ABSTRACT

AUTOMATING REUSE FOR SYSTEMS DESIGN

By

TAEDONG HAN

MAY, 2002

Committee Chair: Dr. Sandeep Purao and Dr. Veda Storey

Major Department: Computer Information Systems

Achieving successful reuse requires overcoming a number of personal, social, and technological obstacles. Viewed from a cost-benefit perspective, automatable solutions are necessary to lower the overhead cost spent in search, adaptation, and integration. To ensure high payoff, reuse must be actively pursued early in the development cycle, when the potential benefits are much higher.

This research proposes innovative approaches that can lower the technological and human barriers to realizing the promise of reuse. The goal is to develop, operationalize, automate, and test a set of related approaches that can facilitate and improve reuse at the early, conceptual design stage of the information systems development process.

This research is done in three phases. The first phase starts from improving an existing study, which suggests a framework for reuse-based design with analysis patterns, by developing an augmented approach incorporating learning mechanisms that exploit prior design experiences. Algorithms are developed in two categories; supervised and
inductive learning mechanisms. The approach is validated in a lab experiment. The results show that the augmented approach is superior in all tested aspects of improvement - scalability, transferability over different problem complexity levels, and across multiple domains. Phase 2 proposes novel reusable artifacts – called design fragment – that have built-in implementation mechanism, and a methodology for building a repository of these artifacts. Feasibility of this approach is assessed by simulating the process. And the last phase validates the artifacts and approaches proposed in phase 2. A research model is developed to explore the possibility of acceptance of design fragments, and a lab experiment is performed using this model. Hypotheses are partially supported. Design fragments are superior to analysis patterns in ease of retrieval and ease of assembly, but not in granularity and abstractness. Ease of retrieval positively influences on perceived ease of use. But granularity and abstractness does not show relationship to perceived usefulness.

The main contribution of this research is to provide practical solution approaches that can facilitate and improve the practice of reuse.
Table of Contents

CHAPTER 1  INTRODUCTION ................................................................. 1
  1.1 MOTIVATION .............................................................................. 1
  1.2 BACKGROUND .......................................................................... 2
  1.3 RESEARCH QUESTIONS AND OBJECTIVES ................................. 3
  1.4 SCOPE AND CONTRIBUTIONS .................................................. 5

CHAPTER 2  LITERATURE REVIEW ...................................................... 7
  2.1 OBJECT-ORIENTED DESIGN .................................................... 7
  2.2 REUSE ..................................................................................... 10
    2.2.1 Reusable Artifacts ............................................................ 10
      2.2.1.1 Comparison of Reusable Artifacts ................................. 15
      2.2.1.2 Approaches to Developing Reusable Artifacts ............... 17
    2.2.2 Designing with Reusable Artifacts ...................................... 18
      2.2.2.1 Retrieval .................................................................... 19
      2.2.2.2 Adaptation .................................................................. 20
      2.2.2.3 Integration .................................................................. 21
  2.3 REUSE AUTOMATION ............................................................... 22
    2.3.1 Known Problems .............................................................. 22
    2.3.2 Approaches to Reuse Automation ....................................... 24
    2.3.3 Tools for Reuse-based Design ............................................ 27
    2.3.4 Automated Design ............................................................ 28

CHAPTER 3  RESEARCH METHODOLOGY ............................................. 30
  3.1 ARCHITECTURE OF THE PROPOSED SOLUTION APPROACH ........ 30
  3.2 SOLUTION APPROACHES ........................................................ 34
  3.3 PROTOTYPE DEVELOPMENT ...................................................... 35
  3.4 VALIDATION ............................................................................. 37

CHAPTER 4  IMPROVING REUSE-BASED DESIGN: AUGMENTING ANALYSIS PATTERNS REUSE WITH LEARNING ................................................. 38
  4.1 INTRODUCTION ......................................................................... 38
    4.2.1 Conceptual Design with Analysis Pattern Reuse ................... 40
    4.2.2 The Role of Learning ........................................................ 42
  4.3 IMPROVING REUSE-BASED DESIGN WITH LEARNING ............... 44
    4.3.1 Reinforcement and Supervised Learning Mechanisms ........... 45
      4.3.1.1 Augmenting Parsing of Natural Language Assertions ........ 45
      4.3.1.2 Initiating the Design Process ......................................... 46
    4.3.2 Relevance-Based and Knowledge-Based Inductive Learning .... 48
      4.3.2.1 Affinity-based Mechanisms ......................................... 48
      4.3.2.2 Synthesis Mechanisms ................................................. 51
      4.3.2.3 Spreading Mechanisms ................................................. 52
  4.4 APPLICATION OF LEARNING MECHANISMS ................................. 54
    4.4.1 Implementation ............................................................... 55
    4.4.2 Training the Learning Mechanisms .................................... 55
List of Tables

Table 2.1 Comparison of reusable artifacts in terms of desirable properties .......... 17
Table 3.1 Research questions, Solution approaches, and Techniques ...................... 34
Table 3.2 Research Objectives, Hypotheses, and Validation Approaches .................. 37
Table 4.1 Elements of Analysis Patterns ............................................................... 40
Table 4.2 Learning Strategies ................................................................................. 44
Table 4.3 Learning Mechanisms for Augmenting Parsing ...................................... 46
Table 4.4 Learning Mechanisms for Augmenting the Initiating of Design ............... 48
Table 4.5 Affinity-based Learning Mechanisms ...................................................... 51
Table 4.6 Learning Mechanisms for Augmenting Synthesis .................................... 51
Table 4.7 Learning Mechanisms for Augmenting Spreading ................................... 53
Table 4.8 Applicability of Learning Mechanisms to the Naïve Approach and Meta-Rules ........................................................................................................ 54
Table 4.9 Thresholds for Learning Mechanisms ...................................................... 56
Table 4.10 Assessment Scheme .............................................................................. 59
Table 4.11 Hypotheses Posited ............................................................................... 61
Table 4.12 Comparative Sizes ([Lorenz and Kidd 1994, de Champeaux 1997]) ......... 62
Table 4.13 Design Error Means for the two Domains ............................................. 65
Table 4.14 ANOVA - Effects of Design Factors on Total Error Rates ..................... 65
Table 4.15 Total Error Rates for Different Sizes under the Two Approaches .......... 66
Table 4.16 Total Error Rates for Different Domains under the Two Approaches ....... 67
Table 4.17 Comparing Improvements across Size Levels ...................................... 68
Table 4.18 Comparing Relative Improvements across Size Levels ....................... 69
Table 4.19 Comparing Improvements under Augmented Approach across Domains .... 70
Table 4.20 Comparing Relative Improvements across Domains ............................ 70
Table 4.21 Summary of Findings .......................................................................... 71
Table 4.22 Improvements under Augmented Approach for Domains at Size Levels .... 71
Table 5.1 Evaluation of Existing Reusable Artifacts ............................................. 79
Table 5.2 Reusing design fragments ...................................................................... 84
Table 5.3 Significant words in requirement statements ......................................... 88
Table 5.4 Keywords in requirement statements .................................................... 88
Table 5.5 Keyword Cohesion .................................................................................. 89
Table 5.6 Keyword clusters ................................................................................... 90
Table 5.7 Instantiated analysis patterns in each design ......................................... 94
Table 5.8 Design fragments ................................................................................... 96
Table 5.9 Results of the Design Fragment Retrieval Process ................................ 97
Table 5.10 Pool of keywords in warehouse management domain ........................... 99
Table 5.11 Thresholds for keyword clustering ...................................................... 100
Table 5.12 Number of keyword clusters generated .............................................. 101
Table 5.13 Design distribution at keyword coverage threshold of 50% and 60% ....... 102
Table 5.14 Parameter and threshold in design clustering simulation ....................... 103
Table 5.15 Number of design fragments generated .............................................. 104
Table 5.16 Parameter and threshold in reuse simulation ...................................... 105
Table 5.17 Retrieving design fragments (threshold = 50%) .................................. 105
Table 5.18 Retrieving design fragments (threshold = 60%) .................................. 105
CHAPTER 1
INTRODUCTION

1.1 Motivation

The software industry is under considerable pressure to meet increased demands (Lapointe, 1999). Although productivity has steadily increased over the past 30 years, the gap between demands placed on the software industry and what the state of practice can deliver continues to widen (Gibbs, 1994; Pearson, 1999). Several decades of research have confirmed that reuse is the only realistic approach to bridging this gap (Krueger, 1993; Mili et al., 1995). Achieving successful reuse, however, requires overcoming a number of personal, social and technological obstacles (Jacobson, 1996).

Viewed from a cost-benefit perspective – the costs of reuse must be lowered and the potential benefits improved. Every instance of reuse has an overhead cost consisting of time and effort spent in search, adaptation, and integration. Automatable solutions to lower these costs are, therefore, necessary to facilitate reuse (Mili et al., 1995). Similarly, to ensure high payoff, reuse must be actively pursued early in the development cycle, when the potential benefits are much higher (Prieto-Diaz, 1993) – instead of attempting it at the implementation stage by focusing on code reuse (“State of the Art,” 1998). In mathematical terms, creating a design from existing artifacts (design by composition) is similar to the set cover problem and hence, challenging (Mili et al., 1995; Brodie, 1998). Effective reuse, therefore, requires, among other things, tools and techniques that can assist the designer (Nierstrasz & Meijler, 1995). Without such modes of assistance, realistic reuse-based design approaches have remained an elusive goal.
Accordingly, this research is focused on developing innovative approaches that can lower the technological and human barriers to realizing the promise of reuse. The goal of this research is to develop, operationalize, automate, and test a set of related approaches that can facilitate and improve reuse at the early, conceptual design stage of the information systems development process.

1.2 Background

Reuse involves two processes: identifying reusable artifacts and building their repository, for reuse, and building systems with reusable artifacts, with reuse (Mili et al., 1995).

For reuse is the building phase. It focuses on how the reusable artifacts are built(indexed/organized). The range of reusable artifacts to be identified is wide. It not only includes software routines or individual objects, but also extends to any reusable by-products during software systems development such as requirement analyses, designs, domain architecture, documentation, etc. Recently, software patterns (Appleton, 1997) have emerged as a viable alternative to facilitate reuse. They have been identified at the conceptual design stage (analysis patterns (Coad et al., 1995; Fowler, 1997)) as well as the detailed design stage (design patterns (Gamma et al., 1995; Pree, 1994)). Both represent generic solutions that can be applied, by analogy, in different domains (Gamma et al., 1995). In spite of this variety, the task creating reusable artifacts remains effort-intensive and few reusable artifacts provide built-in mechanism for designing with reuse.

