

Improving Analysis Pattern Reuse in Conceptual Design: Augmenting Automated Processes with Supervised Learning

Sandeep Puro • Veda C. Storey • Taedong Han

School of Information Sciences and Technology, The Pennsylvania State University, University Park, State College, Pennsylvania 16802

Department of Computer Information Systems, J. Mack Robinson College of Business, Georgia State University, Box 4015, Atlanta, Georgia 30302

Department of Management Information Systems, University of Nevada, Las Vegas, Nevada 89154
spurao@ist.psu.edu • vstorey@gsu.edu • than@unlv.edu

Conceptual design is an important, but difficult, phase of systems development. Analysis patterns can greatly benefit this phase because they capture abstractions of situations that occur frequently in conceptual modeling. Naïve approaches to automate conceptual design with reuse of analysis patterns have had limited success because they do not emulate the learning that occurs over time. This research develops learning mechanisms for improving analysis pattern reuse in conceptual design. The learning mechanisms employ supervised learning techniques to support the generic reuse tasks of retrieval, adaptation, and integration, and emulate expert behaviors of analogy making and designing by assembly. They are added to a naïve approach and the augmented methodology implemented as an intelligent assistant to a designer for generating an initial conceptual design that a developer may refine. To assess the potential of the methodology to benefit practice, empirical testing is carried out on multiple domains and tasks of different sizes. The results suggest that the methodology has the potential to benefit practice.

(Reuse; Design Automation; Object-Oriented Systems; Learning Mechanisms; Conceptual Design; Analysis Patterns; APSARA; Software Development)

1. Introduction

Conceptual design is an important, but difficult, phase of systems development. It requires capturing user requirements in a representation independent of the implementation platform. This involves understanding and modeling users' requirements—a task that calls for considerable expertise because these requirements can be incomplete and difficult to articulate (Batra 1993). Efforts to improve conceptual design, therefore, have the potential to be extremely valuable. Research in software engineering suggests that *reuse* is the most effective approach to improving

software development (Krueger 1992). Reuse involves the design of new systems from existing artifacts or higher-level specifications¹ (Setliff et al. 1993), which can both shorten the software development life cycle and reduce overall development costs (Gibbs 1994,

¹ Frakes and Fox (1995) define reuse as the use of existing software artifacts or knowledge to build new software artifacts. The reusable elements may be code, requirements, design specifications, or knowledge of the domain, and may also include processes, methods, templates, development experience, design decisions, architectural structures, and documentation (Karlsson 1995, Biggerstaff and Perlis 1989).

Szyperski 1998). Integrating reuse into conceptual design can, however, be particularly challenging.

Several types of artifacts have been developed for reuse during conceptual design. Among them are reified exemplars such as components and business objects (Szyperski 1998) and codified knowledge such as analysis patterns (Appleton 1997). An analysis pattern represents a group of related, generic objects with stereotypical attributes and behaviors (Coad et al. 1995, Fowler 1997) that developers can apply in different domains (Alexander 1977, Lea 1994) to create new designs. Analysis patterns are particularly useful for conceptual design because they provide abstractions of situations that occur frequently, allowing developers to reuse chunks of prior knowledge in new situations.

A few naïve approaches to analysis pattern reuse for conceptual design have been proposed (Purao and Storey 1997, Wohed 2000a). They include performing a lookup in the available examples to identify relevant patterns, and connecting common objects across these patterns to create a new conceptual design. These approaches employ existing libraries of analysis patterns (Coad et al. 1995, Fowler 1997), but do not *learn* from experience as a developer would as he or she moves from one design to the next. Expert designer behaviors, such as making analogies and designing by assembling existing components, are therefore difficult to emulate on a sustained basis. Such naïve approaches could be improved significantly by incorporating learning. We envisage a semi-automated assistant that implements a reuse-based design approach that incorporates learning. Such an assistant can quickly create a preliminary conceptual design, with minimal guidance from the human developer, who can then refine the automatically generated conceptual design. The objectives of this research, therefore, are to:

- Develop a methodology to incorporate learning into automated reuse of analysis patterns during conceptual design and
- Evaluate the potential of the proposed methodology to benefit practice by assessing the contribution of the learning mechanisms through rigorous empirical testing.

The remainder of this paper is organized as follows. Section 2 reviews related research, §3 develops learning mechanisms that form the basis of our

methodology for reuse-based conceptual design, §4 describes the training of the learning mechanisms, along with an initial test of feasibility. In §5 we present results from empirically testing a prototype that implements the methodology, and §6 discusses implications of the results. Concluding remarks appear in §7.

2. Background and Related Research

This section defines analysis patterns, explains how their reuse can aid conceptual design by mimicking expert designer behaviors, and discusses how learning can better emulate these behaviors to improve reuse-based conceptual design.

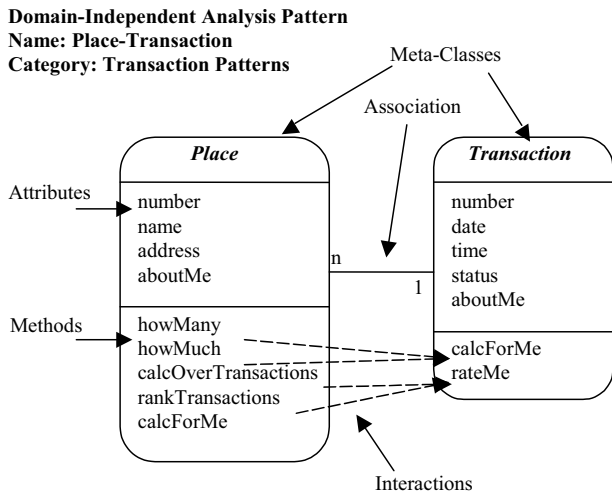
2.1. Analysis Patterns

An analysis pattern is a group of related, generic objects (meta-classes) with stereotypical attributes (data definitions), behaviors (method signatures), and expected interactions defined in a domain-neutral manner (Coad et al. 1995, Fowler 1997). For example, the analysis pattern PLACE-TRANSACTION, shown in Figure 1, contains two meta-classes—PLACE and TRANSACTION (Coad et al. 1995), each with generic attributes. *Date*, for example, is an attribute of TRANSACTION but not of PLACE. The behaviors also differ, with five specified for PLACE and two for TRANSACTION. The dotted lines indicate likely interactions between classes. The pattern can be instantiated as *bank-deposit*, *warehouse-shipment*, or *store-sale*, depending upon the application domain.

Analysis patterns resemble the notion of chunks of formalized knowledge² that are at a higher level of abstraction than individual classes (Appleton 1997). Identification of analysis patterns involves the creation of domain-independent abstractions (Buschmann et al. 1996). Because these abstractions represent frequently

²Such formalized knowledge can consist of organized patterns of information (i.e., *chunks*) (Ericsson and Staszewski 1989) that has been linked with expertise. Casakin and Goldschmidt (1999), for example, suggest that as expertise develops this knowledge becomes more structured and better integrated with past experiences. So it can be retrieved from memory in larger chunks.

Figure 1 An Analysis Pattern



Typical Object Interactions:

howMany → calcForMe calcOverTransactions → calcForMe
 howMuch → calcForMe rankTransactions → rateMe

Examples:

Place: airport, assembly line, bank, clinic, depot, garage, geographic entity, hangar, hospital, manufacturing site, plant, region, sales outlet, service center, shelf, station, store, warehouse, zone.
 Transaction: agreement, assignment, authorization, contract, delivery, deposit, incident, inquiry, order, payment, problem, report, purchase, refund, registration, rental, reservation, sale, shift, shipment, subscription, time charge, title, withdrawal.

Combinations:

Participant-Transaction; Specific Item-Transaction; Transaction-Transaction Line Item; Transaction-Subsequent Transaction.

occurring problem situations, analysis patterns can minimize problem-solution mismatches during reuse (Maiden and Sutcliffe 1993), and can be effective during earlier phases in the development life cycle (Mili et al. 1995). Although patterns³ have been created for different phases of software development (e.g., design patterns, Gamma et al. 1995), this research focuses on analysis patterns (Coad et al. 1995, Fowler 1997) for conceptual design.

2.2. Conceptual Design with Reuse of Analysis Patterns

Creating a conceptual design with reuse of analysis patterns requires significant adjustments to the

³ The idea of a pattern can be traced to Alexander (1964), who originally identified patterns that could be assembled to design building environments. His ideas have since been applied to software design (Gabriel 1996), with ongoing work on pattern languages, discovery, and documentation (Harrison et al. 2000).

generic reuse tasks of retrieval, adaptation, and integration (Prieto-Diaz 1993). First, retrieval requires that the domain-independent analysis patterns be interpreted to determine if they are applicable to the application domain (Frakes and Nejme 1990). Second, adaptation requires that the retrieved analysis patterns be adjusted for the application domain (Johannesson and Wohed 1999). Finally, integration requires that the adapted analysis patterns be connected to create a conceptual design (Fowler 1997) (a class diagram that is a synthesis of instantiated analysis patterns). A *mechanistic* view of these three steps suggests that reuse involves decomposing the problem to create slots into which available patterns can be fitted. This view resembles the set cover problem (NIST 2002), confirming that reuse-based design with analysis patterns is a complex problem (Mili et al. 1995).

A *human-developer-focused* view of the reuse process, on the other hand, reveals two challenges to reuse-based design with analysis patterns. First, reuse of analysis patterns in a new domain requires *analogy making* (Gentner 1983), which means the developer must understand a solution created *by someone else* to evaluate its applicability in the application domain. The analogy making task requires knowledge about (1) the analysis patterns, (2) the application domain, and (3) the way that the analysis patterns map against requirements specified for the application domain. Second, as the problem gets larger, analogy making is not sufficient. Partial solutions must be assembled to obtain larger solutions. Such *design by assembly* requires intelligent combining of patterns. These two challenges can be compared with documented expert design behaviors. Expert developers use standard abstractions (Batra and Davis 1992, Cole 1987), similar to analysis patterns, and perform reasoning by analogy (Prietula and Simon 1989). They also use pattern-oriented mental models (Rouse and Morris 1986) that they combine as needed (Storey et al. 1995). Table 1 compares expert design behaviors to opportunities for reuse provided by analysis patterns. A human-developer-focused view, therefore, must support analogy making and designing by assembly to take advantage of the reuse opportunities provided by analysis patterns.

