional requirements from problem descriptions, which we call extraction [31]. In addition to elicitation and extraction, one may reuse previously developed specifications as a form of acquisition [11, 29]. Several patterns were also applied to functional and non-functional requirements to enable reuse [21, 37].

2) ***Improve Requirements Quality*** – While quality is difficult to measure, some argue that quality improves with reduced ambiguities, inconsistencies, etc. Konrad and Cheng aim to reduce ambiguities in natural language requirements specifications by mapping textual requirements to temporal logic specifications [22]. For others, quality may refer specific attributes, such as security. The Secure Tropos framework is an approach proposed by Giorgini et. al. that employs security patterns [16].

3) ***Improve Requirements Engineering Process*** – A pattern can improve the overall requirements process. Araújo et. al. introduce a pattern to improve the process by modeling scenario-based requirements, because users understand scenario patterns easily [2]. Hagge and Lappe support project managers with patterns for capturing requirements knowledge to enable knowledge transfer between users [17]. Rolland and Plihon describe patterns to generate new requirements process models [34]. Similarly, patterns exist to tailor the requirements process to accommodate a specific domain [15].

4) ***Improve Compliance*** – Requirements researchers introduce patterns to address compliance with laws, policies and standards. Breaux et. al. use patterns to identify "gaps between legal and product requirements" and hence, recommend ways that improve compliance [5]. Kartseva et. al. propose a pattern to improve compliance with business requirements in networked organizations where objects of economic values are exchanged and could be placed at risk [20].

5) ***Improve Runtime Performance*** – Patterns to improve runtime performance include predicting the run-time behavior and model [9], and reducing run-time anomalies using patterns [24, 25].

The above goals illustrate our small pattern sample. Because our sample was limited to one conference, it is not a complete list. In addition, the goals are not independent from each other, as some goals overlap and interact with each other (e.g., goals to improve specification also improve acquisition). However, by categorizing patterns by the goal they seek to achieve, we may better compare patterns by the source of knowledge from which the patterns originate and strategies for expressing patterns targeted at the same goal.

B. Requirements Pattern Knowledge Sources

We classified each pattern by the source of knowledge from which the pattern was derived. For example, a pattern may be derived from meeting transcripts by analyzing how participants communicate in a particular meeting. Depending on how the pattern is expressed, it may be applied to other meeting transcripts or to actual meetings, which is a topic we do not address in this section. Figure 1 summarizes the results of our classification: nodes correspond to knowledge types (the root node has a double-line border) and edges point from classes to subclasses. For example, "documented communication" is a subclass of "textual documentation," which has subclasses "questionnaires", "interview transcripts", and "meetings transcripts," etc. We distinguish meeting transcripts, which document which actor exactly said what and can be produced from audio recordings, from actually applying a pattern to active discussions in a meeting, which we would classify as a type of "direct experience" in Figure 1. Finally, each node includes the reference [in brackets] to the cited paper that describes the corresponding pattern.