With reuse is the implementation phase of reuse, and depends on which type of reusable artifacts is used. A few, naïve, approaches are available to support new system
design with reuse of analysis patterns. These, naïve, reuse-based design approaches, utilize simple NLP techniques (e.g., Purao and Storey (1997a; 1997b), Wohed (2000)) without taking into account contextual factors such as differences in industry and application domains (Glass & Vessey, 1996) or designer preferences. Nor do they emulate the learning, which can occur via repeated application and designer interaction, such as different propensities that promote or hinder the use of specific patterns in different design problems. In spite of its simplicity and shortcomings, the basic framework for reuse-based design with analysis patterns suggested by Purao and Storey (1997a) provides the starting point for this research.

1.3 Research Questions and Objectives

The initial research question starts from the background above:

*Question 1. How can we improve the naïve approach to conceptual system design with reuse of analysis patterns?*

The first objective to address the question above, based on the deficits of current naïve approaches, is:

*Objective 1. To develop an augmented approach to design with reuse incorporating learning mechanisms that exploit prior experiences.*

The next step of the study is to leverage the knowledge in conceptual designs assembled above to develop artifacts for reuse. Thus the question for the study is:

*Question 2. How can we develop a new, more effective artifact for reuse, by leveraging the knowledge in conceptual designs assembled in the above (question 1) phase?*

This question deals with known problems of designing with reuse such as problem-solution space mismatch, and integration problems in artifact adaptation [Mili et al., 1995]. These problems can be solved to some degree by creating artifacts that require
less search, composition, and adaptation costs. Thus, the objective to address the second question is:

*Objective 2. To propose a novel reusable artifact with built-in implementation mechanism.*

And the last step of the study is to explore whether the proposed reusable artifact will be attractive to prospective developers. The research question is:

*Question 3. How can we determine whether the proposed reusable artifact is better and more useful than the existing artifacts?*

The objective is:

*Objective 3. To explore adoption intention of prospective developers with respect to the proposed reusable artifact.*

The approaches we propose must be automatable to ensure practical application, and their effectiveness will be demonstrable via appropriate validation procedures. The research questions, objectives, and hypotheses corresponding to each question are stated as shown on Figure 1.1 below.
1.4 Scope and Contributions

Following its increasing acceptance for new system development and dictated by our choice reusable artifact (analysis patterns), the paradigm for conceptual design that we use in this study is object-orientation (Booch, 1994; Booch et al., 1999). The reuse
approaches that we propose to develop are aimed at individual designers working on moderate size problems. Although the proposals may be extended to larger designs that are developed by the team of designers, such extensions are beyond the scope of this research, which is focused on demonstrating, by proof of concept, novel technological approaches to facilitate reuse at the individual level. The study is aimed at early, conceptual design where the potential for improvement is considered the most, and does not cover later stages, such as implementation, for which existing approaches can be used through code reuse.

The main contribution of this research is to provide solution approaches that can facilitate and improve the potential of reuse. Each solution approach will be developed, operationalized, automated, and subjected to appropriate validation procedures. The first phase will propose and test an automated design reuse process that can reduce the demands on a designer’s time and efforts to find, adopt, and integrate appropriate reusable artifacts for new design situations. The second phase will develop an inductive approach (instead of the current, highly manual approaches), developing a novel reusable artifact that can leverage knowledge from existing designs such as the ones created in phase 1. The third phase will explore adoption intention of designers to adopt the novel artifact proposed in phase 2.
CHAPTER 2
LITERATURE REVIEW

Following the research goals outlined in chapter 1, we can identify three research streams that are central to our concerns: object-oriented design, software reuse, and automation of design/reuse. Research from these streams is reviewed in this chapter. Techniques (e.g., machine learning) used to develop the solution approaches are briefly reviewed in Appendix G.

2.1 Object-Oriented Design

The two most common ways of modeling reality are from an algorithmic perspective and from an object-oriented perspective (Booch et al., 1999). In the algorithmic approach, the main building block of all software is the procedure or function. In the object-oriented approach, however, the main building block of all software systems is the object (or class). An object is a thing, generally drawn from the vocabulary of the problem space or the solution space.

Another key advantage is that the object-oriented approach facilitates reuse through the key properties of object-oriented paradigm. Encapsulation and information hiding prevents other modules from accessing a module’s internal details, forbids outside interference, and protects other modules from depending on details which may change (Booch, 1994; Taylor, 1992). This increases modularity and reusability of an object/class. Inheritance and polymorphism also support reusability (Lim, 1998). Inheritance allows data and methods defined in super classes, and reused in sub classes. Polymorphism enables different implementations to be hidden behind a common
The most important aspect of using object-oriented techniques is that elements in the information system be defined as objects that can interact with each other by sending messages. This view is closer to the way human beings perceive and interact with real-world entities. One benefit of this is that during design process, designers can continue to use the users’ vocabulary. Though this mapping is not exact, due to constraints such as platforms, performance, and generalization, the benefits of the object-oriented paradigm are not lost if these differences are documented (D’Souza & Wills, 1999). During object-oriented design, therefore, similar constructs are used to describe both concepts in the users’ domain as well as those in the software domain.

Unlike the algorithmic approach, the object-oriented paradigm supports multiple views of modeling real world. These include: (a) structural – what is known about an object and how they are arranged, (b) functional – how this system must react in response to events, (c) dynamic – how a society of objects interact with one another to satisfy this functionality, and (d) behavioral – how the sequences of states of an object are transformed in response to events (D’Souza & Wills, 1999). Perspective declarations about object-oriented development methods such as UML (Booch et al., 1999) call these (a) class diagrams, (b) use cases, (c) interaction diagrams, and (d) state transition diagrams. The representation modes to capture solution images devised by the designer can, therefore, be of many different kinds. Using multiple modeling techniques in this manner allows the designer to check for internal consistency of the specification and allows a more complete specification of the solution to emerge as each model informs the others.
Figure 2.1 Multiple Representations in Object-Oriented Design

Among these views, it is the class diagram that captures the implementable aspect of the system. The other perspectives allow the designer to explore, understand, simulate and check this underlying aspect. Accordingly, this research is focused on reuse of class diagrams for system development. A class diagram shows a set of classes, interfaces, and collaborations and their relationships (Booch et al., 1999). Graphically, a class diagram is a collection of vertices and arcs. Objects are represented as vertices and their relationships as arcs. Figure 2.1 above shows an example of the three models, and how
they inform one another.

2.2 Reuse

Software reuse is the process of creating software systems from existing software assets, rather than building software systems from scratch (Krueger, 1992). The potential for reuse is enormous, since the majority of each new application system could be assembled from reusable software parts, assuming, of course, that the appropriate parts could be predetermined, built, and made readily available to system developers. Furthermore overall product quality improves if quality components are reused.

Although the available literature on software reuse can be identified in many different directions\(^1\), we cannot deny that the driving force of reuse has been technological and methodological aspect of reuse. Meyer (1987) believes that technical/methodological factors are the major roadblocks to effective software reuse.

Research on reuse can be broadly classified into (a) for and (b) with reuse. The main objective of the former is development of reusable artifacts, while the latter is concerned with the development of systems with reusable artifacts.

2.2.1 Reusable Artifacts

The definition of software artifacts is not the same among researchers. Narrowly viewed, the artifacts mean only code components such as subroutine or object libraries.

---

\(^1\) Early works focused on definitions, concepts, frameworks, and scopes of software reuse (Biggerstaff & Perlis, 1989; Biggerstaff & Richter, 1987; Deutsch, 1983; Freeman, 1983; Horowitz & Munson, 1984; Matsumoto, 1984; Prieto-Diaz & Freeman, 1987; Wegner, 1983). Most dominant topics have been on the technological and methodological issues of software reuse (Chang & Eastman, 1993; Cox, 1990; Deutsch, 1989; Dubinsky et al., 1989; Freeman, 1987; Goguen, 1989; Kang, 1988; Krueger, 1992; Lenz et al. 1987; Mili et al. 1995; Prieto-Diaz & Neighbors, 1986; Rice & Schwetman, 1989; Volpano & Kieburzt, 1989). Some addressed organizational/managerial issues (Card & Comer, 1994; Davis, 1994; Fafchamps, 1994; Frakes & Isoda, 1994; Griss & Wosser, 1995; Wasmund, 1993).
(Gaffney & Durek, 1989). However, many studies propose the broader scope of software artifacts than just code components. For example, Biggerstaff and Perlis (1989) propose that “reused knowledge includes artifacts such as domain knowledge, development experience, design decisions, architectural structures, requirements, designs, code, documentation, and so forth.” Jones (1984) lists five artifacts: reusable data, reusable architecture, reusable design, reusable program, and reusable module. Goldberg and Rubin (1990) suggest five categories: algorithm reuse, reuse of classes and instances, reuse of application frameworks, reuse of full applications, and reuse of interface specifications. Leach’s list (1997) has probably the broadest perspective. He identifies a total of 18 reusable artifacts including reusable requirements, reusable documentation, reusable algorithms, reusable interface specifications, etc. as well as reusable architecture, reusable design, reusable modules, and so on.

This section describes the most common reusable artifacts; components, patterns, frameworks, and domain models.

### Components

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Name : char</td>
</tr>
<tr>
<td>- Address : char</td>
</tr>
<tr>
<td>- Telephone : int</td>
</tr>
<tr>
<td>+ create()</td>
</tr>
</tbody>
</table>

*Figure 2.2 A Class*

The term components can be used to indicate any reusable artifacts such as class libraries and frameworks as well as individual classes, regardless of the granularity level. Generally, a component is defined as a single class,  

---

2 The following list is Leach’s perspectives on the totality of reusable artifacts: reusable architecture, reusable requirements, reusable design, reusable programs/common systems, reusable modules, reusable transformation systems, reusable cost models/plans/schedules, reusable experiential/metrics/measurement data, reconfiguration of reusable systems, reusable data for use by programs, reusable documentation, reusable negotiations with customers, reusable negotiations with software vendors, reusable algorithms, reusable classes/instances, reusable interface specifications, reusable inputs to application generators, and reusable inputs to very high-level languages.
or more likely a small collection of classes, as it is used in most of component literatures. Compared to other larger granularity reusable artifacts, a component is considered a smallest unit of independent deployment. Thus, for a component to be composable with other components by a third party, it needs to be sufficiently self-contained (Szyperski, 1998). In general, it is known that it is difficult to locate appropriate components.