Table 1 Mapping Expert Design Behaviors Against Analysis Pattern Reuse Opportunities

Expert Design Behaviors	Analysis Pattern Reuse Opportunities
Use of standard abstractions (Batra and Davis 1992)	Reuse of generic solutions in multiple domains (Buchanan and Mennier 1996)
Abstraction of real-world situations (Cole 1988), and reasoning by analogy (Prietula and Simon 1989)	Reuse at a higher level of abstraction (Appleton 1997), minimizing problem-solution space mismatch (Maiden and Sutcliffe 1993)
Use of pattern-oriented mental models (Rouse and Morris 1986, Storey et al. 1995)	Potential for compositional reuse earlier in the development life cycle (Mili et al. 1995)

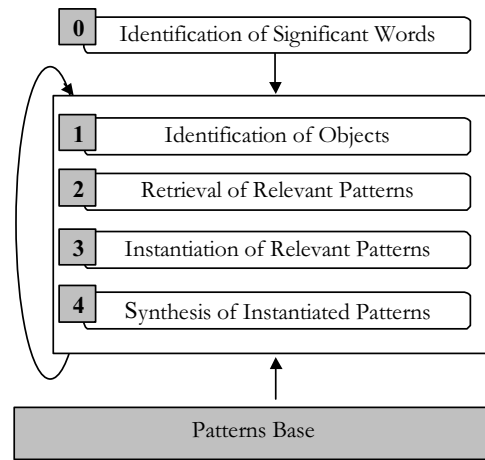
In spite of the potential to improve reuse-based conceptual design with analysis patterns, only two approaches have been proposed for reuse of analysis patterns⁴ (Purao and Storey 1997, Wohed 2000a). Both approaches are naïve in that they (1) perform a simple lookup in the examples available to identify relevant patterns and (2) connect common objects across the patterns to create a conceptual design. They either follow or imitate the three generic reuse steps of retrieval, adaptation, and integration (Prieto-Diaz 1993). For example, Wohed (2000a) implements a domain-specific process that focuses on retrieval, but assumes the adaptation and integration steps. She describes an automated system that interviews users to obtain answers, which are mapped to a set of patterns that have previously been instantiated and connected in a *booking* application domain. The approach by Purao and Storey (1997) implements a domain-independent approach⁵ that uses simple natural language assertions such as “a system to track sales at different stores” as inputs. The repository of patterns by Coad et al. (1995) is used to retrieve, instantiate, and synthesize analysis patterns to assemble a preliminary conceptual design (Figure 2).

These naïve approaches demonstrate how expert designer behaviors may be mimicked by an automated

⁴ Other approaches to pattern reuse have focused on the design and implementation stages, not the conceptual design stage (Florijn et al. 1997, Bansiya 1998).

⁵ Their approach follows the tradition of research in automated conceptual modeling (see, for example, Bubenko and Wangler 1992, Lowry and McCartney 1991).

Figure 2 Naïve Reuse-Based Design



approach. Neither approach can adjust its solutions based upon prior experiences, which limits their sustained usefulness. Further, they demonstrate that naïve pattern retrieval can lead to problems such as missing opportunities for repeated or recursive use of patterns, and selective superimposition (Purao 1998). These approaches, however, provide useful starting points for investigating how learning can improve analysis patterns reuse during conceptual design.

2.3. Improving Analysis Pattern Reuse with Learning

Simon (1981) defines learning as “changes to a system that...enable it to do the same task or tasks drawn from the same population more efficiently and more effectively the next time” (p. 28). This broad definition of learning can be represented as a generic learning model with four components: *learning mechanisms*, *performance element*, *critic*, and *problem generator* (Nilsson 1998). The learning mechanisms track prior experiences to improve the performance element. A critic confirms or rejects improvements suggested by the learning mechanisms. The problem generator corresponds to prior experiences that feed the learning mechanisms. For learning in analysis patterns reuse, the performance element is the existing (naïve) reuse process outlined above. The developer is the critic and the problem generator is the previous applications. Learning mechanisms, then, need to be developed to

improve the performance element based upon previous applications and feedback from the critic.

There are, unfortunately, significant obstacles that intervene in a straightforward translation of the four components. The most important is the nature of the performance element: “conceptual design of information systems with reuse of analysis patterns.” The retrieval, adaptation, and integration phases (Prieto-Diaz 1993) can be further decomposed into multiple steps (Purao and Storey 1997). These steps are clearly cumulative, and interdependent. The conceptual design task is, therefore, different from a simple, single-step task,⁶ where a goal condition is easily identified, and the learning mechanism can adapt its behavior to achieve this goal. In those cases, learning can occur by specifying rewards that are tied to an outcome—reinforcing good decisions and penalizing bad ones (Kaelbling et al. 1996). For example, expert classification systems can learn by continually adjusting weights to distinguish between correct and incorrect decisions (e.g., Bayesian belief revisions similar to Dey and Sarker 2000).

Many practical problems, including this research problem, involve multiple tasks that cannot be stated in the form of a single-step goal. For complex tasks, the final outcome is not known until all decisions are made. Then, a learning strategy must rely on feedback from the critic at each step. The critic can give the learning agent positive or negative rewards (Nilsson 1998, p. 175). The objective for the learning agent, therefore, is to maximize the amount of reward it receives, and the importance of the critic, makes supervised learning⁷ an appropriate strategy. This form of learning adds the human element to the process, suggesting a semi-automated solution. For example, a learning mechanism could suggest a specific analysis pattern, and the developer may confirm

or reject its appropriateness. The learning mechanism would need to adjust the suggestions next time based upon the developer’s answer. The learning mechanism, therefore, would predict future rewards by considering a sequence of earlier actions (Nilsson 1998, p. 177, Barto et al. 1995, Kaelbling et al. 1996) that can be context-dependent.

2.4. Summary of Theoretical Bases

The review of prior research identifies the multiple theoretical bases that contribute to the definition of the research problem, and highlights approaches to addressing it. First, analysis patterns capture the notions of chunking and higher levels of abstractions. Second, reuse of analysis patterns suggests that naïve approaches can be augmented with learning to better emulate expert design behaviors such as analogy making and designing by assembly. Finally, the complexity of design with reuse of analysis patterns suggests that the problem contains a sequence of tasks that would best be addressed by a supervised learning strategy. The next section develops these ideas further to identify features that must be part of a methodology for incorporating learning into analysis patterns reuse.

3. Incorporating Learning into Analysis Patterns Reuse

Based upon the review of prior research, the goal of incorporating learning into analysis patterns reuse can be articulated as effective emulation of the expert design behaviors of analogy making and designing by assembly. Emulating analogy making requires learning mechanisms that take advantage of the features of analysis patterns that support the retrieval and adaptation reuse phases. Emulating designing by assembly requires learning mechanisms that exploit features of analysis patterns that support the integration reuse phase. To address these challenges, 15 learning mechanisms were developed to improve tasks outlined in a naïve approach (Purao and Storey 1997). These are summarized in Table 2, with details given in Appendix A. The learning mechanisms use reinforcement learning, a subclass of supervised learning, to

⁶ For example, a simple risk assessment classification may involve the decision to grant or deny credit (Basel 1999).

⁷ Briscoe and Caelli (1996) classify learning mechanisms into several categories: symbolic empirical learning (e.g., decision trees, inductive logic, supervised learning, unsupervised learning, conceptual clustering), analytical/explanation-based learning (e.g., composite rules, search control knowledge), case-based reasoning (e.g., analogical reasoning, exemplars), and statistical learning (e.g., Bayesian learning, neural networks).

Table 2 Mapping Learning Mechanisms Against Key Challenges

PHASE/Task/Learning Mechanism	Challenges Addressed		
	Preparing for Analogy Making	Simulating Analogy Making	Supporting Designing by Assembly
PHASE: RETRIEVAL			
Task: requirements parsing			
Discard	✓		
Shield	✓		
Guide	✓		
Task: object retrieval			
Competing objects		✓	
Competing words		✓	
Task: pattern retrieval			
Direct instantiation		✓	
Pattern affinity		✓	
Pattern indifference		✓	
PHASE: ADAPTATION			
Task: object and pattern instantiation			
Partial instantiation affinity		✓	
Inverse		✓	
Instantiation affinity		✓	
PHASE: INTEGRATION			
Task: synthesis of instantiated patterns			
Synthesis by object			✓
Synthesis by keyword			✓
Task: graph spreading			
Spreading by keyword		✓	✓
Spreading via recursion		✓	✓

Note. Phases (Prieto-Diaz 1993), Tasks (Purao and Storey 1997).

take into account the important role of the critic in the complex conceptual design task.

Reinforcement learning generates suggestions that are based on decisions by the developer in prior situations. The canonical form of reinforcement learning is operationalized by a number of mechanisms. Other learning mechanisms capture two variations of reinforcement learning: statistical and inductive (Muggleton and De Raedt 1994). Statistical learning selects from among multiple possibilities to generate suggestions. Inductive learning uses the structure inherent in analysis patterns to induce suggestions. The learning mechanisms do this by using knowledge available in analysis patterns such as objects, instantiations, and associations (Figure 1), and by using knowledge gained in their use in prior cases.

Table 3 Mapping Learning Mechanisms to Learning Types

Learning Types	Learning Mechanisms
Canonical reinforcement learning	Discard, shield, guide, direct instantiation, pattern affinity, pattern indifference, inverse, partial instantiation affinity, instantiation affinity, synthesis by object, spreading by recursion, spreading by keyword
Reinforcement learning: statistical learning	Competing objects, competing words
Reinforcement learning: inductive learning	Synthesis by keyword

Table 3 summarizes these learning mechanisms by types of learning.

The learning mechanisms, therefore, represent a systematic effort to operationalize the dual objectives of analogy making and designing by assembly by instantiating a theoretically grounded set of learning mechanisms that draw on reinforcement learning, and by using knowledge at different levels of abstraction in the analysis patterns, and by using the usage history. Each learning mechanism was formally stated to ensure appropriate operationalization. An example is shown in Figure 3.