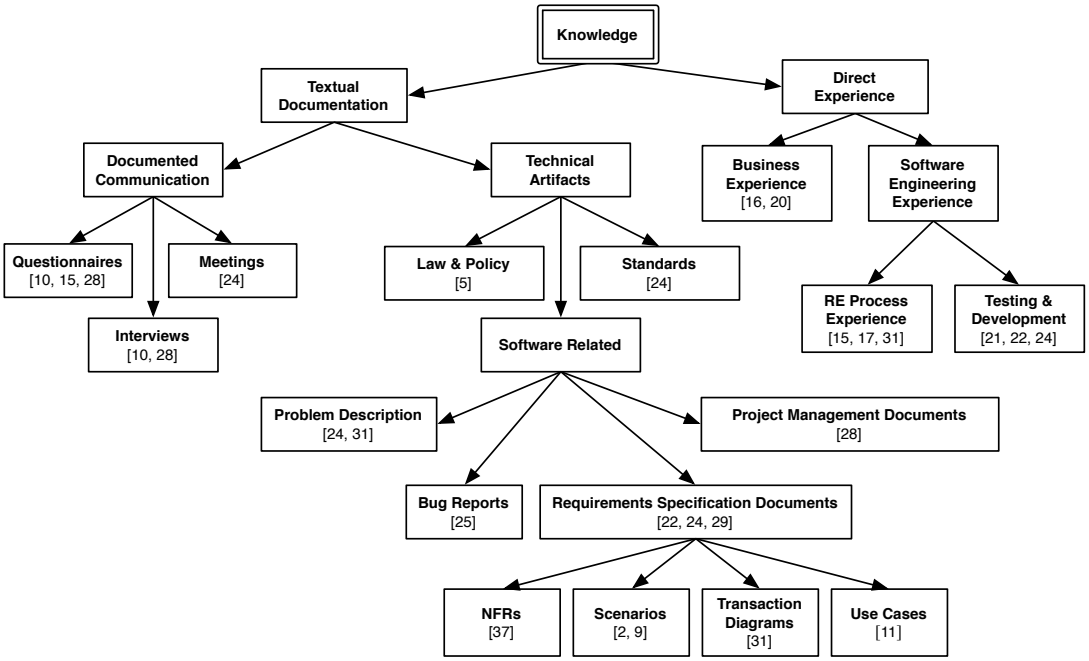


Figure 1. Preliminary Taxonomy of Pattern Sources

With respect to our sample of requirements pattern papers, we found more patterns applied to textual documentation than direct experience and requirements specifications were the most common type of textual documentation studied. These include functional and non-functional requirements (NFRs) expressed using a variety of formats such as scenarios [2, 9], transaction diagrams [31], and use-case models [11]. Patterns applied to textual documentation are frequently used to improve requirements elicitation and quality. These patterns enable visualizing NFRs [37], converting natural language requirements to formal or semi-formal specifications [22] and validating requirements [24]. In addition, these studies also sought to partially automate requirements activities using tools [37, 22, 24]; whereas only one direct experience study used a tool to examine processes [34]. Patterns that apply to direct experience, such as business or engineering practices that occur in real time, generally focus on improving requirements processes [15, 17, 34].

C. Requirements Pattern Representations

Requirements patterns can be represented in many ways: natural language, first-order logic, various forms of graphs and templates. Often, the representation is determined by how one expresses the solution, but this is not always the case. In Figure 2, we summarize the analysis of our sampled patterns' representations into four main categories that appear as three root nodes with double-line borders: *template-based patterns* that consist of key-value pairs, where the engineer instantiates the pattern by supplying the values to various keys; *natural language (NL) patterns* that are described by textual narrative (excluding NL statements that contain "variables" or "slots" to which an engineer can assign values, which we call syntactical templates); *first-order logic* (FOL) patterns that are expressed in any variant of FOL; and *graph-based patterns*, where the pattern is expressed using nodes and edges with various semantics (e.g., a node in a social network may correspond to a person, whereas a node in the UML class diagram may correspond to an object's class). In Figure 2, each edge describes a subclass relationship, wherein a subclass has an edge leading to a super class. For example, a sequence diagram is graph-based representation that is in the broader category of the UML. Finally, each node includes the reference [in brackets] to the cited paper that describes the pattern.

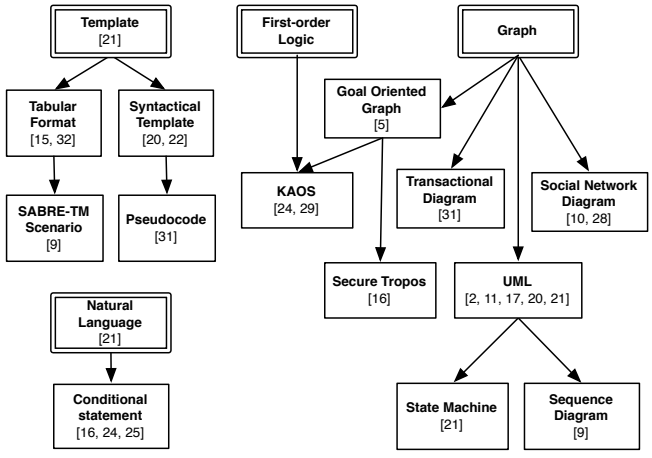


Figure 2. Preliminary Taxonomy of Pattern Representations

While natural language patterns are repeated observations, such as misunderstandings [25], or principles (e.g., "specifications mirror project structure") [16], other patterns express a solution using the same representation as the pattern. Konrad and Cheng reused the existing UML standard to "represent common requirements patterns" [21]. They use state, sequence, and class diagrams to describe patterns from the automotive domain, such as the "user interface pattern" and the "fault handler pattern" that illustrate structural and behavioral requirements. Alternatively, in Transaction Based Analysis, Rawsthorne introduces a novel notation called transaction diagrams that he claims are "robust and easily communicated to the user" [31]. The use of custom notations allows tailoring the pattern instantiations to the semantics of the domain, as opposed to reworking existing semantics, which can be awkward (e.g., extending the meaning of a "class" to be an architectural component, as opposed to a source code module in an object-oriented programming language). We believe the choice of pattern representation can influence the analysts' ability to recognize important problem cues and apply the pattern, a topic that we discuss in Section III, under *Representations*.