Business components are highly domain dependent. For more effective reuse, the use of higher granularity reusable artifacts is significantly more demanding than monolithic integrated component solutions. Figure 2.2 shows a business component, a Customer class.

**Analysis Patterns**

An analysis pattern is a group of communicating objects with stereotypical responsibilities and interactions. They provide generic solutions that can be applied by analogy in different domains. Analysis patterns provides a catalogue of patterns that have emerged in a wide range of domains including trading, measurement, accounting, and organizational relationships (Fowler, 1997; Coad et al., 1995). Coad et al. identified 31 analysis patterns. Figure 2.3 shows an example of an analysis pattern.

**Design Patterns**
Design patterns are defined in Gamma et al. (1995) as “descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.” Design patterns allow programs to share knowledge about their design. They encourage large-scaled reuse of successful design outcomes to reduce unnecessary parts in design process. In their catalog, Gamma et al. identified 23 design patterns. Analysis patterns focus on object-oriented modeling, while design patterns focus on object-oriented design and programming. Most research in this area has focused on creating patterns for reuse, which has resulted in a number of libraries of patterns now being available (Coad et al., 1995; Fowler, 1997; “Portland,” 1997). There does not appear to have been research directed at the design of systems by automatically retrieving and synthesizing patterns. Figure 2.4 shows an example of a design pattern.

**Figure 2.4** A Design Pattern adopted from (Gamma et al., 1995)
A framework is a reusable, partially completed application that can be specialized to produce custom applications (Johnson & Foote, 1988; Fayad et al., 1997). It captures many patterns of interaction between objects and consists of a suite of concrete and abstract classes, explicitly designed to be used. A framework can also include additional utilities to aid in the completion of end-user applications (Rogers, 1997). In contrast to smaller artifacts, frameworks are targeted for particular business units (such as data processing or cellular communications) and largely independent of the domains to which they are applied to (Fayad & Schmidt, 1997). Framework reusability leverages the domain knowledge and prior effort of experienced developers in order to avoid recreating

![Diagram of persistence framework design](Heinchkiens, 1999)
and revalidating common solutions to recurring application requirements and software
design challenges. However, application developers in more complex domains have
traditionally lacked standard “off-the-shelf” frameworks. Designers should go through
highly technical and complex adaptation processes. Currently, there are no widely
accepted standards for designing, implementing, documentation, and adapting
frameworks. Figure 2.5 shows an example design of a persistence framework.

A domain model is a problem-oriented architecture that captures the similarities
and variations of the family of systems that compose the application domain (Gomaa,
1995). Like frameworks, domain models have a large granularity and are the most
abstract reusable artifacts. In this approach, an application system is constructed not by
reusing objects directly, but by using standardized generic models in a domain. The
benefit is that reusable models are easy to understand and widely applicable. On the
contrary, this approach is more difficult to customize the models to apply in practice
rather than using smaller units like components.³

2.2.1.1 Comparison of Reusable Artifacts

In general, low granularity artifacts such as code fragments and individual objects
are domain-independent. Domain dependent reuse indicates reuse within a specific
application domain. In this case, the semantics of the components are domain-dependent.
Searching for domain-dependent components is very costly. Large granularity artifacts
such as domain models and frameworks are highly domain-dependent. Domain
independent reusable artifacts can be reused regardless of the application domain.

³ Example of domain model is omitted. Domain modeling is almost a full scaling modeling. Domain
models can be represented in various diagrams such as object relationship diagram and interaction
diagrams (“A Domain Model,” 1997).
Artifacts of this type are highly abstract with common behavior throughout various domains. However, the abstraction level of the common models is so high that those models require much work to modify and apply them to the real applications. Figure 2.6 shows the relative position of three reusable artifacts, which are components, patterns, and models, in terms of domain dependency, granularity, and application dependency.

Also, different stages of systems development are related to the size of artifacts. For example, large granularity artifacts such as domain models and frameworks can be used in early stages like analysis and design phases. They can be used when creating the system project plan to initially identify opportunities for reuse in the system project (McClure, 1997). Meanwhile, small granularity artifacts like code fragments are used at later development stages such as implementation and maintenance. Figure 2.6 also illustrates the general use of reusable artifacts in different development phases.

Finally, reusable artifacts are compared in terms of desirable properties for future reuse. Table 2.1 summarizes each reusable artifact in terms of three criteria: usability, (re)usefulness, and efficiency (Mili et al., 1995). Coarse artifacts such as frameworks and domain models have high/medium ease of retrieval and medium ease of assembly.
Meanwhile, medium artifacts such as analysis patterns and design patterns, and fine artifact like components have low/medium ease of retrieval and assembly. It means that coarse artifacts have higher usability than medium/fine artifacts. However, components have low abstractness, and thus, is to reuse. In efficiency, all artifacts requires high efforts for creation and reuse.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>Ease of Retrieval</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Ease of Assembly</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>N.A.</td>
</tr>
<tr>
<td>(Re)usefulness</td>
<td>Granularity</td>
<td>Fine</td>
<td>Medium</td>
<td>Medium</td>
<td>Coarse</td>
<td>Coarse</td>
</tr>
<tr>
<td></td>
<td>Abstractness</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Creation Effort</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Reuse Effort</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 2.1 Comparison of reusable artifacts in terms of desirable properties

2.2.1.2 Approaches to Developing Reusable Artifacts

For developing reusable artifacts, the dominant methodology is *domain analysis*. Domain analysis identifies and abstracts common features of a set of already developed systems within the domain (Prieto-Diaz, 1990). Inputs to domain analysis can be various sources of domain knowledge and output is domain models. Sources of domain knowledge are literatures and deductive requirements from domain experts. Most of current studies (Neighbors, 1989; Prieto-Diaz, 1987; McCain, 1985; Lubars, 1991) suggest frameworks and focus on normative methodologies for domain analysis as a part of systems development life cycle.
With these approaches, the designer must have clear knowledge about domain, and consider the boundaries of what objects to include and to what degree they should be abstracted (Lubars, 1991). Ideally, the resulted design must be the best design in the domain, but it is not always guaranteed. However, domain analysis approach does not seem to be a practically efficient approach. It takes too much time and efforts of designers to develop reusable artifacts. Retrieval of artifacts resulted from these approaches also requires time and efforts. This is a significant drawback of the existing approaches that we attempt to address in this research.

2.2.2 Designing with Reusable Artifacts

In general, there are two approaches with reuse: component-based and model-based. The component-based approach involves assembly of existing components to create new applications. The model-based approach involves adapting standard, generic domain models to create new applications. Both approaches have advantages and disadvantages. Components are difficult to understand and search for, but do not require a great deal of modification. Models are easier to understand, but require adaptation to and instantiation in a specific environment. The model-based approach reuses a model itself at a high level of abstraction, not at the individual object level. The component-based approach has been adopted already to the actual system development at the implementation phase.

Building new applications with reusable artifacts need not follow a radically different life cycle from building new applications without reusable artifacts (Arnold & Stepoway, 1987; Burton et al., 1987; Jones, 1990). However, the problem of designing
systems with reuse involves three reuse-specific interlocking, although the level of involvement may different according to approaches: (1) retrieval of relevant reusable artifacts, (2) adaptation of the retrieved artifacts as appropriate in the problem domain being considered, and (3) integration of the retrieved artifacts with other parts of the solution. Retrieval refers to the search for potentially useful artifacts. Adaptation is the understanding and modification of retrieved artifacts to ensure that, taken together, they satisfy the developer’s requirements. Understanding selected artifacts is a costly activity and often involves exploring lower-level parts of the component such as the code (Karlsson, 1995). After understanding, modification is followed to make them fit the requirements. In most cases, the selected artifacts cannot be carried from one application to another without changes. This process involves removing the inappropriate parts and replacing them; this may in turn affect other parts of the component, provoking additional adaptations. Finally, integration is a step to verify whether the retrieved artifacts is compatible with its environment, and to composite them into the application.

2.2.2.1 Retrieval

To retrieve appropriate artifacts, there should be a match between the description of the requirements and the descriptions of the artifacts in the artifact library. Mili et al. (1995) formalized the artifact\(^4\) retrieval problem. According to them, there is the distinction between a problem space and a solution space. Problem space is further divided into actual problem space, problem space as understood by the designer, and query space, which consists of the developer’s perceived need’s translation into a

\(^4\) Mili et al. focused on the low level granularity (building-blocks approach) artifacts, and explained that the high level granularity artifacts (generative approach) do not affect much the steps that it does not automate.
“query” that the component retrieval system can understand. The solution space is also divided into sub spaces; *artifact instances space, artifact class space, and codes/indices space*. The artifact instances space consists of artifacts, some of which may be equivalent in some respects. Within the class space, these artifacts are represented by the same class. The codes space consists of the descriptions of the artifact classes using an encoding or indexing language. In practice, the encoding step inevitably results in a loss of information. In the best case, “indexing” encodes only a subset of the properties of an artifact class. Then the appropriate artifact(s) are retrieved when an encoded description of the designer’s need (query) is matched to the encoded description of the artifacts in the library. Figure 2.7 shows this model.

![Figure 2.7 A model of artifact retrieval adopted from (Mili et al., 1995)](image)

2.2.2.2 *Adaptation*

Adaptation is the process of understanding the retrieved artifacts and modifying in case the artifacts can not be used as is. The need for modification becomes clear for two cases: when the encoded description of the artifact does not match perfectly the query, and when the artifact whose encoded description does match the query may be inadequate (Mili et al., 1995). In either case, it indicates the mismatch between a query
and the encoding of an artifact. Mili et al. explain modification problem in the context of a transformational view of software development, in which software development is seen as a sequence of transformation (Agresti, 1986).

There are rule of thumb for the change in many ways that can be applied to the modification to the artifacts: 1) localizing the effects of changes, thus guiding reusers in the process of adapting retrieved artifacts to their needs (Moriconi & Winkler, 1990; Podgurski & Clarke, 1990), 2) simplifying program structures, which enhances program reusability and understandability (Huang, 1990), and 3) slicing programs to extract specific functionalities, in case the retrieved artifact does more than what is required (Gallagher & Lyle, 1991; Huang, 1990).