To ensure appropriate enactment of these learning mechanisms, two further components are crucial: evidence accumulation and decision rules (Figures 4A and 4B, respectively). The first refers to how evidence such as prior experience and feedback from the critic is captured. The second refers to how this evidence is used in the form of threshold levels in the decision rules. Figure 4A shows the algorithm to accumulate prior evidence, which requires recording occurrences of each acceptance or rejection by the developer. Figure 4B gives the decision rules, i.e., the rule for making suggestions based on whether the accumulated evidence meets the threshold. Together, the algorithms and decision rules enable accumulation of usage history and its use for various types of learning. An important consideration is the contextual or domain-dependent nature of these two tasks (Briscoe and Caelli 1996), because the application of (domain-independent) analysis patterns can be different in different domains. Evidence must, therefore,

Figure 3 An Example Learning Mechanism: Partial Instantiation Affinity

Formal Statement	Explanation
$\forall pr \in [\text{Patterns Retrieved} \mid \text{current}] \mid$ $pr \equiv p \in [\text{Patterns}] \mid \exists o \in p \mid$ $o \text{ instantiated} \in pr \wedge o' \in p \mid \text{is not instantiated} \in pr$ $\forall pi \in [\text{Pattern Instantiations} \mid \text{learned}] \mid pi \equiv p \in [\text{Patterns}] \mid$ <p style="margin-left: 2em;">If $\forall o \in p$</p> <p style="margin-left: 4em;">$\exists o. \text{Instantiation} \in pr = o. \text{Instantiation} \in pi$</p> <p style="margin-left: 2em;">Then $\forall o' \in pr \mid o \neq o'$</p> <p style="margin-left: 4em;">Suggest $o''. \text{Instantiation} \in pi$ as $o'. \text{Instantiation} \in pr$</p> <p style="margin-left: 6em;">If designer concurs,</p> <p style="margin-left: 4em;">Instantiate $o' \in pr$ as $o'. \text{Instantiation} \in pi$</p>	If a pattern retrieved for the current design contains objects that are not instantiated, look up instantiations of that object in prior designs and suggest these to the designer.

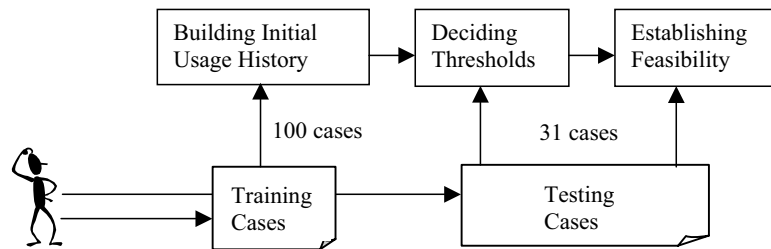
Figure 4A Base Algorithm for Evidence Accumulation

$c_i \in C$ Prior Cases, $d_i \in D$ Contexts (Captured as Application and Industry Domain) $s_j \in S$ Possible States, $a_k \in A$ Possible Actions suggestion $(a_k, s_j \mid d_i)$ Action a_k is suggested at state s_j in context d_i $\{0,1\}$ compatible (a_k) Assertion a_k is compatible with analysis patterns following Table 1 acceptance $(a_k, s_j \mid d_i)$ Suggested action a_k is accepted at state s_j in context d_i $\{0,1\}$ reinforced $(a_k, s_j \mid d_i)$ Potential reward for suggesting action a_k at future states s_j in context d_i $\forall c_i \in C$ Do $\quad \forall d_i \in D$ Do $\quad \quad \text{If } s_j \supset c_i$ $\quad \quad \quad \text{If suggestion } (a_k, s_j \mid d_i) \wedge \text{acceptance } (a_k, s_j \mid d_i)$ $\quad \quad \quad \quad \text{reinforced } (a_k, s_j \mid d_i) \leftarrow \text{reinforced } (a_k, s_j \mid d_i) + 1$ $\quad \quad \quad \text{If suggestion } (a_k, s_j \mid d_i) \wedge \neg \text{acceptance } (a_k, s_j \mid d_i)$ $\quad \quad \quad \quad \text{reinforced } (a_k, s_j \mid d_i) \leftarrow \text{reinforced } (a_k, s_j \mid d_i) - 1$
--

Figure 4B Decision Rules

Case 1: Canonical Reinforcement Learning $\text{If } s_{\text{current}} = s_j$ $\quad \text{Set suggestion } (a_k, s_j \mid d_i) \leftarrow 1 \mid \text{reinforced } (a_k, s_j \mid d_i) > \text{threshold}$ Case 2: Reinforcement Learning with Statistical Learning $\text{If } s_{\text{current}} = s_j$ $\quad \text{Set suggestion } (a_k, s_j \mid d_i) \leftarrow 1 \mid \text{reinforced } (a_k, s_j \mid d_i) = \max$ $\quad \quad \text{reinforced } (a_n, s_j \mid d_i),$ $\quad \quad a_n \in A \wedge \text{reinforced } (a_k, s_j \mid d_i) > \text{threshold}$ Case 3: Reinforcement Learning with Inductive Learning $\text{If } s_{\text{current}} = s_j$ $\quad \text{Set suggestion } (a_k, s_j \mid d_i) \leftarrow 1 \mid \text{reinforced } (a_k, s_j \mid d_i) > \text{threshold} \wedge$ $\quad \quad \text{compatible } (a_k)$

Figure 5 Training the Learning Mechanisms



be accumulated in a manner that allows this domain dependence to be exploited for decision rules.

Thresholds that are part of these decision rules represent an instantiation of reinforcement learning because they use prior cases to predict possible rewards in future cases. For example, a learning mechanism may indicate that the developer considers patterns that occur in at least 60% of prior cases, and may proceed to retrieve these patterns to present to the developer for his or her consideration. The threshold values may be specified by a developer or adjusted, based upon use and the developer's propensity to entertain suggestions. The threshold values may be absolute or relative (e.g., percentage or frequency), and may weigh recent occurrences heavily (e.g., Markov chains).⁸ The discussion above provides the rationale for features that must be implemented to illustrate the feasibility of these ideas.

4. Implementation

A prototype was developed using Java™, which could be used either as a naïve approach (without the learning mechanisms), or as an augmented approach (with the learning mechanisms). The system is capable of logging the usage or turning off this capability to allow for testing, based on the currently available usage history. The patterns library and the usage history are structured as a relational database and the patterns proposed by Coad et al. (1995) used to populate the patterns library. This prototype provides the basis for (1) training the learning mechanisms and (2) demonstrating the feasibility of the research.

⁸ For the purpose of this proof of concept, we have relied on simple thresholds that capture absolute and relative frequencies. The refinements mentioned above may further improve performance of the learning mechanisms.

4.1. Training the Learning Mechanisms

A usage history was built for four industries: Retail, health care, construction, and university; and four application domains (Glass and Vessey 1996): human resources, inventory control, scheduling, and training. Requirement statements were collected from students enrolled in a graduate information systems program. One hundred and thirty-one usable cases were generated. Of these, 100 were randomly selected to build the initial usage history by using the prototype for only the naïve approach as shown in Figure 5. This ensured that the learning from the initial set of cases was unbiased. A significant fraction of the patterns library was exercised by these requirement statements. On average, a pattern was used 23 times, and approximately four new keywords were added for a generic object.⁹

To establish appropriate threshold values, the augmented prototype (with learning mechanisms) was used to assess how the learning mechanisms performed based upon the populated usage history. New test cases, from the 31 available, were used to assess whether the training data allowed the learning mechanisms to clearly separate relevant and irrelevant suggestions. One researcher played the role of the critic to judge whether suggestions from the learning mechanisms were relevant. During these trials, the usage history was held constant. Each learning mechanism was invoked to determine the appropriate threshold values.

Because developer interaction is needed, and because it is easy for a developer to discard spurious suggestions, an inclusive strategy was adopted (allowing Type II errors) to determine the threshold value.

⁹ Approximately 75% of the patterns in the database were used, and 148 new keywords were added.

For example, the threshold value for the learning mechanism Guide was decided on the basis of a minimal threshold of 60% that would result in the inclusion of *all* relevant suggestions (minimizing Type I errors). Threshold values were identified for other learning mechanisms following a similar logic. For the learning mechanisms Pattern Indifference, Inverse, Synthesis by Object, and Spreading by Recursion, zero thresholds were used because they represented situations where even a single occurrence suggests possibilities that would otherwise be overlooked. Table 4 summarizes these thresholds. The threshold values do not capture optimum values, because they are highly dependent upon a developer's willingness to tolerate incorrect suggestions, the application domain, and the available usage history.

4.2. Demonstrating Feasibility of the Research

The augmented approach was also tested with the 31 cases remaining from the initial set of cases to ensure that the augmented approach with learning mechanisms represents a coherent approach to reuse-based design. This testing provided the initial complete test of the feasibility of the research. One of the researchers exercised the prototype several times with the thresholds in Table 4 providing the basis for the invoking of learning mechanisms. An *ideal* model was manually constructed by the researchers for each case. First, classes were identified by examining the requirements statements. Then, the patterns base was consulted to find classes similar to those identified. Using these patterns, more classes were added to the design. Finally, the class names were adjusted, where possible, to reflect the examples in the patterns base.

Table 4 Initial Threshold Values for Learning Mechanisms

PHASE/Task	Learning Mechanism		Threshold
RETRIEVAL/ requirements parsing	Discard	60%	IF observed in 60% or more prior cases THEN suggest as candidate
	Guide Shield		
RETRIEVAL/ object retrieval	Competing objects	20%	IF observed in 20% or more prior cases THEN suggest as candidate
	Competing words		
RETRIEVAL/ pattern retrieval	Direct instantiation	75%	IF observed in 75% or more prior cases THEN suggest as candidate
	Pattern affinity	60%	IF observed in 60% or more prior cases THEN suggest as candidate
	Pattern indifference	Any	IF (not) observed in any prior case THEN suggest as candidate
ADAPTATION/ object and partial instantiation	Pattern instantiation	60%	IF observed in 60% or more prior cases THEN suggest as candidate
	Inverse	Any	IF observed in any prior case THEN suggest as candidate
	Instantiation affinity	60%	IF observed in 60% or more prior cases THEN suggest as candidate
INTEGRATION/ synthesis of instantiated patterns	Synthesis by object	Any	IF observed in any prior case THEN suggest as candidate
	Synthesis by keyword	60%	IF observed in 60% or more prior cases THEN suggest as candidate
INTEGRATION/ graph spreading	Spreading by keyword	60%	IF observed in 60% or more prior cases THEN suggest as candidate
	Spreading by recursion	Any	IF observed in any prior case THEN suggest as candidate

Two of the three researchers codeveloped the ideal models; the third provided independent validation.