III. Towards an Experimental Framework

In software engineering, many researchers appeal to Christopher Alexander's original definition of a pattern, which denotes that patterns describe a recurrent problem that "occurs over and over again," and that yields a repeatable solution that one can "use a million times over" [1]. This definition poses certain challenges to the pattern experimenter, who seeks to demonstrate *problem recurrence* and *solution repeatability* and moreover to determine which *contextual cues* are necessary for an analyst to recognize the problem and apply the pattern in practice. In this paper, we explore a simplified version of pattern application, while recognizing the possibility for other scenarios outside the scope of our discussion. We define *pattern application* as a human analyst's ability to: 1) recognize the goal of a given pattern; 2) recognize cues in the pattern and cues in an input description (the problem) that indicates when the pattern may be applicable; and 3) apply the pattern to yield an output description (a solution) that satisfies the technical constraints of the pattern. We define *pattern validity* as the probability that, given an input description, the analyst can apply the pattern to yield an output description that satisfies the technical output constraints of the pattern. Validity may be a quantitative value, or a qualitative argument constructed from empirical evidence. We now discuss validity in the context of the three components of our patterns taxonomy (goals, sources and representations):

Goals: Requirements patterns presumably aid an analyst to achieve some goal. In requirements engineering, goal satisfaction is described by *how* a goal is achieved, maintained or avoided [12]. Non-functional and soft goals can be satisficed, which requires defining the acceptable limits in which a non-optimal solution is accepted [26]. Thus, for pattern experimenters, it is important to define the goal of the pattern and how the investigator proposes to measure the satisfaction of this goal. For example, it is easy to claim that a pattern improves requirements quality, but it is more challenging to show in detail how the quality was actually improved and what type of quality evaluation assisted in reaching that conclusion. The strength of the argument made by the investigator that the pattern application consistently satisfies the goal correlates with pattern validity: strong arguments yield higher pattern validity, and vice versa. To the extent that the experimenter can measure the goal satisfaction statistically, the validity argument becomes more precise, repeatable and generalizable. Patton suggests combining qualitative and quantitative methods for research studies because each approach has its strengths and weaknesses: the statistical nature of quantitative methods makes the findings broader and more generalizable, while the qualitative approach enriches the results with detailed information that provides a deeper understanding of the problem [30].

Sources*:* The source of knowledge from which a pattern is derived may be highly structured, such as scenarios, bug reports, and use cases, or less structured, such as meeting notes, diaries, or video. We conjecture that unstructured sources will be more challenging to prove pattern validity, because the pattern user must recognize cues within the source to determine when to apply the pattern. In structured sources, these cues may correspond to typed data, such as pre-conditions in a scenario, or bug report types in a finite list. We believe that knowledge sources that are *more enumerable* and *less ambiguous* will improve the pattern user's ability to make reliable and repeatable inferences when matching patterns to input descriptions. Enumerative means the pattern user can count the various parts of an input description, thus, distinguishing one part from another by types or codes. While there are many kinds of ambiguity in requirements, wider variation in how pattern users interpret these types or codes may lead to misapplication, thus reducing pattern validity.

Representations*:* The relationship between the problem structure and the pattern representation, or solution structure, can affect validity. For example, Marczak et al. derived a brokerage pattern from questionnaires and interviews (see Figure 1) [28]. The brokerage pattern emerged as a broker *b* who sits between two other agents, the communication sender *s* and receiver *r*; the broker plays a critical role in ensuring project success, when senders and receivers do not communicate directly [28]. Because the target representation includes a social network consisting of links between senders and receivers, and because the knowledge source consists of descriptions of communication acts between stakeholders, the pattern representation is logically aligned with the knowledge source. Thus, we believe it is easier to express the solution using this representation. However, if the source was not logically aligned, and instead the pattern user was required to make several inferences to connect the knowledge source to the representation, we would expect to see a decrease in successful pattern applications and thus a decrease in validity.