2.2.2.3 Integration

Integration involves the composition verification and validation problem. Given a set of artifacts and a schema for composing them, it checks that the proposed composition is feasible (verification) and satisfies a given set of requirements (validation). There are two general methods for describing compositions of artifacts; either at the specification level and at the realization (implementation) level (Mili et al., 1995). Specification languages usually provide built-in composition operators with well-defined semantics. When compositions takes place at the artifact realization level, a much smaller range of behavioral compositions are obtained, but we are assured that these compositions are feasible without additional development. Compositions are usually described using the so-called module interconnection languages (Hall & Weedon, 1993; Prieto-Diza & Neighbors, 1986). Module interconnection languages describe module compositions by
specifying: 1) the \textit{obligations} of the individual participants and 2) the \textit{interactions} between the artifacts.

Implementation of reuse (with reuse) would not be easy if it is separately considered from building reusable artifacts (for reuse). This study integrates and automates two aspects of reuse by providing built-in retrieval mechanisms in a new kind of reusable artifacts.

\section*{2.3 Reuse Automation}

Known problems corresponding to barriers in each tasks with reuse (described in chapter 1) indicate that the current approaches to reuse are not practical. Also when the number of artifacts in the library is large, designers can no longer afford to examine and inspect each artifact individually to check its appropriateness. Consequently, an automated method to perform reuse seems to be an only practical solution (Mili et al., 1995). In the following sections, known problems are described first, and then the current approaches to reuse automation and their problems are discussed.

\subsection*{2.3.1 Known Problems}

Three known problems corresponding to each tasks with reuse are: 1) mismatch between problem spaces and solution spaces (in retrieval), 2) inability of designers to reasoning with analogy (in adaptation), and 3) integrating reuse with existing work practice (in integration).
First, in general, queries seldom return software components that fit the needs exactly. It may be caused by both sides. Problem may not be presented accurately due to a not strong enough query language, inaccurate query representation of problem, etc. Also poor encoding that materializes classes in the solution space may lose information. Accordingly, retrieval takes place by comparing the query to the approximate descriptions of component classes (Mili et al., 1995). Text and lexical-based encoding and retrieval suffers from a number of problems. First, an agreed (or agreeable) vocabulary must be developed. That is both labor-intensive and conceptually challenging. There are trade-offs between precision and size of vocabulary and choice between what is referred to as pre-coordinated or post-coordinated indexing (Sorumgard et al., 1993). Software-specific challenges include the fact that one-word or one-phrase abstractions are hard to come by in the software domain (Krueger, 1992; Sorumgard et al., 1993). Further, queries tend to be fairly tedious to enter. With text and lexical description-based methods, retrieval algorithms treat queries and codes as mere symbols and the codes are inherently ambiguous and imprecise. By contrast, specification languages have their own semantics within which the fitness of a component to a query can be formally established (Chen et al., 1993; Mili et al, 1994; Zaremski & Wing, 1993).

Second, designers, especially inexperienced designers, have difficulties in understanding the full analogy with the functionality and the domain of the reusable artifacts (Maiden & Sutcliffe, 1992). Designers appear to exhibit a mental laziness which was manifest in copying rather than reasoning while reusing the reusable artifacts. A frequent mistake made by designers is to focus on surface, lexical properties of the
reusable artifacts, whereas successful reuse requires comprehension of deeper analogous concepts. Also designer’s mental model formation can be error-prone and difficult.

Finally, composition may occur either at the specification level or at the realization level (implementation) level as described in section 2.2.2.3. The problem with specification-level composition is that it is often difficult to characterize specification level manipulations by manipulations on the actual realizations of these specifications (Mili et al., 1994). Another problem in this process is how to process the compositions when none of the individual artifacts matches the user query. As for this matter, Hall (1993) describes an artifact retrieval method that explores combinations of artifacts. Mili et al. (1995) work on a combination of Hall’s work and Zaremski and Wing’s work (1993) on signature matching. They showed that the set cover problem, which is known to be NP-complete (Garey & Johnson, 1979), could be reduced to the function realization problem.

2.3.2 Approaches to Reuse Automation

Automating reuse process requires encoding the problem space and the solution space, followed by design of a matching algorithm. The choice of the encoding methods and of the matching algorithms for the artifact retrieval involves a number of trade-offs between cost, complexity, and retrieval quality (Mili et al., 1995). Mili et al. classify encoding and retrieval approaches into three classes: 1) text-based encoding and retrieval, 2) lexical description-based encoding and retrieval, and 3) specification-based encoding and retrieval.

---

5 The function realization problem consists of finding all the compositions of functions (signatures) that consume no more than the inputs specified by the designer’s query and produce at least the outputs specified by the designer’s query (Mili et al., 1994).
With text-based encoding and retrieval, arbitrarily complex string search expressions supplied by the reuser are matched against this textual representation of an artifact. In this approach, no encoding is required, and queries are fairly easy to formulate. Its disadvantage is that plain-text encoding is neither sound nor complete. This approach and variants have been used in a number of studies (Frakes & Nejmeh, 1990; Yoelle et al. 1991; Mili et al., 1994; Frakes & Pole, 1992).

With lexical descriptor-based encoding, typically, subject experts inspect the artifacts and assign to them key phrases taken from a predefined vocabulary that reflects the important concepts in the domain of discourse (Arnold & Stepoway, 1987; Burton et al., 1987; Mili et al., 1994; Prieto-Diaz & Freeman, 1987). With this approach, an agreed vocabulary has to be developed that is both labor-intensive and conceptually challenging (Sorumgard et al., 1993). Further, from the reuser’s point of view, a familiarity with the vocabulary is needed.

With text and lexical descriptor-based methods, any meaning assigned to queries, artifact codes, or the extent of match between them is external to the encoding language. By contrast, specification-based languages have their own semantics within which the fitness of an artifact to a query can be formally established (Chen et al., 1993; Mili et al., 1994; Zaremski & Wing, 1993). Specification-based methods with behavioral specification (and not just signatures) suffer from considerable costs. First, there is the cost of deriving and validating formal specifications for the artifact of the library (Moineau & Gaudel, 1991). The second cost has to do with the computational complexity of proof procedures. And there is the cost for the reuser to write full-fledged specifications for the desired artifacts.
When text or lexical-based encoding scheme is used, a full-fledged language understanding system would be required. Many text-based and lexical-based encoding and retrieval systems suffer from vocabulary problems, power of language problems, etc. in both problem space and solution space. Specification-based encoding and retrieval systems can avoid these problems, but still require designer to specify components in a formal or mathematical language, and to create queries to retrieve components to fit. Also modifying reusable artifacts may degrade both the quality and the productivity advantages of reuse. This study explores the way to remove these steps and seeks a full automation of encoding and retrieval without designer’s efforts. By automation, it is expected that possible causes of retrieving irrelevant results will be greatly reduced and that the time required for searching component can be also enormously reduced. This also influence adaptation and integration process. Basically, high modification rate of retrieved component is caused by low matching results between query and components retrieved. Also one of major integration problem is NP-complete combination problem when the specified query retrieves low relevant components. Thus, high recall and precision rate of retrieval indicates less modification of components retrieved and less integration problem.

Also most of prior studies have been carried out on the component level. They have focused on encoding and specifying individual components, and consequently have had those lexical problems and mismatch in query and specification. This study creates and reuses design level artifacts rather than component level classes. Specification and retrieval are made on the design level, and individual component classes are not specified
separately and retrieved accordingly as part of design. This is expected to provide higher reuse leverage and less error margin for the problems described above.

### 2.3.3 Tools for Reuse-based Design

There are also several studies about reuse tools among systems development stages, regardless of tasks above. Purao and Storey (1997a; 1997b) automated analysis patterns to synthesize the object-oriented conceptual design. In their approach, first, identifying objects from the requirement statements requires *natural language processing (NLP)* to parse the statements into significant keywords, and eventually objects. Based on the objects identified, analysis patterns are retrieved from the pattern base, which is a *relational database*, instantiated, and synthesized into a complete design. A library of analysis patterns (similar to Coad et al., 1995) forms the basis of the pattern base. It contains 31 patterns; that is, groups of genetic objects such as ‘Actor,’ ‘Participant,’ ‘Transaction,’ ‘Place,’ etc. This study extends their work for generating and organizing reusable designs.

Bansiya (1998) presents DP++, a tool that automates design pattern detection, identification, and classification in programming language (C++). Patterns are identified based on key structural and functional relationships between classes and objects, which are interfaces (abstract classes), base classes and subclasses, template classes, inheritance relationships, aggregation by physical containment of instance variables, aggregation by references/pointer instance variables, and method parameter-based uses relationships. In the other hand, Budinsky et al. (1996) discuss automatic code generation from design patterns. The user of the tool supplies application-specific information for a given
pattern, from which tool generates all the pattern-prescribed code automatically. This could be easier to do than the reverse engineering as in DP++. Florijn et al. (1997) propose a tool supporting the use of patterns both in forward engineering and backwards engineering. However, the focus is on developing object-oriented programs with patterns.

All these studies focus on the conversion between patterns and implementation (programming codes). No study has been found yet focusing on automating generation of design patterns as in this study. Those studies above work based on the 23 basic design patterns derived from Gamma et al. (1995). Consequently it requires the user to input his/her own specifications and to continuously modify the basic patterns. Automation of coding from patterns may be more straightforward than automation of generating design models, and eventually family of design patterns. (Or at least different class structures, namely family of class definition in design patterns.) It could be the reason that those studies were not aided by learning algorithms.

2.3.4 Automated Design

Since we focus on the conceptual design stage of the development process, our problem can also be considered an extension of research in automated conceptual modeling (Bubenko & Wangler, 1992). Current approaches in this stream assist a designer in developing the conceptual specification of an application from natural languages assertions about the problem (Lowery & McCartney, 1991). The assertions are used to build a conceptual schema incrementally, through an interactive dialog with the designer to obtain additional information. Since a conceptual and lexical model does not
exist (*a priori*), research in this area tries to combine results in computational linguistics with practical needs as well as theoretical advances in conceptual modeling (1).

Representative of this research are prototypes such as the View Creation System (Storey & Goldstein, 1988), Computer-Aid for ER modeling (Hawryszkiewycz, 1985), Expert Database System (Choobineh et al., 1998), and others. These systems aid the designer in articulating the requirements and structuring the information gathered through such articulation. Several extensions to advancing this research area have been suggested. These include exploiting domain knowledge, use of functional knowledge, and semantics-based enhancements.