The ideal models represent what an expert designer might achieve by incorporating analysis patterns into a design, unaided by the naïve or augmented approaches. Using the ideal model as the benchmark allows us to assess whether the augmented approach can generate results that are closer to those created by experts. The closer to the ideal model, the better the contribution of the learning mechanisms. Figure 6 shows two representative results from this initial testing.

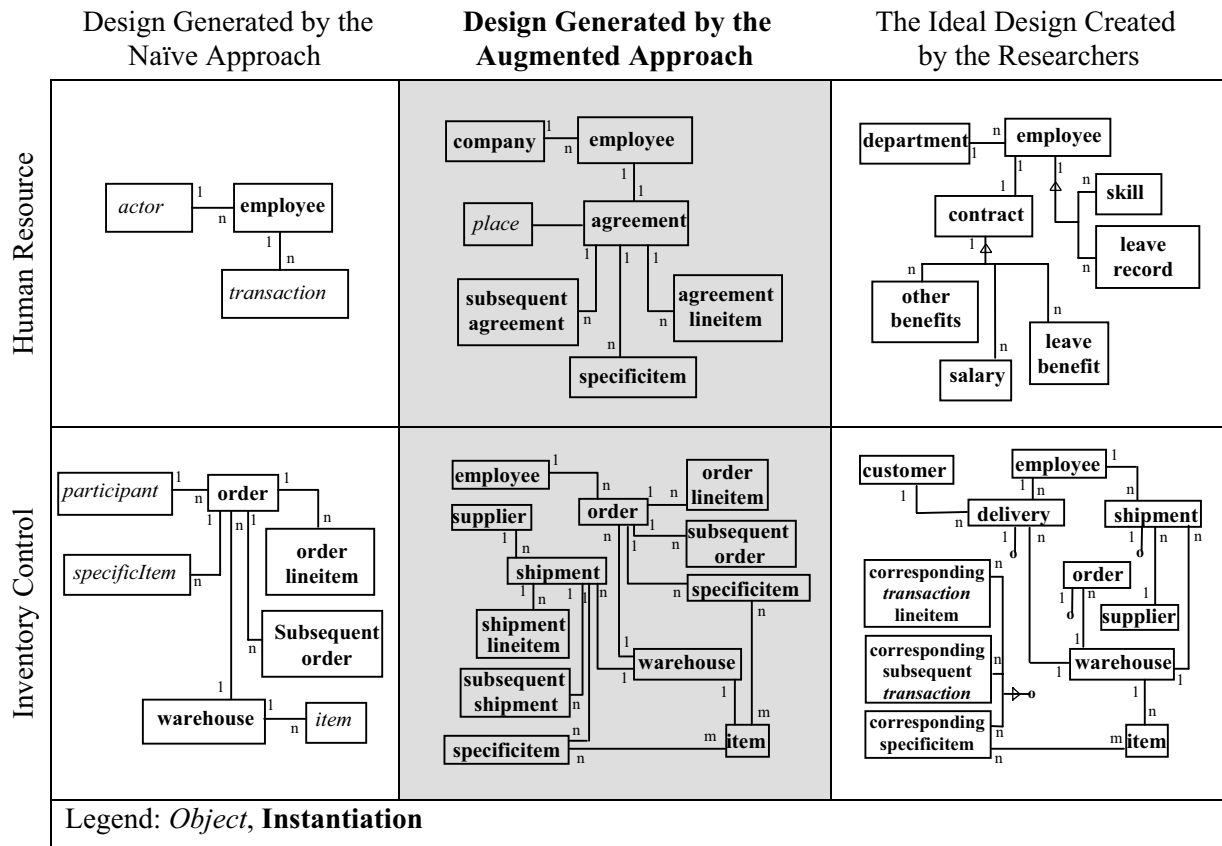
The first column shows designs created using the naïve approach, the second shows design generated (using the *same* requirements) with the augmented approach (with learning), and the third shows the ideal design. For easier reading, the figure shows only classes. Designs generated with the augmented

approach are much closer to the ideal model than those generated by the naïve approach. An empirical investigation was, therefore, considered the essential next step to assess the potential of the proposed methodology.

5. Empirical Evaluation

The objective of the empirical evaluation is to investigate whether the learning mechanisms are effective in producing better designs, on a sustained basis, and thus to understand whether the methodology can potentially benefit practice. The experimental design focuses on the two main challenges to reuse-based conceptual design, i.e., emulating expert design behaviors of (1) simulating analogy making and (2) designing by assembly. Testing the first challenge requires evaluation across multiple domains. The

Figure 6 Comparing Naïve and Augmented Solutions Against the Ideal Design



choice is also supported by the domain-independent nature of analysis patterns, which must be applied in specific domains; and the domain-specific nature of evidence accumulation and the decision rules described earlier.

Testing the second challenge requires evaluation across different sized tasks. This choice is further supported by the relatively small size of analysis patterns that must be combined to create larger applications. By testing across multiple domains and across different sized tasks, the research design implicitly addresses the issue of effectiveness on a sustained basis. The final independent variable is dictated by the nature of reuse support—naïve versus augmented (see Figure 7). For this variable, the naïve approach is considered as *control* and the augmented approach is considered as *treatment* (following Straub et al. 1993) to draw conclusions about the incremental contribution of the learning mechanisms incorporated in the latter. The base research model for the experiment, therefore, is: Errors in Design = f (Design Approach, Domain, Task Size), with design approaches A1—Naïve, A2—Augmented; domains D1—Trained, D2—Untrained but Related; and task size levels S1—Small, S2—Medium, and S3—Moderate. Three core hypotheses were posited (see Table 5).

Figure 7 Research Model

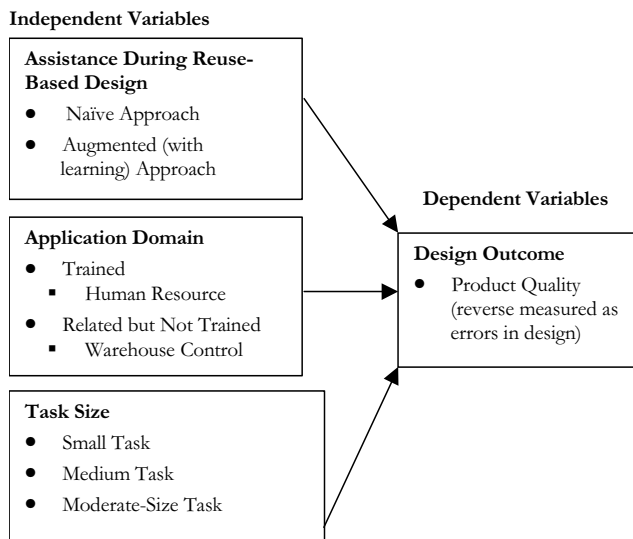


Table 5 Hypotheses Posited

Hypothesis 1	Designs produced with the augmented approach contain fewer errors compared with those produced using the naïve approach. Null: $E(A2) = E(A1)$.
Hypothesis 2	At each size, designs produced with augmented approach contain fewer errors than those produced with naïve approach. Null: $E(A2, S_n) = E(A1, S_n) \mid n = 1, 2, 3$.
Hypothesis 3	For each domain, designs produced with augmented approach contain fewer errors than those produced with naïve approach. Null: $E(A2, D_k) = E(A1, D_k) \mid k = 1, 2$.

Independent Variables. The first independent variable, application domains, uses two alternatives. The first, human resource management (D1), represents a traditional organizational function generally responsible for keeping records such as employee skills and benefits. The learning mechanisms were trained in this domain. The second, warehouse management (D2), represents automation of operations dealing with the receipt, storage, and shipment of goods that can support, in real time, an organization’s activities. The learning mechanisms were *not* trained in this domain, but in a related¹⁰ domain (see §4.1). The second independent variable, task levels, uses three alternatives: Small, medium, and moderate-size, which were developed in each domain (Tasks 2000). The well-accepted metrics, (expected) number of classes (Lorenz and Kidd 1994) and number of classes plus relationships (adapted from de Champeaux 1997) provide a useful determinant of system size (see Table 6).

Dependent Variables. The dependent variable is the quality of conceptual designs. Measuring the quality of conceptual designs is inherently problematic (Genero et al. 2002). The key issues include consistency, correctness, and completeness, which can be difficult to operationalize (Moody et al. 1998). The goal of the learning-augmented methodology, however, is not to create the definitive conceptual design for a set of requirements, but rather to generate a synthesized design by reusing knowledge available in analysis patterns that a human developer can refine. One possible way to assess the quality of a

¹⁰ A related domain was appropriate because an entirely unrelated domain would not reflect the benefits of domain-specific evidence accumulation or decision rules.

Table 6 Task Sizes

Size	Human Resource Management		Warehouse Management	
	Classes	Classes Plus Relationships	Classes	Classes Plus Relationships
Small	Objects: 5	Objects + relationships: 9	Objects: 4	Objects + relationships: 7
Medium	Objects: 9	Objects + relationships: 18	Objects: 6	Objects + relationships: 11
Moderate	Objects: 13	Objects + relationships: 26	Objects: 8	Objects + relationships: 17

Note. Lorenz and Kidd (1994), de Champeaux (1997).

generated conceptual design would, therefore, be to measure it against what an expert designer would create by reusing knowledge available in the analysis patterns. This idea is exemplified by the cases described for demonstrating feasibility (§4.2). Such an assessment can draw on well-established norms of Type I (omission) and Type II (commission) errors. The first addresses completeness, and the second, correctness. The analysis patterns themselves (because of their structure) would provide a preliminary check on the logical consistency. An assessment scheme was devised following this rationale (see Appendix B) to measure the dependent variable—quality of the conceptual design, reverse measured as errors in the design.

To take into account differences in size, the quality scores obtained for each design were normalized (Cohen and Cohen 1983). The use of a ratio measure reflects the fact that the conceptual design task has multiple phases (retrieval, instantiation, synthesis). As the task size increases, so do the number of decisions required at each phase. Assessing the quality of a smaller design (e.g., one that requires five decisions, of which two are incorrect) against a larger one (e.g., one that requires 20 decisions, of which two are incorrect), therefore, requires the use of ratio measures. A ratio scale does not indicate a correlation between the number of errors and the size of the model itself. Rather, with each decision representing a potential source of error, it suggests a correlation between the reuse decisions required and the sources of potential errors.