Research Questions: Pattern validity raises a number of research questions for a pattern experimenter that we believe are important, but that we cannot fully address in this paper:

- What are the pattern goals, and how do we measure goal satisfaction experimentally?
- What is the range of input descriptions, and what features from an input description increase or decrease validity?
- What are the cognitive abilities of the analyst?
- What can we change in a pattern to improve perceptions of applicability, or improve the transformation from problem to solution?
- How do we measure technical constraint satisfaction in the output description?

We recognize this formulation is still only a partial attempt to address this area. We assume that the technical constraints literally imbue the output description with the pattern's properties (e.g., maintainability, reliability, privacy, etc.) Whether these properties are preserved after the description is combined with other descriptions, or after the description is merged into a specification, are issues that must be addressed by future work or existing, well-established techniques, such as requirements traceability. We also recognize that this definition of pattern validity cannot apply to all requirements patterns: while some patterns may not be empirically valid by our definition, these patterns could still be quite useful, effective and relevant for reasons that are measured or inferred in other ways. Thus, part of our research is to characterize the types of patterns that fit within the scope of our assumptions, recognizing that our assumptions will inherently limit our focus. We now discuss how we propose to investigate the above questions by appealing to work in cognitive psychology.

IV. THE PSYCHOLOGY OF PATTERNS

"What are the cognitive abilities of the analyst?" is one research question that needs to be addressed for pattern validity. In order to explore possible solutions to such a question, we looked into the cognitive psychology field and, specifically, into the taxonomies of learning that attempt to explain different types of human learning. As the basis of our exploration towards developing an experimental framework, we use the revised Bloom's Taxonomy to preliminarily[2] explain who should perform well at pattern mining and application. We chose Bloom's Taxonomy because it has an extensive history and has received significant scrutiny. In future work, we propose to evaluate a larger range of learning taxonomies to identify the best fit for our research question. These include the Structure of Observed Learning Outcomes (SOLO) taxonomy, which categorizes types of understanding [4], and Fink's taxonomy, which differs from other taxonomies by being non-hierarchical and by focusing on meta-cognition, regulation of the self, social interaction, and caring [13].

Bloom's Taxonomy was originally developed as a means to construct educational testing banks from test items shared across universities. The taxonomy proposes that human learning and performance is hierarchical and consists of six cognitive process dimensions that span from the lowest level, *remember*, to the highest level, *create*, which are reprinted, here for convenience from Krathwohl (2002) [23]:

1) ***Remember*** – Retrieving relevant knowledge from long-term memory

2) ***Understand*** – Determining the meaning of instructional messages, including oral, written, and graphic communication

3) ***Apply*** – Carrying out or using procedures in a given situation

4) ***Analyze*** – Breaking material into its constituent parts and detecting how the parts relate to one another and to an overall structure or purpose

5) ***Evaluate*** – Making judgments based on criteria and standards

6) ***Create*** – Putting elements together to form a novel coherent whole or make an original product

We believe that valid patterns should be situated at the level of application (level 3), wherein patterns that are easily adapted to a problem description to yield a solution require no more effort than the ability to retrieve relevant knowledge, understand instructional messages, and apply procedures to the given situation (a problem description). However, we observe that many security requirements remain situated at the levels of analyze, evaluate and create (levels 4 through 6), which require decomposing the problem space and the security requirements into a common mental model, introducing new expert knowledge to complement and refine the problem description and requirement, and recombining the result into a new solution. We believe the challenge of creating a valid pattern is identifying the knowledge that fits a sufficiently broad collection of problem descriptions and presenting this knowledge in a format that lends itself to application as opposed to additional analysis. We recognize that some patterns may require analysis, regardless.

We believe that patterns make extensive use of abstractions, which diminish or inhibit some properties of real-world things in the domain in order to emphasize other properties that recur, presumably, those properties to which the pattern explicitly refers [14]. Thus, as a problem recurs, the recurring features shared across problem instances correspond to cues that the analyst may recognize match one of their patterns. Furthermore, the pattern describes the solution space using abstract variables: the analyst must identify new information to assign values to these variables. In natural language patterns, these "variables" are broad statements of what to do, e.g., substitute one agent for another agent [24, 38], whereas in syntactic templates, these variables are literal and the analyst need only identify valid phrases to assign to the corresponding slots in the sentence pattern. In cognitive psychology, category theory research explains how humans interact with abstractions. In the following discussion, we equate categories with patterns: as

[2] Disclaimer: Because of the various pattern types, Bloom's Taxonomy may not cover the complete set of relevant skills, e.g., "soft" skills required to engage different personalities during elicitation.

categories contain the objects that they describe, patterns match the situations (or problems) that they can solve. We aim to draw inspiration for category theory in the design of pattern experiments.