Most of these automated approaches, however, are considered to mimic the behavior of designers only at the novice or intermediate level (Storey et al., 1995). Rarely, if ever, expert designer behaviors are encoded in these approaches. These approaches struggle to achieve expert level even at the logical level. To provide intelligent design support at the conceptual design stage, it is necessary that expert design behaviors should be represented in the automated approaches (Storey et al., 1995).

Considering the advantages of reuse in early stage of the development life cycle, the target of this study is design reuse. Automation of design reuse is still in early stage. This study will start from the effort of improving a recent design automation research, which uses analysis patterns, and try to propose a full scale automation mechanism integrating building and implementing reusable artifacts. The conceptual designs used in this study are class diagrams, which involve intrinsic advantages inherited from object-oriented paradigm.
CHAPTER 3
RESEARCH METHODOLOGY

The primary research methodology followed for this research is dictated by the nature of the problem. In general, research problems may be divided into explanatory and improvement (Vaishnavi et al., 1995). In the former, the focus is on discovering theory and explanations for the phenomenon under review, whereas in the latter, the focus is on how we can solve some perceived problem. Improvement research can be thought of, in the Simonian Sense (Simon, 1981), where we choose to solve the problem through the creation of an artifact that serves as an interface between the inner environment of the artifact and the outer environment where the artifact must operate within.

This research can be classified as improvement research since it focuses on “how” to solve the problems posed by the research questions, and proposes the development of a prototype system as a device for embodying our solutions to these problems. The methodology for improvement research can be articulated, following Baldwin & Yadav, (1994) as consisting of the following steps: 1) problem formulation, 2) model development for the proposed solution and system architecture, 3) prototype system development based on the proposed model, 4) system testing, and 5) evaluation and validation of the results.

3.1 Architecture of the Proposed Solution Approach

Figure 3.1 shows our proposed approach to solving the problems proposed by the research questions. Phase 1 addresses the first research question: How can we improve
the naïve approach to conceptual system design with reuse of analysis patterns? As in
the naïve approach, inputs to this phase consist of the problem descriptions as natural
language assertions from the designer. The problem description is processed and analysis
patterns from the pattern library are synthesized by pattern synthesis engine. The process
is augmented with the help of learning mechanisms that leverage the history base. The
stored design history, in turn, is used by learning mechanisms to synthesize analysis
patterns. The expected output is an object-oriented conceptual design. In this manner,
the naïve reuse-based design process (Purao & Storey, 1997a; 97b) will be improved with
learning mechanisms, classified into two categories, supervised and inductive learning.
To demonstrate that the augmented approach is more effective, a controlled experiment is
performed. The objective of the first phase, therefore, is to develop an augmented
approach to design with reuse incorporating learning mechanisms that exploit prior
experiences.

Phase 2 addresses the second research question: How can we develop a new, more
effective artifact for reuse, by leveraging the knowledge in conceptual designs assembled
in phase 1? Inputs to this phase are the conceptual designs (similar to those) generated in
phase 1. The designs are analyzed and used to create a new type of reusable artifact by
using two kinds of clustering algorithms; keyword-based clustering and common pattern-
based clustering. These are combined to create a coherent methodology for automatically
generating reusable artefacts (design fragments) that have built-in mechanisms to
facilitate reuse. A simulation process is developed to operationalize the methodology so
that a repository of design fragments can be automatically created. Design fragments in
the repository are retrieved by a simple algorithm based on the problem description for a
new system. The outputs expected are design fragment(s) and/or design(s) that are retrieved from the repository. To demonstrate the retrieval potential as well, simulation is performed. The objective of the second stage, therefore, is to propose a novel reusable artifact with built-in implementation mechanism.

Finally, phase 3 addresses the third research question: How can we determine whether the proposed reusable artifact is better and more useful than the existing artifacts? This phase is intended to explore appropriateness of the reusable artifacts created in phase 2. Drawing on Technology Acceptance Model (TAM), we develop a model to predict intention of prospective designers to adopt the proposed reusable artifact. A preliminary analysis is performed to understand developers’ intentions. The objective of the third stage is to explore adoption intention of prospective developers with respect to the proposed reusable artifact.

Figure 3.1 below shows the proposed solution architecture with a few additional details about the solution approaches. These are briefly explained next and elaborated in the relevant chapters to follow.
To develop an augmented approach to design with reuse incorporating learning mechanisms that exploit prior experiences.

To propose a novel reusable artifact with built-in implementation mechanism.

Extensive simulation to populate/retrieve the repository.

To explore adoption intention of prospective developers with respect to the proposed reusable artifact.

Development of model based on TAM, by building (re)usefulness and reusability.

Figure 3.1 Proposed Architecture
3.2 Solution Approaches

Table 3.1 summarizes techniques that will be used to develop the solution for each phase. The detailed solutions for each phase are described in the following chapters.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Research Question</th>
<th>Solution Approach</th>
<th>Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>How can we improve the naïve approach to conceptual system design with reuse of analysis patterns?</td>
<td>To develop an augmented approach to design with reuse incorporating learning mechanisms that exploit prior experiences.</td>
<td>Machine Learning - Reinforcement/ supervised learning; Relevance-based/ inductive learning</td>
</tr>
<tr>
<td>2</td>
<td>How can we develop a new, more effective artifact for reuse, by leveraging the knowledge in conceptual designs assembled in phase 1?</td>
<td>To propose a novel reusable artifact with built-in implementation mechanism.</td>
<td>Conceptual Clustering - Keyword-based clustering; Common pattern-based clustering</td>
</tr>
<tr>
<td>3</td>
<td>How can we determine whether the proposed reusable artifact is better and more useful than the existing artifacts?</td>
<td>To explore adoption intention of prospective developers with respect to the proposed reusable artifact.</td>
<td>Construct Development following - Technology Acceptance Model (TAM)</td>
</tr>
</tbody>
</table>

Table 3.1 Research questions, Solution approaches, and Techniques

In phase 1, the study develops learning mechanisms that help designers semi-automatically assemble object-oriented conceptual designs, based on the usage history. Two kinds of machine learning techniques are developed: reinforcement/supervised learning and relevance-based/inductive learning. The former exploits the usage history, which stores the inputs and outputs of each task and the interaction with the designer for each design session. The latter also utilizes the usage history, but in addition, exploits the structure of patterns in the pattern library, such as the participation of an object in multiple patterns. In phase 2, the study develops clustering algorithms to identify design
fragments and organize/store them into a repository. The clustering algorithms are composed of two techniques; keyword-based clustering, and common pattern-based clustering. Designs are classified based on keywords in the problem description. The proposed new reusable artifacts, called design fragments, are identified based on common patterns among designs. And all related algorithms including retrieval of design fragments are simulated. In phase 3, the Technology Acceptance Model (TAM) is used to propose the constructs of (re)usefulness and reusability of design fragment as possible predictors of the intention to use.

3.3 Prototype Development

The study develops multiple research prototypes to materialize the proposed approaches. Each solution approach, including learning mechanisms in phase 1 and clustering algorithms in phase 2, is implemented in Java™. The data and repository, including pattern library, usage history, and design fragment repository, are structured in a relational database. Figure 3.2 shows implementation choices for the prototype system.
To develop an augmented approach to design with reuse incorporating learning mechanisms that exploit prior experience.

To explore adoption intention of prospective developers with respect to the proposed reusable artifact.

To develop an augmented approach to design with reuse incorporating learning mechanisms that exploit prior experience.

Development of model based on TAM, by building (re)usefulness and reusability.

Extensive simulation to populate/retrieve the repository.

To propose a novel reusable artifact with built-in implementation mechanism.

Conceptual Design Pattern Synthesis Engine

Clustering Algorithms

Keyword-Based Clustering

Common Pattern-Based Clustering

Pattern Library

Implemented in Database

Implemented using Java

Phase 1

Problem Description

Pattern Synthesis Engine

Implemented in Database

Design History

Implemented in Database

Implemented using Java

Phase 2

Implemented in Database

Reactive Design Fragments Repository

Implemented using Java

Phase 3

Problem Description for New Application

Implemented using Java

Implemented using Java

New Conceptual Design

Retrieval Algorithms

Implemented using Java

Figure 3.2 Prototype Development
3.4 Validation

Table 3.2 summarizes validation approaches for each phase. Empirical testing is performed to assess the effectiveness of the learning mechanisms in phase 1. Extensive simulation is performed to access the effectiveness of building the design fragment repository and its retrieval mechanisms in phase 2. A lab experiment is performed to explore (re)usefulness and reusability of design fragments in phase 3. Additional details about these are presented in the relevant chapters.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Research Objectives</th>
<th>Corresponding Assessment</th>
<th>Validation Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To develop an augmented approach to design with reuse incorporating learning mechanisms that exploit prior experiences.</td>
<td>Demonstrate that the augmented approach is more effective for creating conceptual designs than the naive approach.</td>
<td>Implementation, Demonstration with multiple cases, Lab Experiment</td>
</tr>
<tr>
<td>2</td>
<td>To propose a novel reusable artifact with built-in implementation mechanism.</td>
<td>Simulate to create a repository of reusable artifacts and demonstrate, by multiple examples, design with reuse of these artifacts.</td>
<td>Implementation, Simulation, Demonstration with multiple cases</td>
</tr>
<tr>
<td>3</td>
<td>To explore adoption intention of prospective developers with respect to the proposed reusable artifact.</td>
<td>Build and explore a model for predicting adoption intention of prospective developers with respect to the proposed reusable artifact</td>
<td>Model Development, Exploratory Experiment</td>
</tr>
</tbody>
</table>

Table 3.2 Research Objectives, Hypotheses, and Validation Approaches
CHAPTER 4
IMPROVING REUSE-BASED DESIGN:
AUGMENTING ANALYSIS PATTERNS REUSE WITH LEARNING

4.1 Introduction

The software industry is under considerable pressure to meet increased demands [Lapointe 1999]. Although productivity has steadily increased over the past 30 years, the gap between demands placed on the software industry and what the state of practice can deliver continues to widen [Gibbs 1994, Pearson 1999]. Several decades of research confirm that reuse is the only realistic approach to bridging this gap [Krueger 1993, Mili et al 1995]. Achieving successful reuse involves overcoming a number of obstacles [Jacobson 1996, Kim and Stohr 1998]. Creating a design from existing artifacts is similar to an NP-complete set cover problem [Mili et al 1995, Brodie 1998]. Effective reuse, therefore, requires tools and techniques that can assist the designer [Nierstrasz and Meijler 1995].