We expect that the augmented approach, with a human developer involved, will produce a range of models that are different from the one that would be produced by a fully automated, naïve approach. Com-

paring the two allows us to evaluate the incremental contribution of the learning mechanisms.

5.1. Experimental Design and Procedures

A laboratory experiment using a 2 (approach) by 2 (domain) by 3 (task size) was conducted, based upon the research model above, with a *new* set of 35 subjects. The three factors resulted in 12 treatment combinations. The subjects were recruited from IT professionals, who were enrolled at the time in a masters program in information systems in a metropolitan university, and who received credit (completing an assignment) for participating in the experiment. To eliminate variability because of differences between subjects from the experimental error (Ramarapu et al. 1997), a repeated measures design was chosen (Kerlinger 1986). Each subject completed four cases that were assigned by randomizing the ⟨approach-domain-size⟩ combination, which provided direct control for order. The random assignment of subjects to order and treatment also ensured that expected differences, including differences in individual capabilities, between grouping (order and treatment levels) were zero at the time of randomization¹¹ (Cohen and Cohen 1983). Instructions were given to the subjects as part of the interface of the prototype (APSARA-Augmented 2000, APSARA-Naïve 2000). Both versions of the prototype had an identical user interface, except for the additional functionality in the augmented approach. Individual sessions, lasting less than two hours each, were staggered to ensure

¹¹ Experience was measured using a questionnaire (Nielsen 1993), along the dimensions: (1) Exposure to the analysis patterns, (2) familiarity with each of the two application domains, and (3) experience with design aids. Post-hoc tests showed that none of these were significantly different between the groups.

Table 7 Summary of Design Quality

Domain	Task Size	Naïve Approach (A1)		Augmented Approach (A2)		<i>t</i>
		E: Mean (<i>n</i>)	E: SD	E: Mean (<i>n</i>)	E: SD	
D1: Human Resource Management	Small (S1)	0.8148 (13)	0	0.4444 (4)	0.2772	2.673
	Medium (S2)	0.9630 (16)	0	0.7469 (9)	0.1126	5.755***
	Moderate (S3)	1.3718 (15)	0	0.5531 (7)	0.2307	9.389***
	Average	1.0586 (44)	N.A.	0.6186 (20)	0.2237	8.796***
						(<i>n</i> = 64)
D2: Warehouse Management	Small (S1)	0.1429 (13)	0	0.1810 (10)	0.1647	0.731
	Medium (S2)	0.8788 (9)	0	0.3939 (14)	0.2736	6.630***
	Moderate (S3)	0.7059 (9)	0	0.4608 (8)	0.1258	5.512**
	Average	0.5200 (31)	N.A.	0.3441 (32)	0.2368	4.203***
						(<i>n</i> = 63)

Note. **p* < 0.05; ***p* < 0.01; ****p* < 0.001.

that peer pressure played a minimal role. One of the researchers was present to answer any questions and the prototype recorded the results.

5.2. Results

There were 35 subjects, completing 4 tasks to generate a total of 140 design instances, of which 13 were not properly completed,¹² yielding 127 usable instances. These were evaluated following the assessment scheme described above. Table 7 shows the means, number of observations, and standard deviations for the dependent variable—design errors (E)—in each cell, marked by a combination of approach (A), task size (S), and domain (D) combination. A lower score indicates a better design.

The behavior of the dependent variable reflects the expected constant results for the naïve approach, and the range of results for the augmented approach (see Figure 7). An ANOVA was conducted to investigate the effects of design factors on the total error rate. Table 8 shows the main effects, and the interactions among factors.

Test of Hypothesis 1: Improvement. We expected that designs produced by the augmented approach (A2) would be better than those produced by the naïve approach (A1). Overall, the augmented

approach did produce designs with a lower average error (0.4497) than the naïve approach (0.8359). Hypothesis H1 was, thus, supported (*t* = 6.263, significant at *p* < 0.001).

Test of Hypothesis 2: Improvement Across Tasks of Different Size. We also expected that designs produced by the augmented approach would continue to be better than those produced by the naïve approach for each size. The augmented approach (A2) did, in fact, produce better designs than the naïve approach (A1) at each size (Table 9). Hypothesis H2 was, thus, supported at each level (*t* = 2.456, significant at

Table 8 ANOVA—Effects Design Factors on Total Error Rates

	DF	<i>F</i> -ratio
Main effects		
Approach (A)	1	195.670***
Domain (D)	1	202.045***
Size (S)	2	87.212***
Two-way interactions		
Approach * Domain (A * D)	1	22.638***
Approach * Size (A * S)	2	16.390***
Domain * Size (D * S)	2	8.972***
Three-way interaction		
Approach * Domain * Size (A * D * S)	2	29.512***
Error	115	
Model	126	90.863***

Note. **p* < 0.05; ***p* < 0.01; ****p* < 0.001.

¹²The errors included ignoring instructions (e.g., about the sequence of steps) or incomplete answers (e.g., not executing steps before recording the answer).

Table 9 Total Error Rates for Different Sizes Under the Two Approaches

Approach	Size		
	Small (S1)	Medium (S2)	Moderate (S3)
Naïve	0.4788	0.9327	1.1221
Augmented	0.2562	0.5321	0.5039
	$t = 2.456^*$	$t = 6.732^{***}$	$t = 7.542^{***}$

Note. $*p < 0.05$; $**p < 0.01$; $***p < 0.001$.

$p < 0.05$ in S1; $t = 6.732$, significant at $p < 0.001$ in S2; $t = 7.542$, significant at $p < 0.001$ in S3).

The augmented approach, therefore, exhibited a propensity for producing designs with fewer errors regardless of size. The naïve approach produced designs with increasing errors (see Figure 8) as the size increased. To further understand the improvements, the *absolute* and *relative* improvement for each size was computed using *pairwise* differences (see Table 10).

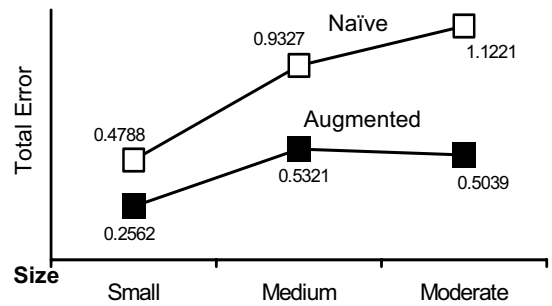
The post-hoc test ($F = 8.647$, significant at $p < 0.01$) revealed that improvement attributable to the augmented approach did not suffer with an increase in task size. It confirmed that the improvement was significantly different between S1 and (S2 and S3), but not significantly different between S2 and S3, further supporting Hypothesis H2 in the expected direction.

Test of Hypothesis 3: Improvement Across Domains. We expected that the designs produced for cases in the trained domain, Human Resource Management (D1), would be at least as good as those for the Warehouse Management domain (D2) that did not receive direct training.¹³ The results indicate that the augmented approach fared well for both domains. For the trained domain, the augmented approach produced designs with a lower average error (0.6186) than the naïve approach (1.0586). For the related domain, the augmented approach also produced designs with a lower average error (0.3441) than the naïve approach (0.5200). Hypothesis H3 was, therefore, supported ($t = 8.796$, significant at $p < 0.001$ in D1; $t = 4.203$, significant at $p < 0.001$ in D2) (see Table 11).

Prima facie, these findings appeared counterintuitive, i.e., the trained domain had higher error rates

¹³ This domain was *related* to a domain that did receive training (Inventory Control).

Figure 8 Design Errors for Different Size Levels



compared to the untrained, but related, domain. To investigate this, the *absolute* and *relative* improvements, attributable to the augmented approach, were computed using *pairwise* differences (see Table 12), similar to the exercise conducted for comparison across size levels.

This comparison revealed that the absolute improvements achieved across the two domains were, in fact, significantly different. The improvement (0.4579) because of the augmented approach in the trained domain (D1), was higher than that (0.2615) for

Table 10 Comparing Improvements Across Size Levels

Measure	Size		
	Small (S1)	Medium (S2)	Moderate (S3)
Absolute improvement due to augmented approach	0.0786	0.3797	0.5128
	Significant	Not significant	
Post-hoc test ($p < 0.05$)	$F = 8.647^{**}$		
Relative improvement due to augmented approach	10.61%	42.36%	46.37%

Note. $*p < 0.05$; $**p < 0.01$; $***p < 0.001$.

Table 11 Total Error Rates for Different Domains Under the Two Approaches

Approach	Average	Domain	
		Trained: Human Resource Management (D1)	Related: Warehouse Management (D2)
Naïve	0.8359	1.0586	0.5200
Augmented	0.4497	0.6186	0.3441
	$t = 6.263^{***}$	$t = 8.796^{***}$	$t = 4.203^{***}$

Note. $*p < 0.05$; $**p < 0.01$; $***p < 0.001$.

Table 12 Comparing Improvements Across Domains

Measure	Domain	
	Trained: Human Resource Management (D1)	Related: Warehouse Management (D2)
Absolute improvement due to augmented approach	0.4579	0.2615
$t = 2.168^*$		
Relative improvement due to augmented approach	40.08%	31.78%

Note. * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$.

the untrained domain (D2) ($t = 2.168$, significant at $p < 0.05$). The relative improvement was also higher in the trained domain (40.08%). Thus, the learning mechanisms improved the quality of designs produced, not only in a domain in which they were directly trained, but also in related domains. The tests, therefore, further supported Hypothesis H3 in the expected direction.