"What is the range of input descriptions, and what features from an input description increase or decrease validity?" is a research question informed by how cognitive psychologies define abstraction. Eleanor Rosch, a prominent category theorist, defines *level of abstraction* in two dimensions (see Figure 3): the horizontal dimension concerns the level of inclusiveness of real world objects in a category, wherein the basic level is the most inclusive; and the vertical dimension concerns the segmentation of categories at the same level of inclusiveness [33]. In Figure 3, the object A is the most inclusive (closest to x=0 along the x axis) and thus serves as the basic level of abstraction; contrast this object with objects B and C, which sit to the right along the x-axis. To increase the flexibility and distinctiveness of categories in the horizontal dimension, a category is defined in terms of "the attributes most representative of items inside and least representative of items outside the category" [33]. Thus, object D has attributes that are least representative of objects A, B, and C; these attributes serve to define the boundary of the category. Segmentation, or the vertical dimension, may be discerned by the dissimilarity of attributes between concepts at the same horizontal level of inclusiveness. Thus, objects B and C share attributes that include them in the category, but also have attributes that segment them, vertically. In Rosch's model, the objects B and C may correspond to a kitchen and dining room chair: both have the same functional property (sitness or sitability), but are associated with different contexts, one informal (the kitchen) and one formal (the dining room). The object A has a more general concept, such as a chair, and D is outside the category, such as a door.

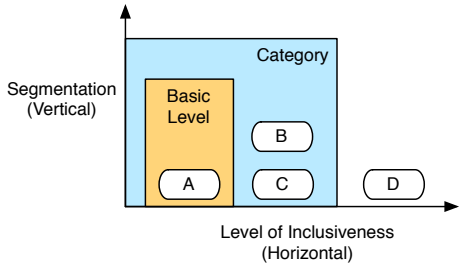


Figure 3. Rosch's Dimensions of Abstraction

In pattern research, we believe that Rosch's dimensions of abstraction may explain how pattern users select patterns from problem cues – the task of pattern selection. Cues that yield high precision (few false positives) and high recall (few false negatives) in pattern selection are preferable to improve pattern validity. Ambiguous cues may indicate input descriptions that are either not relevant to the pattern, or they cannot be solved by the pattern. These cues may decrease precision by introducing false positives. Cues that are difficult to recognize by the population of pattern users across different problem instances will lower recall. To determine when a cue is effective, we now return to Rosch.

Rosch formalizes the vertical dimension (segmentation) as *cue validity*, which is the probability that a cue x is associated with a category y, or $P(y \mid x)$. Categories with high cue validity are more distinct from other categories. For requirements patterns, an experimenter might manipulate terminology used to describe the pattern by abstracting terms or selecting terms with higher cue validity with the aim of improving the analyst's ability to match the pattern to problem description cues. Moreover, parts of a pattern that have low cue validity (e.g., they are less representative of all solutions) may be extracted into variable parts of the pattern to accommodate variation in the solution space. Category theory also defines exemplars, which are existent members of a category previously encountered, and prototypes, which are graded notions of a category [33]. Exemplars aid pattern users by providing examples of the central tendency of a pattern. Two exemplars presented side-by-side have been shown to significantly improve learning of the category more than showing two exemplars in sequence [8].

The research question, "*What can we change in a pattern to improve perceptions of applicability, or improve the transformation from problem to solution?*" concerns how we manipulate pattern representations to affect validity. Egon Weigl experimentally discovered *points-of-shift*, which are cues that lead people to transition from one categorical perspective to another [39]. This notion denotes that multiple categories or abstractions can simultaneously co-exist for a collection of objects and that each category frames how a person groups objects into shared categories. In addition to points-of-shift, which were manipulated experimentally, Weigl also discovered "shocks" that could draw a person's attention away from a situation, entirely. Points-of-shift may be designed into a requirements pattern to reduce the number of overlapping concerns: multiple concerns, such as simultaneously managing structural and behavioral concerns, may overwhelm the analyst, and thus require analysis and evaluation as opposed to application, or introduce shocks that distract from the problem at hand and cause a failure to apply the pattern.