Software patterns [Appleton 1997] have emerged as a viable alternative to facilitate reuse. They have been identified for conceptual design in the form of analysis patterns [Coad et al 1995, Fowler 1997] and for detailed design in the form of design patterns [Gamma et al 1995, Pree 1994]. Both represent generic solutions that can be applied, by analogy, in different domains [Gamma et al 1995]. Analysis patterns represent groups of generic objects, e.g. ‘Actor’ and ‘Transaction,’ with stereotypical properties and responsibilities. Libraries of analysis patterns have been created for reuse [Coad et al 1995, Fowler 1997, Portland 1998].

Several naïve approaches to support new system design with reuse of analysis patterns have been developed. They utilize simple natural language processing (NLP)
techniques (e.g., Purao and Storey [1997a], Wohed [2000]) without taking into account contextual factors such as differences in industry and application domains [Glass and Vessey 1996] or designer preferences. Neither do they emulate the learning that can occur through repeated application and designer interaction, such as different propensities that promote or hinder the use of specific patterns in different design problems. In this research we propose an approach to overcoming these shortcomings.

The objective of this research, therefore, is to: improve the results of conceptual software design based on reuse of analysis patterns by developing an augmented approach that incorporates learning mechanisms to exploit prior experiences.

Specifically, the research goals are to:

• develop learning mechanisms to improve reuse-based design with analysis patterns,
• coalesce these mechanisms into a coherent approach and demonstrate its feasibility, and
• assess the effectiveness of the augmented approach through empirical testing.

The next section defines analysis patterns and discusses naïve reuse-based design with analysis patterns. Section 3 provides the overall architecture of the augmented approach and develops the learning mechanisms. Section 4 describes application of the learning mechanisms. The results from empirically testing the prototype are presented in section 5. Section 6 concludes the paper.

4.2 Prior Research

Prior research has focused on automation of and/or assisting the designer in reusing patterns at the design and implementation stage. Florijn et al [1997] outline approaches to assisting the software designer with tools that can retrieve design patterns. Bansiya [1998] describes a tool to automatically retrieve design patterns to better
structure C++ programs. These efforts, however, do not address the conceptual software design stage, where the impact on productivity is highest [Denis and Wixom 2000, Alter 1999]. Consequently, the need for research to improve reuse (of analysis patterns) during conceptual design is well recognized [GUIDE 1997a, 1997b, Nierstrasz and Meijler 1995].

4.2.1 Conceptual Design with Analysis Pattern Reuse

An analysis pattern is a group of related, generic objects (e.g. ‘Actor,’ ‘Group,’ ‘Transaction’ etc.) with stereotypical attributes (data definitions), and behaviors (method signatures) defined in a domain-neutral manner [Coad et al 1995, Fowler 1997]. Figure 4.1 shows the analysis pattern ‘Participant-Transaction,’ and lists other patterns [Coad et al 1995]. The pattern, ‘Participant-Transaction’ is defined at a high level of abstraction so that it can be used, by analogy, in different domains, to create patterns such as ‘cashier-sale’ or ‘clerk-reservation.’ Information contained in an analysis pattern can, thus, be identified at different levels of abstraction (see table 4.1).

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>Participant -- Transaction</td>
</tr>
<tr>
<td>Object</td>
<td>Participant</td>
</tr>
<tr>
<td>Instantiation</td>
<td>Cashier</td>
</tr>
</tbody>
</table>

Table 4.1 Elements of Analysis Patterns

Abstractions and analogies represent the basis on which reuse of analysis patterns can be carried out [Bhatta and Goel 1997]. Designing systems with reuse of analysis patterns, therefore, involves retrieval of relevant patterns, and their use and synthesis in the problem domain [Purao and Storey 2000]. This requires a specific decomposition of
the problem to create slots into which the patterns can be fitted, making the problem
similar to the set cover problem [Mili et al 1995], which is NP-complete [Garey and
Johnson 1979]. Related research, automated conceptual modeling [Bubenko and Wangler
1992, Rolland and Proix 1992], has proposed approaches that allow a software designer to
describe the problem as natural language assertions, which are then used to build a
conceptual design using NLP techniques. These, however, require the designer to
articulate everything, whereas an experienced designer often relies on episodic
knowledge [Schenk et al 1998] to ‘fill in the blanks.’ Analysis patterns allow such an
extension of natural language assertions made by the designer, thus providing an
opportunity to contribute to research on automated conceptual modeling.

A naïve approach to semi-automate conceptual design with analysis patterns is
suggested by [Purao and Storey 1997a, 1997b, Purao 1998]. It begins with simple natural
language assertions such as: “a system to track sales at different stores.” Using a library
of patterns (Coad et al [1995]), the approach, shown in figure 4.2, creates a preliminary
conceptual design. It first parses the natural language assertions retaining key words and

---

**Figure 4.1 An Analysis Pattern**

<table>
<thead>
<tr>
<th>Participant</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>number</td>
</tr>
<tr>
<td>startDate</td>
<td>date</td>
</tr>
<tr>
<td>endDate</td>
<td>time</td>
</tr>
<tr>
<td>password</td>
<td>status</td>
</tr>
<tr>
<td>authorization_level</td>
<td>aboutMe</td>
</tr>
<tr>
<td>howMany</td>
<td>calcForMe</td>
</tr>
<tr>
<td>howMuch</td>
<td>rateMe</td>
</tr>
</tbody>
</table>
calcOverTransactions, rankTransactions, inAuthorized

**Figure 4.2 A Naïve Approach**

0. Identification of Significant Words
1. Identification of Objects
2. Retrieval of Relevant Patterns
3. Instantiation of Relevant Patterns
4. Synthesis of Instantiated Patterns

Patterns Base
phrases such as ‘track,’ ‘sales,’ ‘different’ and ‘store’ (step 0). Based upon a simple lookup, it then identifies appropriate objects such as ‘Transaction’ for sale and ‘Place’ for store (step 1), and retrieves relevant patterns (step 2). For example, the object ‘Place’ suggests the pattern ‘Place-Transaction.’ The object ‘Transaction’ suggests other patterns, such as ‘Transaction-SubsequentTransaction,’ and ‘Transaction-TransactionLineItem.’ These patterns are then instantiated using the key words (step 3). For example, the pattern ‘Place-Transaction’ is instantiated as ‘Store-Sale,’ and ‘Transaction-TransactionLineItem’ is instantiated as ‘Sale-SaleLineItem.’ Finally, the retrieved and (sometimes partially) instantiated patterns are synthesized using overlapping objects (step 4). The process can iterate. Another naïve approach for analysis pattern retrieval is suggested by Wohed [2000]. Her approach interviews users to obtain answers about the ‘booking’ domain, which are then mapped to instantiations of patterns.

4.2.2 The Role of Learning

A naïve approach for the reuse of analysis patterns does not learn from one design session to the next, and thus, can make similar errors of omission and commission over multiple design sessions. Specifically, it does not learn about the applicability and use of different patterns in different domains. For example, the pattern, ‘Actor-Participant’ may need to be instantiated as ‘Customer-Cashier’ in a retail application, but as ‘Passenger-Ticket Agent’ in an airline reservation application. Learning mechanisms, properly integrated with a naïve approach, can provide intelligent suggestions to the designer, leading to a more accurate and complete conceptual design.
A learning model consists of four components as shown in figure 4.3. The performance element carries out the required tasks, e.g. retrieval of patterns. The learning mechanism tracks prior experiences and infers meaningful rules. The software designer plays the role of the critic who confirms or rejects inferences based on these rules. The problem generator corresponds to prior design experiences, which feed the learning mechanisms. Most learning mechanisms can be classified into two categories.

- **Reinforcement and Supervised Learning**: Reinforcement learning requires feedback in the form of approvals or rejections. For instance, based on such designer feedback in prior cases, the learning mechanism may adjust the suggestions, say, pattern retrieval in subsequent cases. Supervised learning refers to learning from outputs that are partially provided by an evaluator. For example, if the designer expands suggestions made by a learning mechanism for pattern retrieval, the learning mechanism may adjust the suggestions in subsequent cases.

- **Relevance-based and Inductive Learning**: These learning strategies continue to use reinforcement and supervised learning styles. In addition, relevance-based learning utilizes interdependencies between different fragments of knowledge, learned or initial. For example, inferences may be drawn for one part of a pattern (say, instantiation of an object) based on current design information for other parts of the pattern that match prior design experiences. Inductive learning further exploits the
‘learned knowledge’ to generate new knowledge. For example, if two objects, ‘Place’ and ‘Container’ share many examples (such as ‘store’) from prior experiences, this may be generalized in an inductive manner to suggest synthesizing patterns containing these objects.

Table 4.2 summarizes the learning strategies and their use.

<table>
<thead>
<tr>
<th>Learning Strategy</th>
<th>Technique</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reinforcement</td>
<td>Exploit Confirm/Reject Feedback</td>
<td>Useful for known situations</td>
</tr>
<tr>
<td>Supervised</td>
<td>Exploit Additional Alternatives Feedback</td>
<td></td>
</tr>
<tr>
<td>Relevance-based</td>
<td>Exploit Interdependencies</td>
<td>Useful for new situations</td>
</tr>
<tr>
<td>Inductive</td>
<td>Generate New Knowledge</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.2 Learning Strategies**

4.3 Improving Reuse-based Design with Learning

The role of the learning mechanisms is to act as *intelligent assistants*, making suggestions based upon one or more learning strategies. We use the problem-domain independent approach [Purao et al 1997a, Purao 1998] as the base approach for improvement through learning. The learning mechanisms, however, are domain-dependent, making information about industry and application domain [Glass and Vessey 1996] a necessary part of the augmented approach. Figure 4.4 outlines the augmented approach where the basic learning model (figure 4.3) is used to improve the naïve approach (figure 4.2). The learning mechanisms fall into five groups as shown in the figure. Each exploits some or all of the information available from the analysis patterns (table 4.1) and the learning strategies (table 4.2). Fifteen specific mechanisms were developed following this approach, which are discussed next.
4.3.1 Reinforcement and Supervised Learning Mechanisms

4.3.1.1 Augmenting Parsing of Natural Language Assertions

The first set of learning mechanisms applies to the identification of significant words (step 1 of the naïve approach) and are invoked in sequence.