Finally, the summary results were plotted to obtain an intuitive understanding of the results. Figure 9 shows these results, which correspond to the summary results in Table 7. The figure shows two interesting anomalies. The first occurs for D1, A2 (augmented), as the error rate jumps from S1 to S2, and then falls for S3. The second occurs for D2, A1 (naïve), as the error rate jumps from S1 to S2 and then falls for S3. Together, these two anomalies contribute to narrowing the improvement for (D1, S2) and widening the improvement for (D2, S2). Additional tests were conducted and process logs for cases in these cells further analyzed to locate possible reasons for these anomalies. The anomaly for D1 (Figure 9A) could not be

Table 13 Summary of Findings

Hypothesis	Expected Direction	Finding
H1 Overall improvement	$E(A2) \leq E(A1)$	Supported
H2 Improvement across task sizes	$E(A2, S_n) \leq E(A1, S_n) \mid n = 1, 2, 3$	Supported
H3 Improvement across domains	$E(A2, D_k) \leq E(A1, D_k) \mid k = 1, 2$	Supported

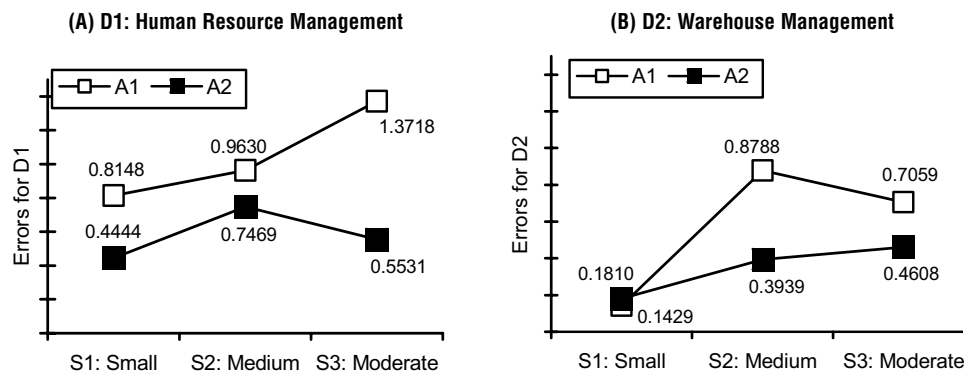
traced to any particular learning mechanisms. A t -test comparing S2 and S3 indicated that the difference was not significant ($t = 0.2041$, not significant at $p < 0.05$). A replication may be necessary to understand this further. The anomaly for D2 (Figure 9B) could be traced to the difference in the task descriptions between S2 and S3. The task statement for S2 was not a proper subset of that for S3. The differences in the two manifested in a higher proportion of missing objects for S2 (−16 on average) than for S3 (−11 on average) (e.g., the word *dock* was required for S2 but not for S3), resulting in a higher average error rate for S2 than that for S3.

In spite of these minor anomalies, the overall results strongly supported Hypotheses H1, H2, and H3 in the expected direction, indicating that the proposed learning mechanisms significantly reduced errors caused by naïve automation of analysis patterns reuse, regardless of task size or domain. Table 13 summarizes the results.

6. Discussion

The results demonstrate the potential of the learning mechanisms to benefit practice for reuse-based design. An approach to reuse of analysis patterns,

Figure 9



augmented with learning mechanisms, generates conceptual designs that are significantly better than a naïve approach. The augmented approach can be realized as a semi-automated intelligent assistant to create a preliminary conceptual design, which a developer can refine. Results from empirical testing indicate that the approach has considerable potential for sustained usefulness in practice, across multiple domains and across multiple sizes. The learning mechanisms achieve these two objectives by mitigating the two key challenges of simulating analogy making and designing by assembly.

Because of the complexity of the conceptual design task, it is exceedingly difficult to isolate the specific contribution of each learning mechanism. The results demonstrate a product-centered perspective, whereas isolating the specific contribution of the learning mechanisms would require a process-oriented perspective. Therefore, in addition to the results reported in the previous section, process logs of a small number of cases were traced to understand how the learning mechanisms contributed to the eventual success. Specifically, they were analyzed to assess how well they addressed the challenges of simulating analogy making and designing by assembly.

6.1. Simulating Analogy Making

The first 11 learning mechanisms (Table 2) address the challenge of preparing for and simulating analogy making. Eight mechanisms augment the generic reuse phase *retrieval*, and three the generic reuse phase *adaptation*. They facilitate the tasks of requirements parsing, object and pattern retrieval, and object and pattern instantiation. The learning mechanisms augment these tasks across different domains (see Tables 11 and 12), suggesting that a fundamental aspect of analogy making—moving from a higher level of abstraction in the analysis patterns to their application in specific domains—was well supported by the learning mechanisms. The results were also encouraging because the learning mechanisms supported such analogy making in a domain that was not directly trained. Tests for Hypothesis 2 showed that, for the different domains in the experiment, learning did, indeed, improve designs, and that the learning was transferable to related domains. The two domains

selected for the empirical analysis were sufficiently different to suggest that such analogy making is likely to succeed in other domains.

The process perspective revealed some interesting patterns. First, the presence of words (in the requirements statements) could have supermeanings or submeanings. Using these words profitably would require knowledge other than what the usage history could provide. For example, the word *chair* may have the meanings *furniture* or *head of a department*. Addressing this problem may require integrating domain ontologies (Gruninger and Fox 1995) along with the learning mechanisms. The second problem dealt with words in requirements statements that were synonymous, for which no objects were retrieved. For example, the words *portfolio* and *pool* from the requirements for a financial application may signify the same concept. Without the benefit of an object that maps to both (which would trigger the Competing words learning mechanism), a domain ontology could be used to understand that these two words signify the same concept.

6.2. Designing by Assembly

The last four learning mechanisms (Table 2) address the challenge of design by assembly, and are applicable to the generic reuse phase *integration*. They facilitate pattern synthesis, and iteration through the design process. The learning mechanisms augmented these tasks across different sizes (see Tables 9 and 10), suggesting that the support for design by assembly remained, regardless of size increases. The task sizes selected were appropriate for a limited empirical analysis that focused on the potential of the approach. For a problem of larger size, a human developer would need to divide the functionality of the system into multiple subsystems (Genero et al. 2002). The results would, then, be appropriate indicators for the application of the augmented reuse-based approach for each such component or subsystem.

The process perspective, in addition to the results above, revealed possible avenues for further improvement. First, words that were not mapped to any objects, but were shielded during the design process (by the learning mechanism Shield) were typically not connected to any existing objects. The problem can be

difficult to correct in an automated manner, although use of domain ontologies might suggest alternatives to the developer. A second, related, possibility was the existence of multiple connections across some objects. This posed a problem in a small number of cases, with the developer intervening to add unnecessary connections. Preventing such superfluous relationships is a difficult problem, although some simple metrics (e.g., the number of connections across every pair of objects) may be included in an implementation to warn the developer against the addition of superfluous relationships.

Overall, the results demonstrate that the learning mechanisms provide considerable automated support for both simulating analogy making and designing by assembly. The results are significant for several reasons. First, they demonstrate the effectiveness of learning mechanisms in improving reuse-based conceptual design. From a theoretical perspective this is the first demonstration of a combination of domain-independent artifacts (analysis patterns), and domain-dependent learning mechanisms for improving conceptual design. Although the analysis patterns suggested by Coad et al. (1995) were used in this research, the results are applicable to any patterns (e.g., those by Fowler 1997) that contain the elements in Figure 1. Second, the research demonstrates the feasibility of automating the reuse of analysis patterns in complex, conceptual design problems. The learning mechanisms provide a heuristic approach to addressing the set cover problem of applying analysis patterns to cover the requirements space. Third, the research extends prior work on automating conceptual design (Bubenko and Wangler 1992) through reuse by developing an approach to incorporate learning mechanisms into a naïve approach to conceptual design.

There is also an important methodological outcome from this research process. Our research approach involves testing claims from a technological innovation using a laboratory experiment, and interpreting the results in terms of theoretical challenges addressed by the technological innovation. Our procedure has focused on the incremental contribution of an innovation (which we termed Augmented) over a previous solution approach (which we termed Naïve).

The procedure should be useful for other researchers who may engage in similar endeavors.

The research has some limitations, such as the use of students as subjects. However, the subjects were recruited from IT professionals, who were enrolled at the time of the study in a specialized, graduate-level program in information systems in a metropolitan university. Therefore, they represent, at least to some extent, a sample from a self-selected set of individuals who have elected to become information systems professionals (Purao et al. 2001). Similar use of novice subjects has been reported elsewhere (Irwin 2002, Agarwal et al. 1996). Another possible limitation is the use of a percentage score for calculating quality. The simple score does, however, provide a reasonable indication of design quality, and is appropriate for capturing errors introduced by multiple potential sources, attributable to a sequence of design decisions. Finally, the experiment contains a limited test of scalability or assessment across domains. Introduction of the innovation into an organization, possible deployment on the Internet, and data collection from several organizations are beyond the scope of this paper. The intent of the results reported in this paper is to demonstrate the potential of the proposed approach for sustained usefulness. Issues of confidence in the training set and setting of threshold values are important. These, however, require further work such as optimization of threshold values and field studies. The results we have presented suggest an initial *proof of concept*, followed by a rigorous test of the *potential* of the approach. Generalizability of the results such as thresholds used, therefore, is necessarily limited to one of the cases suggested by Baskerville (1996). These limitations present opportunities for future research.

7. Conclusion

This research proposed and empirically tested an approach to semi-automating conceptual design with analysis patterns reuse, augmented with learning mechanisms. The learning mechanisms use supervised learning to facilitate reuse of domain-independent artifacts such as analysis patterns. The empirical analysis suggests that the results are useful across domains and scalable within the confines

of the laboratory experiment. This research represents a combination of reuse situations, namely, functional and conceptual reuse (Zhang and Lyytinen 2001), because it demonstrates how analysis patterns, as garnered knowledge, can be reused during conceptual design across different domains. Although it is beyond the scope of this research, our study also provides the potential for instantiation reuse. That is, as a result of the learning processes, it is possible to find new variations in a family of designs and, in turn, to generate new analysis patterns.

The results of this research have several implications for practice. First, conceptual modeling is an important, but difficult, part of systems development, so, systematic approaches to improving conceptual design are very beneficial. We have shown that the conceptual design process can be semi-automated and enhanced with learning to improve the quality of the design obtained, while it requires less work on the part of the developer. The learning mechanisms can be implemented as an intelligent assistant, and applied within several organizations to learn about the use of analysis patterns in several domains and across industries. Novice developers using the approach should be able to produce designs that are close to the quality of those produced by experienced developers.

Experienced developers might produce designs that are as good or better than those obtained by our approach. However, they will also benefit from an initial design that they can refine. The research would, therefore, benefit organizations because they would not need as many experienced developers, and the burden on the experienced developers would be reduced. Other plausible uses of the approach may be as a training aid for developers to become familiar with the use of analysis patterns in different situations. Wohed (2000b) suggests a similar application. Incorporating this in the current research would require an explanation engine to communicate the rationale for the suggestions made by the intelligent agent. Another possibility is to use the methodology to generate multiple, alternative designs quickly, allowing more creativity during design and reducing design fixation. To assess the improvement in speed and cost of the design, the differences in time

(between different versions of support and the manual expert driven approach) may be tracked. The use and usefulness of the methodology in these different avenues remain topics for future research.