V. Discussion and Summary

In this paper, we describe preliminary results towards developing a framework for pattern experimentation. Because of the broad scope of what many industry and academic authors mean by the term pattern, we conducted a literature review to ascertain why patterns are used, what sources of knowledge patterns are derived from, and what types of patterns exist within a small search of requirements engineering research papers (limited to one conference's proceedings over the past 19 years). We also explore research findings from cognitive psychology that we believe can inform how we evaluate pattern validity. While these findings are preliminary, we believe these results can help other pattern experimenters to frame their patterns using our taxonomy (in its present form or by extension or revision) and lead researchers to design experiments that test variations on their patterns.

As part of future work, we plan to extend our literature review to include other sources, including workshop and conference proceedings and journals. We regret that we had to exclude some interesting and relevant pattern results that we found in short papers, posters, and workshop papers due to our strict exclusion criteria. In addition, some papers use similar notions to patterns, but call these patterns by different names (e.g., problem frames [19]). This shortfall will be remedied in future work. There are also other pattern discussions in the software engineering field such as the Pattern Languages of Programs (PLoP) community efforts that we would like to consider. In addition, we plan to further extend our analysis of cognitive psychology findings to refine our experimental framework with the aim of validating the framework in a series of pilot studies.

ACKNOWLEDGMENT

This research was funded by the Army Research Office - (W911NF-09-1-0273).

REFERENCES

[1] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S., *A Pattern Language*, Oxford University Press, 1977.

[2] Araújo, J., Wittle, J., Kim, D-K. "Modeling and composing scenario-based requirements with aspects," *12th IEEE Int'l Req'ts Engr. Conf.*, pp. 58- 67, 2004.

[3] Bachmann, F., Bass, L., Klein, M. "Deriving architectural tactics: a step toward methodical architectural design," *Software Engineering Institute Technical Report #CMU/SEI-2003-TR-004*, March 2003.

[4] Biggs, J.B., Collis, K.F. "Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)" *Academic Press New York*, 1982.

[5] Breaux, T.D., Anton, A.I., Boucher, K., Dorfman, M. "Legal requirements, compliance and practice: an industry case study in accessibility," *16th IEEE Int'l Req'ts Engr. Conf*., pp. 43-52, 2008.

[6] Breaux, T.D., Baumer, D.L. "Legally "reasonable" security requirements: A 10-year FTC retrospective," *Computers & Security* 30(4): 178-193, 2011.

[7] Bright, P. "RSA finally comes clean: SecurID is compromised," *ArsTechnia*, June 6, 2011.

[8] Christie, S., Gentner, D. "Where hypotheses come from: learning new relations by structural alignment." *Journal of Cognition and Development*, 11(3): 356-373, 2010.

[9] Cleland-Huang, J., Chang, C.K., Kim, H., Balakrishnan, A. "Requirements-based dynamic metrics in object-oriented systems," *5th IEEE Int'l Symp. Req'ts Engr*., pp. 212-219, 2001.

[10] Damian, D., Marczak, S., Kwan, I.. "Collaboration Patterns and the Impact of Distance on Awareness in Requirements-Centred Social Networks," *15th IEEE Int'l Req'ts Engr. Conf.,* pp. 59-68, 2007.

[11] Diaz, I., Pastor, O., Matteo, A. "Modeling interactions using role-driven patterns," *13th IEEE Int'l Req'ts Engr. Conf*., pp. 209-218, 2005.

[12] Dardenne, A., Van Lamsweerde, A., Fickas, S. "Goal-directed requirements acquisition," *Science of Computer Programming*, 20(1-2): 3-50, 1993.

[13] Fink, L.D. "Creating significant learning experiences: An integrated approach to designing college courses" *Jossey-Bass Inc Pub*, 2003.

[14] Fisher, S.C. "The process of generalizing abstraction; and its product, the general concept" *Psychological Review Company*, 11(2): 1916.

[15] Gaska, M.T., Gause, D.C. "An approach for cross-discipline requirements engineering process patterns," *3rd Int'l Conf. Req'ts Engr*., pp. 182-189, 1998.

[16] Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N. "Modeling security requirements through ownership, permission and delegation," *13th IEEE Int'l Req'ts Engr. Conf*., pp. 167-176, 2005.