- The *discard words* mechanism uses the history of words discarded by designers during previous sessions to provide intelligent suggestions for better parsing the
requirements. For example, “direct” or “like” do not add to the context and can be suggested as candidates 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 in the problem domain that the designer could not map directly to any objects. For example, ‘portfolio’ is an important concept in finance, but a designer may be unable to map it to an object. Under the naïve approach, the word is eliminated from consideration as the next task, identification of objects, is carried out. 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 the industry or application domain, that do not appear in the natural language assertions. For example, ‘gallery’ may be important for an art dealer but is not specified in: “tracking works from different artists, displayed at different places.”

Table 4.3  Learning Mechanisms for Augmenting Parsing

| 1. Discard | ∀ w ∈ [Words | learned] |
|            | If w ∈ [Discarded Words | learned] |
|            | Then Suggest was a candidate for Discarding |

| 2. Shield | ∀ w ∈ [Words | learned] ∈ [Object Examples | initial knowledge] |
|           | If w ∈ [Retained Words | learned] |
|           | Then Suggest was a candidate for Retaining |

| 3. Guide | ∀ kw ∈ [Domain-Dependent Keywords | learned] |
|          | If kw ∈ [Words | learned] |
|          | Then Suggest kw was a candidate for Adding |

4.3.1.2 Initiating the Design Process

The second set of learning mechanisms improves entry into the design process. Thus, mechanisms in this set primarily apply to step 2 (figure 4.3), although some also apply to other steps.
The competing objects mechanism applies to cases where one word identifies multiple generic objects. It uses information about domains to more accurately identify objects based on a significant word. For example, ‘store’ may map to ‘Place’ in the retail domain or to ‘Container’ for inventory control. It might even map to both for warehousing applications.

The competing words mechanism applies to the reverse cases, where more than one word from the natural language assertion suggests the same generic object. For example, ‘sale’ as well as ‘reservation’ may map to ‘Transaction’ from the assertion “keeping track of sales and ensuring that the inventory for each sale is reserved for pickup by the customer.”

The direct instantiation mechanism exploits a complete pattern and its example; that is, it suggests an alternative to performing the first three tasks simultaneously. For instance, if the requirements description contains ‘aircraft,’ this mechanism finds ‘aircraft-cargo’ and ‘aircraft-passenger,’ both instantiations of the pattern ‘Container-Content;’ and ‘aircraft-engine,’ an instantiation of the pattern ‘Assembly-Part.’ The learning mechanism then suggests these for inclusion in the current design.

### 4. Competing Objects

\[
\forall o, d \in [\text{Objects Identified} | \text{current}] \\
\text{If } \exists o, d | o \neq d | \text{identifier } w == \text{identifier } w (w \in [\text{Words} | \text{current}]) \\
\text{Then } \text{Rank using Domain-Specific Frequency, } \forall o | \text{identifier is identical} \\
\text{Suggest Retaining the top ranked object for the word}
\]

### 5. Competing Words

\[
\forall w, w' \in [\text{Words} | \text{current}] \\
\text{If } \exists w, w' | w \neq w' | \text{identified by } w == \text{identified by } w' \\
(\text{w} \in [\text{Objects Identified} | \text{current}]) \\
\text{Then } \text{Rank using Domain-Dictated Frequency, all w | identified by w is identical} \\
\text{Suggest Retaining the top ranked word for identifying the object}
\]
### 4.3.2 Relevance-Based and Knowledge-Based Inductive Learning

#### 4.3.2.1 Affinity-based Mechanisms

The third set of learning mechanisms exploits the co-existence of patterns and similarities among the instantiations of these patterns. It, thus, applies to steps 2 and 3 (figure 4.3).

- The *Pattern Affinity* mechanism uses information about patterns often used together in prior designs to suggest the inclusion of one pattern, if the other is included.

Consider the patterns ‘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 have never co-existed in prior designs, for example, patterns ‘Container-Content’ and ‘Assembly-Part.’ If they are retrieved for the current design, the designer is informed that this represents an anomalous situation.

---

### Table 4.4 Learning Mechanisms for Augmenting the Initiating of Design

<table>
<thead>
<tr>
<th>6. Direct Instantiation</th>
<th>( \forall \omega \in \text{Words} ) ( | \text{current} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \exists p_i \in \text{Prior instantiations, learned} ) ( | \omega \text{Instantiation} \in p_i = \omega )</td>
<td></td>
</tr>
<tr>
<td>Then ( \text{Suggest} ) ( o ) as candidate object for Identification with word ( \omega )</td>
<td></td>
</tr>
<tr>
<td>( \text{If} ) designer accepts</td>
<td></td>
</tr>
<tr>
<td>Then ( \text{Suggest} ) ( p ) as a candidate pattern for Retrieval</td>
<td></td>
</tr>
<tr>
<td>( \text{If} ) designer accepts</td>
<td></td>
</tr>
<tr>
<td>Then ( \forall o' \in p | o' \neq o \text{ Suggest} ) ( o'.\text{Instantiation} \in p ) as ( o'.\text{Instantiation} \in p )</td>
<td></td>
</tr>
</tbody>
</table>
The **Partial Instantiation Affinity** mechanism utilizes information about prior instantiations of a pattern. Since patterns are retrieved based on identification of an object, it is likely that initially, they will be only partially instantiated. For example, after ‘Place-Transaction’ is retrieved and instantiated based on the keyword ‘store’ (i.e. object ‘Place’), ‘Transaction’ remains un-instantiated. The pattern may have been instantiated in prior designs as ‘store-sale,’ ‘store-shipment,’ and ‘airport-reservation.’ Then, possible instantiations of ‘Transaction’ in the current design may be suggested as ‘sale,’ or ‘shipment.’

The **Inverse** mechanism represents a variation of the Partial Instantiation Affinity mechanism. It utilizes inverse relationships among pre-stored and learned keywords. For example, ‘Transaction-SubsequentTransaction’ might have been partially instantiated for the current design as ‘Sale-SubsequentTransaction.’ Then, the Partial Instantiation Affinity mechanism can be applied using the word ‘sale’ as well as its inverse ‘purchase.’ If any prior pattern instantiations are found that correspond to these words, the suggestions made to the designer will also include the inverse. 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, ‘order-shipment’ and ‘customer-salesperson’ represent related instantiations of the patterns ‘Transaction-SubsequentTransaction’ and ‘Actor-Participant’. They provide a plausible instantiation of one pattern if the other pattern has been used with the related instantiation. Clearly, this mechanism applies only to patterns that are not even partially instantiated upon retrieval (e.g. those suggested by
the Pattern Affinity mechanism). Consider the pattern ‘Participant-Transaction’ for the current design, which also contains the pattern ‘Assembly-Part’. Based on prior designs that share the current instantiation of ‘Assembly-Part,’ various instantiations of ‘Participant-Transaction’ are suggested to the designer, such as: ‘cashier-sale’, ‘customer-purchase’ or ‘employee-sale.’

<table>
<thead>
<tr>
<th>Pattern Affinity</th>
<th>( \forall d \in \text{[Prior Designs]} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall \text{pr} \in \text{[Patterns Retrieved</td>
<td>\textit{current}</td>
</tr>
<tr>
<td>( \text{If} ) ( \text{pr} \in \text{[Patterns Retrieved</td>
<td>\textit{learned}</td>
</tr>
<tr>
<td>( \text{Then} ) ( [\text{Comparable Designs}] = [\text{Comparable Designs}] \cup d )</td>
<td></td>
</tr>
<tr>
<td>( \forall d \in \text{[Comparable Designs]} )</td>
<td></td>
</tr>
<tr>
<td>( \exists p = \text{pr} \in \text{[Patterns Retrieved</td>
<td>\textit{learned}</td>
</tr>
<tr>
<td>( \text{Then Suggest} \text{ pr as a candidate pattern for Retrieval} )</td>
<td></td>
</tr>
</tbody>
</table>

| Pattern Indifference | \( \forall \text{pr} \in \text{[Patterns Retrieved |\textit{current}|]} \) |
|----------------------|-----------------------------------------------|
| \( \forall \text{pr} \in \text{[Patterns Retrieved |\textit{current}|]} \) |
| \( \text{If} \) \( \forall \text{pr'} \in \text{[Patterns Retrieved |\textit{current}|]} \) \( \text{pr} \neq \text{pr'} \) \( \exists p = \text{pr'} \in \text{[Patterns]} \) |
| \( \forall d \in \text{[Prior Designs]} \) |
| \( \exists \text{o instantiation} \in \text{pr = o instantiation} \in \text{pi} \) |
| \( \text{Then Suggest} \text{d'.Instantiation in pi as d .Instantiation in pr} \) |
| \( \text{If} \) designer concurs, |
| \( \text{Instantiate d = pr as d .Instantiation in pi} \) |

| Partial Instantiation Affinity | \( \forall \text{pr} \in \text{[Patterns Retrieved |\textit{current}|]} \) |
|-------------------------------|-----------------------------------------------|
| \( \forall \text{pi} \in \text{[Pattern Instantiations |\textit{learned}|]} \) |
| \( \exists o \in p \) |
| \( \text{Then Suggest} \text{d'.Instantiation in pi as o .Instantiation in pr} \) |
| \( \text{If} \) designer concurs, |
| \( \text{Instantiate d = pr as o .Instantiation in pi} \) |

| Inverse | \( \forall \text{pr} \in \text{[Patterns Retrieved |\textit{current}|]} \) |
|---------|-----------------------------------------------|
| \( \exists o \in p \) |
| \( \text{Then Suggest} \text{d'.Instantiation in pi and (d .Instantiation in pi).Inverse} \) |
| \( \text{as d .Instantiation in pr} \) |
### Affinity-based Learning Mechanisms

#### 4.3.2.2 Synthesis Mechanisms

The learning mechanisms for synthesis allow retrieved and/or instantiated patterns to be synthesized in the problem domain. It, thus, applies to step 4 in figure 4.3.

- **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.