Further research is also needed to improve the methodology itself. First, an alternative approach to evaluating the contribution of individual learning mechanisms may include using techniques such as process tracing (Todd and Benbasat 1987). Second, the rules embedded in the structure of a pattern may themselves be modified during reuse. This will require a more active role from the designer instead of the current role of a critic who accepts or rejects suggestions. Such pattern modifications could augment the patterns base itself, and indirectly further augment the reuse process. Third, knowledge gained through the augmented approach suggests that it may be possible to generate larger-grain, domain-specific, reusable artifacts, improving the design products further (Han et al. 1999). Fourth, the learning mechanisms may be adapted for application in other environments such as system configuration and parameterization, deriving design alternatives from use cases, and mapping designs to goals or constraints. We are currently pursuing some of these extensions.

Acknowledgments

The authors wish to thank Sal March, the Senior Editor, the AE, and the reviewers for their work on previous versions of this manuscript. The authors also thank Bala Ramesh, Arun Rai, Ashley Bush Mark Keil, Detmar Straub, and Matti Rossi for their comments on earlier drafts, and Maung Sein for his counsel during data analysis. This research was supported by the J. Mack Robinson College of Business, Georgia State University.

Appendix A. Illustration of Learning Mechanisms

The illustration serves as a description of learning mechanisms following an example from the human resource domain.¹⁴

We need a system to manage our employees and staff members, their qualifications, leave records, benefits, and track retirement funds. The system will be responsible for storing résumé and personal information on employees, and will create, record, and classify employee skills to facilitate allocation of skills to departments as required.

¹⁴ Formal representations of learning mechanisms along with dependencies, and meta-rules for their invocation, can be found in Purao et al. (2001).

The learning mechanisms are grouped following the three generic reuse phases and the tasks within each phase outlined in Table 2.

PHASE: RETRIEVAL

Task: Requirements Parsing.

Purpose. Augmenting parsing of requirements stated as natural language assertions.

- The Discard mechanism uses the history of words discarded by designers during previous sessions to provide intelligent suggestions for better parsing of requirements. For example, if the word *qualifications* was not used much in the prior cases, it is suggested as a candidate for discarding.
- The Shield mechanism identifies words from previous sessions that were important but could not be exploited using the then existing knowledge in the usage history. These words represent important concepts that the developer could not map directly to any objects. For example, *retirement* is an important concept in the human resource domain, but not mapped to a specific object. The shield mechanism allows such important words to remain without requiring an explicit mapping to an object.
- The Guide mechanism suggests important words to the designer, based on prior cases in the industry or application domain that do not appear in the natural language assertions. For example, *contract* may be important for the human resource domain, but was not specified in the requirement statements.

Task: Object Retrieval.

Purpose. Augmenting retrieval of objects based on parsed requirements statements.

- The Competing Objects mechanism applies to cases where one word identifies multiple generic objects. It uses information from prior cases to more accurately identify objects based on a significant word. For example, *employee* may map to *Participant* and *Member*, one of which may be suggested over the other.
- The Competing Words mechanism applies to the reverse cases, where more than one word from the natural language assertion suggests the same generic object. It uses information from prior cases to suggest one of the available alternatives as the preferred alternative. For example, *employee* as well as *staff* may map to the object *Participant*.

Task: Pattern Retrieval.

Purpose. Augmenting retrieval of patterns based on objects or parsed requirements statements.

- The Direct Instantiation mechanism exploits a complete pattern and its example. It suggests an alternative that performs retrieval and instantiation simultaneously. For instance, with *résumé* in the assertion, this mechanism finds *resume_bank-resume*, an instantiation of the pattern *Container-Content*, and with *employee*, finds *employee-performance*, an instantiation of the pattern *Participant-Transaction*. The learning mechanism then suggests these for inclusion in the current design.
- The Pattern Affinity mechanism uses information about patterns that are often used together in prior designs. Consider *Place-Transaction*, *Container-Content*, and *Container-ContainerLineItem*

retrieved for the current design based on identification of *place* and *container*. If prior designs are found where all these patterns co-exist, any other patterns that also co-exist in these designs (e.g., *Actor-Participant*) are suggested as candidate patterns for retrieval in the current design.

- The Pattern Indifference mechanism is the inverse of the Pattern Affinity mechanism. It uses information about patterns that may not have co-existed in prior designs, for example, patterns *Place-Transaction* and *Plan-PlanExecution* derived from *management*. If they are retrieved for the current design, the designer is informed that this is an anomalous situation.

PHASE: ADAPTATION

Task: Object and Pattern Instantiation.

Purpose. Augmenting instantiation of retrieved objects and patterns based on prior cases.

- The Partial Instantiation Affinity mechanism utilizes information about prior instantiations of a pattern. Because patterns are retrieved based on identification of an object, it is likely that, initially, they will be only partially instantiated. For example, after *Participant-Transaction* is retrieved and instantiated based on the keyword *employee* (i.e., object *Participant*), *Transaction* remains uninstantiated. The pattern may have been instantiated in prior designs as *employee-assignment* and *employee-contract*. Then, possible instantiations of *Transaction* in the current design may be suggested as *assignment* or *contract*.
- The Inverse mechanism is a variation of the Partial Instantiation Affinity mechanism. It uses inverse relationships among prestored and learned keywords. For example, *Transaction-SubsequentTransaction* might have been partially instantiated for the current design as *Sale-SubsequentTransaction*. If any prior pattern instantiations are found that correspond to the word *sale*, or its inverse *purchase*, suggestions made to the developer will include the corresponding instantiations, and their inverses. For example, if *purchase-payment* is found as a prior instantiation, then *payment* and its inverse *receipt* are suggested to instantiate the object *SubsequentTransaction*.
- The Instantiation Affinity mechanism reflects the fact that some patterns have related instantiations. For example, *department-employee* and *resume-resume_item* represent related instantiations of the patterns *Group-Member* and *Container-Content*. They provide a plausible instantiation of one pattern if the other pattern has been used with the related instantiation. Based on prior designs that share the current instantiation of *Group-Member*, various instantiations of *Container-Content* are suggested, such as *benefit-benefit_items*, *resume_bank-resume*, or *skill-skill_item*.

PHASE: INTEGRATION

Task: Synthesis of Instantiated Patterns.

Purpose. Augmenting synthesis of instantiated patterns based on prior cases.

- The Synthesis by Object mechanism uses information about prior superimpositions of objects, such as *Place* and *Container*. Then,

if the current design contains these objects, a synthesis of patterns containing these two objects can be suggested for the current design. For example, *department* can be both *Place* and *Container*. It is then suggested to the developer that patterns containing these two objects be synthesized.

- The Synthesis by Keyword mechanism is based on the fact that some objects share a number of instantiations in prior designs. The generic objects *Item* and *Content* may share a number of prior instantiations such as *product*, *item*, *defect*, and *line* from different domains. If these generic objects have been identified for the current design, they can be suggested as candidates for synthesis. *Item* and *Content* can be synthesized for shared items (lines, or products) common among various instantiations such as *benefit*, *skill*, *résumé*, etc.

Task: Graph Spreading.

Purpose. Allows iteration after an initial pass is made through all of the design steps.

- Spreading by Keyword utilizes information about instantiations shared by different objects to retrieve new objects and patterns. For example, *Container-Content* may have been retrieved for the current design based on the keyword *department* (object *Container*); the object *Content* may have been instantiated as *resume_bank* via another learning mechanism (such as Partial Instantiation Affinity). The new keyword *resume_bank* may then be used to identify new objects, such as *Associate* (following the knowledge available in the patterns-base and usage history), and consequently retrieve additional patterns such as *Associate-OtherAssociate*.
- Spreading via Recursion exploits the fact that aggregate patterns (aggregate-part) such as *Assembly-Part* or *Container-Content* can be applied recursively, when the part object can be further decomposed. Recursion and prior instantiations then suggest consequent instantiations. For example, *resume_bank* can play the role of *Container* for *résumé*, the *Content* of *resume_bank*. Based on prior instantiations, the *résumé* and *skill* may be suggested as further instantiation for the recursive application of the *Container-Content* pattern.

Appendix B. Assessment Scheme

An assessment scheme was developed, using measures similar to those suggested by Moody et al. (1998) and Storey (1993), to compare the model generated with the naïve or augmented approach against the ideal model. It captures two types of errors. The first are Type I errors, i.e., errors of omission, such as missing objects, missing instantiations of objects, and missing relationships among objects, not counting those errors caused by missing objects. The second are Type II errors, i.e., errors of commission, such as the incorrect identification of objects, and incorrect relationships, other than those errors caused by incorrect objects. Errors of both types are classified as major, minor, or reverse. Major errors seriously affect the quality of a model. These are penalized at three points per error. Minor errors are less severe, and are penalized at one point per error. Finally, reverse errors represent plausible extensions that a human developer may consider, and are rewarded. These include objects and relationships that may be appropriate given

the requirements but were not anticipated by the researchers when creating the ideal model. Because the decision between incorrect and plausible can be subjective, the reward is conservatively set at one point per reverse error. The table below shows the assessment scheme. The numbers in each cell represent the points assigned to an error type.

Error Type	Modeling Element		
	Object Instantiation		Association
Type I: Errors of omission			
Missing elements	-3	-1	-3 (-1, if due to missing objects)
Type II: Errors of commission			
Incorrect elements	-3	-1	-3 (-1, if due to incorrect objects)
Redundant elements	-1	-1	-1 (-1, if due to redundant objects)
Plausible elements	+1	N.A.	+1

References

Agarwal, R., A. P. Sinha, M. R. Tanniru. 1996. Cognitive fit in requirements modeling: A study of object and process methodologies. *J. Management Inform. Systems* 13(2) 137-162.

Alexander, C. 1964. *Notes on the Synthesis of Form*. Harvard University Press, Cambridge, MA.

—. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, Oxford, U.K.

Appleton, B. 1997. Patterns and software: Essential concepts and terminology. Retrieved December 2002, from <http://www.enteract.com/~bradapp/docs/patterns-intro.html>.

APSARA-Augmented. 2000. Front-End (disabled). Retrieved December 2002, from <http://www.unlv.edu/faculty/than/ApsaraA/APSARA-ML.html>.