[17] Hagge, L., Lappe, K. "Patterns for the RE process," *12th IEEE Int'l Req'ts Engr. Conf*., pp. 90-99, 2004.

[18] Jackson, M. "The world and the machine," *IEEE Int'l Conf. Soft. Engr.*, 1995, pp. 283-292.

[19] Jackson, M. *Problem Frames: Analysing and Structuring Software Development Problems*, Addison-Wesley, 2001.

[20] Kartseva, V., Gordijn, J., Tan, Y-H. "Value-based design of networked enterprises using e3 control patterns," *15th IEEE Int'l Req'ts Engr. Conf*., pp. 91-100, 2007.

[21] Konrad, S., Cheng, B.H.C. "Requirements patterns for embedded systems," *13th IEEE Int'l Req'ts Engr. Conf.,* pp. 127-136 2002.

[22] Konrad, S., Cheng, B.H.C. "Facilitating the Construction of Specification Pattern-based Properties," *13th IEEE Int'l Req'ts Engr. Conf*., pp. 329-338 2005.

[23] Krathwohl, D.R. "A revised Bloom's taxonomy: an overview," *Theory Into Practice*, 41(4): 212-218, 2002.

[24] Lutz, R., Nelson, S., Patterson-Hines, A., Frost, C.R., Tal, D. "Identifying contingency requirements using obstacle analysis," *13th IEEE Int'l Req'ts Engr. Conf*., pp. 263- 272 2005.

[25] Lutz, R., Mikulski, I.C., "Resolving requirements discovery in testing and operations," *11th IEEE Int'l Req'ts Engr. Conf*., pp. 33-41, 2003.

[26] Mylopoulos, J., Chung, L., Nixon, B. "Representing and using nonfunctional requirements: a process-oriented approach," *IEEE Trans. Soft. Engr.*, 18(6): 483-497, 1992.

[27] Marczak, S., Damian, D. "How interaction between roles shapes the communication structure in requirements-driven collaboration," *19th IEEE Int'l Req'ts Engr. Conf*. pp. 47-56, 2011.

[28] Marczak, S., Damian, D., Stege, U., Schroter, A. "Information brokers in requirement-dependency social networks," *16th IEEE Int'l Req'ts Engr. Conf*., pp. 53-62, 2008.

[29] Massonet, P., van Lamsweerde, A. "Analogical reuse of requirements frameworks," *3rd IEEE Int'l Symp. Req'ts Engr*., pp. 26-37, 1997.

[30] Patton, M.Q.. "Qualitative research and evaluation methods" *Sage Publications Inc*, 2002.

[31] Rawsthorne, D.A. "Capturing functional requirements through object interactions," *2nd IEEE Int'l Conf. Req'ts. Engr*., pp. 60-67, 1996.

[32] Ridao, M., Doorn, J., Do Prado Leite, J.C.S., A. "Domain independent regularities in scenarios," *5rd IEEE Int'l Symp. Req'ts Engr.*, pp. 120-127, 2001.

[33] Rosch, E. "Principles of Categorization," Rosch, E., Lloyd, B.B. (eds), *Cognition and categorization*, Lawrence Erlbaum, pp. 27-48, 1978.

[34] Rolland, C., Plihon, V. "Generic method chunks to generate process models fragments," *2nd IEEE Int'l Conf. Req'ts. Engr.,* pp. 173-180, 1996.

[35] Shaw, M., Garvin, D. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[36] Sidel, R. "Card processor: hackers stole account numbers," Wall Street Journal, April 2, 2012.

[37] Supakkul, S., Hill, T., Chung, L., Tun, T.T., Sampaio do Prado Leite, J.C., "An NFR pattern approach to dealing with NFRs," *18th IEEE Int'l Req'ts Engr. Conf.,* pp. 179-188, 2010.

[38] van Lamsweerde, A., Letier, E. "Handling obstacles in goal-oriented requirements engineering," *IEEE Trans. Soft. Engr.,* 26(10): 978-1005, 2000.

[39] Weigl, E. *The Psychology of So-called Processes of Abstraction,* Translated by Rioch, M.J., Edited by Landis, C., Goldstein, K. 1927

[40] Zetter, K. "How digital detectives deciphered Stuxnet, the most menacing malware in history," *Wired Magazine*, July 11, 2011