APSARA-Naïve. 2000. Front-End (disabled). Retrieved December 2002, from <http://www.unlv.edu/faculty/than/ApsaraN/APSARA-Naive.html>.

Bansiya, J. 1998. Automating design-pattern identification. *Dr. Dobb's J.* Retrieved December 2002, from <http://www.ddj.com/documents/s=919/ddj9806a/9806a.htm>.

Barto, A., S. Bradtke, S. Singh. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1-2) 81-138.

Basel. 1999. Principles for the management of credit risk. *Basel Committee Publications No. 54* (July). Retrieved December 2002, from <http://www.bis.org/publ/bcbs54.htm>.

Baskerville, R. 1996. Deferring generalizability: Four classes of generalization in social enquiry. *Scandinavian J. Inform. Systems* 8(2) 5-28.

Batra, D. 1993. A framework for studying human error behavior in conceptual database modeling. *Inform. Management* 25 121-131.

- , J. Davis. 1992. Conceptual data modeling in database design: Similarities and differences between expert and novice designers. *Internat. J. Man-Machine Stud.* 37 83–101.
- Biggerstaff, T. J., A. J. Perlis. 1989. Introduction. T. J. Biggerstaff, A. J. Perlis, eds. *Software Reusability*, Vol. I. ACM Press, NY, xv–xxv.
- Briscoe, G., T. Caelli. 1996. *A Compendium of Machine Learning: Symbolic Machine Learning*. Ablex Pub. Corp., Norwood, NJ.
- Bubenko Jr., J. A., B. Wangler. 1992. Research directions in conceptual specification development. Loucopoulos, Zicari, eds. *Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development*. Wiley & Sons, CA.
- Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. 1996. *Pattern-Oriented System Architecture, Volume 1: A System of Patterns*. Wiley Publications, Hoboken, NJ, 325–343.
- Casakin, H., G. Goldschmidt. 1999. Expertise and the use of visual analogy: Implications for design educations. *Design Stud.* 20 153–175.
- Coad, P., D. North, M. Mayfield. 1995. *Object Models: Strategies, Patterns, & Applications*. Prentice Hall, Upper Saddle River, NJ. Also available at *Strategies and Patterns Handbook: Hypertext Edition*, Version 2.0a. Object International, Inc. Retrieved December 2002, from <http://www.cis.ksu.edu/~hankley/d644/CoadPatterns/>.
- Cohen, J., P. Cohen. 1983. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, Mahwah, NJ.
- Cole, W. G. 1987. Metaphor graphics and visual analogy for medical data. *Sympos. Comput. Appl. Medical Care*, Washington, D.C., AMIA, Bethesda, MD.
- de Champeaux, D. 1997. *Object-Oriented Development Process and Metrics*. Prentice-Hall, Upper Saddle River, NJ.
- Dey, D., S. Sarkar. 2000. Modifications of uncertain data: A Bayesian framework for belief revision. *Inform. Systems Res.* 11(1) 1–16.
- Ericsson, K. A., J. J. Staszewski. 1989. Skilled memory and expertise: Mechanisms of exceptional performance. H. A. Simon, K. Kotovsky, D. Klahr, eds. *Complex Information Processing: The impact of Herbert A. Simon*. Lawrence Erlbaum, Hillsdale, NJ, 235–267.
- Florijn, G., M. Meijers, P. van Winsen. 1997. Tool support for object-oriented patterns. *Proc. European Conf. Object-Oriented Programming*, Jyväskylä, Finland.
- Fowler, M. 1997. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Reading, MA.
- Frakes, W. B., C. J. Fox. 1995. Sixteen questions about software reuse. *Comm. ACM.* 38(6) 75–87.
- , B. A. Nejme. 1990. An information system for software reuse. *Software Reuse: Emerging Technology*. IEEE CS Press, Washington, D.C., 142–151.
- Gabriel, R. 1996. *Patterns of Software: Tales From the Software Community*. Oxford Books, Oxford, U.K.
- Gamma, E., R. Helm, R. Johnson, J. Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.
- Genero, M., J. Olivas, M. Piattini, F. Romero. 2002. Assessing object-oriented conceptual models maintainability. *Internat. Workshop Conceptual Model. Quality*, Tampere, Finland (October).
- Gentner, D. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive Sci.* 7(2) 155–170.
- Gibbs, C. W. 1994. Software's chronic crisis. *Sci. Amer.* (Sept.) 86–95.
- Glass, R., I. Vessey. 1996. Contemporary application-domain taxonomies. *IEEE Software* (July) 63–76.
- Gruninger, M., M. S. Fox. 1995. Methodology for the design and evaluation of ontologies. *Workshop Basic Ontological Issues Knowledge Sharing*, Montreal, Canada, 19–20.
- Han, T., S. Purao, V. Storey. 1999. A methodology for building a repository of object-oriented design fragments. *Proc. 18th Internat. Conf. Conceptual Model*, Paris, France, 203–217.
- Harrison, E., B. Foote, H. Rohnert, eds. 2000. *Pattern Languages of Program Design 4*. Addison-Wesley, Reading, MA.
- Irwin, G. 2002. The role of similarity in the reuse of object-oriented analysis models. *J. Management Inform. Systems* 19(2) 219–248.
- Johannesson, P., P. Wohed. 1999. The deontic pattern—A framework for domain analysis in information systems design. *Data and Knowledge Engrg.* 31(2) 135–153.
- Kaelbling, L. P., M. L. Littman, A. W. Moore. 1996. Reinforcement learning: A survey. *J. Artificial Intelligence Res.* 4 237–285.
- Karlsson, E. 1995. *Software Reuse: A Holistic Approach*. John Wiley & Sons, Hoboken, NJ.
- Kerlinger, F. N. 1986. *Foundations of Behavioral Research*. Holt, Rinehart and Winston, Fort Worth, TX.
- Krueger, C. W. 1992. Software reuse. *ACM Comput. Survey* 24(2) 131–183.
- Lea, D. 1994. Christopher Alexander: An introduction for OO designers. *Software Engrg. Notes* 19(1) 39–46.
- Lorenz, M., J. Kidd. 1994. *Object-Oriented Software Metrics*. Prentice-Hall, Upper Saddle River, NJ.
- Lowry, M., R. McCartney, eds. 1991. *Automating Software Design*. AAAI Press/MIT Press, Cambridge, MA.
- Maiden A., C. Sutcliffe. 1993. Exploiting reusable specifications through analogy. *Comm. ACM* 35(4) 55–64.
- Mili, H., F. Mili, A. Mili. 1995. Reusing software: Issues and research directions. *IEEE Trans. Software Engrg.* 6(June) 528–562.
- Moody, D., G. G. Shanks, P. Darke. 1998. Improving the quality of entity relationship models—Experience in research and practice. *Proc. 17th Internat. Conf. Conceptual Model*, Singapore, 255–276.
- Muggleton, S., L. De Raedt. 1994. Inductive logic programming: Theory and methods. *J. Logic Programming* 19–20 629–679.
- Nielsen, J. 1993. *Usability Engineering*. Academic Press, San Diego, CA.
- Nilsson, N. 1998. *Artificial Intelligence: A New Synthesis*. Morgan Kaufman Publishers, San Francisco, CA.
- NIST. 2002. Set Cover Problem. *Dictionary of Algorithms and Data Structures*. Retrieved December 2002, from <http://www.nist.gov/dads/HTML/setcover.html>.

- Prieto-Diaz, R. 1993. Status report: Software reusability. *IEEE Software* **10**(3) 61–66.
- Prietula, M. J., H. A. Simon. 1989. The experts in your midst. *Harvard Bus. Rev.* **67**(1) 120–124.
- Purao, S. 1998. APSARA: A tool to automate system design via intelligent pattern retrieval and synthesis. *DataBase Adv. Inform. Systems* **29**(4) 45–57.
- , V. Storey. 1997. Intelligent support for retrieval and synthesis of patterns for object-oriented design. *Proc. 16th Internat. Conf. Conceptual Model*, Springer-Verlag, Los Angeles, CA.
- , A. Bush, M. Rossi. 2001. Problem and design spaces during object-oriented design: An exploratory study. *Proc. 34th Annual Hawaii Internat. Conf. System Sci.*, Maui, HI (January).
- Ramarapu, N. K., M. N. Frolick, R. B. Wilkes, J. C. Wetherbe. 1997. The emergence of hypertext and problem solving. *Decision Sci.* **28**(4) 825–850.
- Rouse, W. B., N. M. Morris. 1986. On looking into the black box: Prospects and limits in the search for mental models. *Psych. Bull.* **100**(3) 349–361.
- Setliff, D. E., E. Kant, T. Cain. 1993. Practical software synthesis—Introduction. *IEEE Software* **10**(3) 6–9.
- Simon, H. 1981. *The Science of the Artificial*. MIT Press, Cambridge, MA.
- Storey, V. C. 1993. A selective survey of the use of artificial intelligence methods for database design systems. *Data and Knowledge Engrg.* **11** 61–102.
- , C. Thompson, S. Ram. 1995. Understanding database design expertise. *Data and Knowledge Engrg.* **16**(Aug.) 97–124.
- Straub, D., P. Carlson, E. Jones. 1993. Deterring cheating by student programmers: A field experiment in computer security. *J. Management Inform. Systems* **5**(1) 33–48.
- Szyperski, C. 1998. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Reading, MA.
- Tasks. 2000. Task descriptions used for experiments. Retrieved December 2002, from http://cob.nevada.edu/Han_www/AugmentedReuse/task-descriptions.htm.
- Todd, P., I. Benbasat. 1987. Process tracing methods in decision support systems research: Exploring the black box. *Management Inform. Systems Quart.* **11**(4) 493–512.
- Wohed, P. 2000a. Conceptual patterns for reuse in information systems analysis. *Proc. 12th Internat. Conf. Adv. Inform. Systems*, Stockholm, Sweden, 157–175.
- . 2000b. Tool support for reuse of analysis patterns—A case study. A. Laender, S. Liddle, V. Storey, eds. *Proc. 19th Internat. Conf. Conceptual Model*, LNCS 1920, Springer, 196–209.
- Zhang, Z., K. Lyytinen. 2001. A framework for component reuse in a metamodeling-based software development. *Requirements Engrg. J.* **(6)** 116–131.

Sal March, Senior Editor. This paper was received on April 18, 2001, and was with the authors 12 months for 2 revisions.