- **The Synthesis by Keyword** mechanism is based on the fact that some objects share a number of instantiations in prior designs. For example, 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.

| Table 4.5 Affinity-based Learning Mechanisms |

| Table 4.6 Learning Mechanisms for Augmenting Synthesis |
4.3.2.3 Spreading Mechanisms

The spreading mechanisms allow iteration after an initial pass is made through all of the design steps (figure 4.4). First, to facilitate the spreading mechanisms, a spreading by object technique is used. Given a keyword, the object with a matching instantiation and the corresponding pattern would have been retrieved in phase 1. At this stage, other patterns that are connected to the retrieved pattern are selected, thus spreading the design. For example, the keyword ‘sale’ would have retrieved the pattern ‘transaction-participant.’ Now, the spreading will consider patterns in which ‘participant’ is involved, e.g. ‘participant-actor.’ This simple spreading by object technique enables implementation and use of the learning mechanisms.

- **Spreading by Keyword** utilizes information about instantiations shared by different objects (similar to Synthesis by Keyword) to identify new objects, and consequently, new patterns. For example, ‘Container-Content,’ may have been retrieved for the current design based on the keyword ‘warehouse’ (object ‘Container’); the object ‘Content’ may have been instantiated as ‘loading dock’ via another learning mechanisms (such as Partial Instantiation Affinity). The new keyword ‘loading dock’ may then be used to identify new objects, such as ‘Associate,’ and retrieval of additional patterns such as ‘Associate-Other Associate.’

- **Spreading via Recursion** exploits the fact that some patterns are used recursively in the design process. These are often aggregate patterns such as ‘Assembly-Part’ or ‘Container-Content.’ The structure of an aggregate pattern is composed of two parts: an aggregate object and part object. When the part object can be further decomposed,
the pattern applies recursively; that is, the part object is used as a pivot to recursively apply the same pattern. This mechanism uses such recursion along with prior instantiations to suggest the consequent instantiation. For instance, if ‘Assembly-Part’ is retrieved and instantiated as ‘aircraft-engine’, based on prior instantiations, the decomposition ‘engine-engine part’ is suggested as an instantiation for the recursive application of the ‘Assembly-Part’ pattern.

### 14. Spreading by Keyword

| ∀ pr ∈ [Patterns Retrieved | amerge] | pr = p ∈ [Patterns] | ∃ q, d ∈ p | q d are instantiated |
| ∀ d ∈ p | d ≠ q | d not identified by w ∈ [Words | amerge] |
| ∀ d' ∈ [Objects] | d' ≠ d |
| If ∃ d', Instantiation | learned = d, Instantiation | amerge |
| Then Suggest d' as a candidate object for Identification |
| If designer concurs |
| Then Suggest d', Instantiation | learned |
| s d, Instantiation | amerge |
| If designer concurs |
| Then ∀ p ∈ [Patterns] | d' ∈ p |
| Suggest p as a pattern for Retrieval |

| 15. Spreading via Recursion |
| ∀ pr ∈ [Patterns Retrieved | amerge] | pr = p ∈ [Patterns] |
| If pr = p ∈ [Patterns Suggested for Recursion] | initial knowledge |
| Then Suggest Retrieval of p and its recursive application |
| If designer accepts |
| If ∀ o ∈ p | o is instantiated |
| If ∃ p ∈ [Prior Instantiations of pr | learned] | Non-Pivot-Object Instantiation ∈ Initial pr = Pivot-Object Instantiation ∈ p |
| Then Suggest Non-Pivot-Object Instantiation ∈ p as Non-Pivot-Object Instantiation ∈ Recursive pr |

Table 4.7 Learning Mechanisms for Augmenting Spreading

The firing of the learning mechanisms is controlled by the steps of the naïve approach (figure 4.2) as well as dependencies among the mechanisms and meta-rules that describe the appropriateness of the mechanisms. Table 4.8 summarizes the applicability of the learning mechanisms to the steps of the naïve approach (figure 4.4) and these meta-rules.
4.4 Application of Learning Mechanisms

The learning mechanisms must be implemented to test their feasibility. They cannot be useful, however, without training on prior cases (usage history). Initial testing is also needed for identifying thresholds for firing the learning mechanisms. Finally, to evaluate their performance, an assessment scheme is needed. Figure 4.5 summarizes these steps.

![Figure 4.5 Training the Learning Mechanisms and Preparing for Evaluation](image)

Table 4.8 Applicability of Learning Mechanisms to the Naive Approach

<table>
<thead>
<tr>
<th>Learning Mechanism</th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Iterate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discard Words</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shield</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guide</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Competing Objects</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Competing Words</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct Instantiation</td>
<td>✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reinforcement and Supervised Learning

1. Discard Words
2. Shield
3. Guide
4. Competing Objects
5. Competing Words
6. Direct Instantiation

Relevance-based and Inductive Learning

7. Pattern Affinity
8. Pattern Indifference
9. Partial Instantiation Affinity
10. Inverse
11. Instantiation Affinity
12. Synthesis by Object
13. Synthesis by Keyword
14. Spreading by Keyword
15. Spreading via Recursion

1. Implementation of the Augmented Approach
2. Training the Learning Mechanisms
3. Deciding Thresholds
4. Initial Testing
5. Determining Assessment Scheme

Typically used in sequence
Must precede 9, 10, 11
Typically used in the first iteration
Must precede 11
Typically used after 7
Cannot follow 6
Variation of 9
Typically after 7 and 4
Typically after 11
Iteration

100 cases
31 cases
131 Cases in 4 Domains
131 Cases in 4 Domains
31 cases
54
4.4.1 Implementation

The learning mechanisms were incorporated as an extension of the naïve approach, and implemented in a prototype, Automated Pattern Synthesis and Retrieval Assistant (APSARA), using Java™. The Patterns Library and the Usage History were structured as a relational database. The patterns library was populated with the patterns proposed by Coad et al [1995].

4.4.2 Training the Learning Mechanisms

The learning mechanisms were trained using cases (usage history) collected from students enrolled in a graduate information systems program. The students were instructed to create a short statement of requirements for different software applications. Four industries: retail, health care, construction, and university; and four application domains [Glass and Vessey 1996]: human resources, inventory control, scheduling, and training were selected, and students randomly assigned to an industry and application domain. For example, one student was asked to provide a short statement of requirements for a human resource application for a health care company; another was asked to do the same for a scheduling application for a university. This provided 131 usable cases, of which 100 were selected to build the Usage History by exercising the prototype as only the naïve approach. As each case was entered, the design sessions, including any feedback from the designer, were recorded in the Usage History. A significant fraction of the Patterns Library was exercised by the training cases. On average, a pattern was used 23 times, and approximately 4 new keywords were added for a generic object.
4.4.3 Thresholds for Relevant Suggestions

Trials of the learning mechanisms with new test cases indicated that the training data clearly separated relevant and irrelevant suggestions, within a given application domain. For instance, given an application domain, a few keywords were clearly relevant (applicable to more than 80% of the prior cases in the domain) and many others were clearly irrelevant (applicable, often, to less than 5% of the cases in that domain). The threshold values were, therefore, decided on the basis of minimal thresholds discovered in the initial testing that would result in the inclusion of all relevant suggestions. Because designer interaction is required as part of the approach and it is relatively easy for the designer to discard any spurious suggestions, this inclusive strategy was adopted. Note, however, that the thresholds can be easily changed depending upon the designer preferences. The thresholds used are summarized in Table 4.9.

<table>
<thead>
<tr>
<th>Group</th>
<th>Mechanism</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>Discard</td>
<td>60% IF observed in 60% or more prior cases THEN suggest as candidate</td>
</tr>
<tr>
<td></td>
<td>Shield</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Guide</td>
<td></td>
</tr>
<tr>
<td>Initiating</td>
<td>Competing Objects</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Competing Words</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct Instantiation</td>
<td>75%</td>
</tr>
<tr>
<td>Affinity</td>
<td>Pattern Affinity</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td>Partial Instantiation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pattern Instantiation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pattern Indifference</td>
<td>Any IF (not) observed in any prior case THEN suggest as candidate</td>
</tr>
<tr>
<td></td>
<td>Inverse</td>
<td></td>
</tr>
<tr>
<td>Synthesis</td>
<td>Synthesis by Object</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td>Synthesis by Keyword</td>
<td></td>
</tr>
<tr>
<td>Spreading</td>
<td>Spreading by Keyword</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>Spreading by Recursion</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.9 Thresholds for Learning Mechanisms
4.4.4 Initial Testing of Prototype

The first phase of testing was performed with the 31 cases that remained after building the usage history. The prototype was exercised several times, each invoking different learning mechanisms. For each case, ‘ideal’ models were manually constructed by the researchers to be used as benchmarks. The creation of the ideal models involved three steps, again performed manually. First, classes were identified by examining the requirements statements. Only after this step was completed, was the patterns-base 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 available in the patterns-base. Two of the three researchers co-developed the ideal models, with the third researcher providing independent validation.

The ideal models, constructed in this manner, show results that an expert might achieve, by incorporating analysis patterns in the design, unaided by either the naïve or augmented approaches. Using the ideal model as the benchmark, allows us to examine whether the augmented approach can help bridge the gap between results generated by the naïve approach and those created by the independent experts. The closer we get to the ideal model, the better the contribution of the learning mechanisms. Figure 4.6 shows two representative results from this testing.
Figure 4.6 Comparing Naïve and ‘With Learning’ Solutions against the Ideal Model

The first column shows designs created using the naïve approach, the second represents design generated (using the same requirements) with the augmented approach (with learning), and the third shows the ‘ideal’ model. For easier reading, the figure shows only classes, without attributes or methods. Designs generated with the augmented approach are much closer to the ideal model than those generated by the Naïve approach. They also suggest a meaningful way to measure efficacy of the learning mechanisms, by comparing these results.

4.4.5 Assessment Scheme

An assessment scheme was developed using a comparison of the model developed with the naïve or augmented approach against the ideal model. Type I errors represent errors of omission, such as: missing objects, missing instantiations of objects, and missing relationships among objects, not counting those due to missing objects. Type II errors are errors of commission, such as: the incorrect identification of objects, and incorrect relationships, other than those due to incorrect objects. They may, however, include plausible extensions, that is, objects and relationships that may be
appropriate given the requirements but were not anticipated by the researchers when creating the ideal model. Errors of both types can, therefore, be classified as major, minor or reverse. Major errors seriously affect the quality of a model. These are penalized at 3 points per error. Minor errors have less effect on the quality of a model and are penalized at 1 point per error. Finally, reverse errors (that is, plausible extensions) are rewarded. Because the decision between incorrect and plausible can be subjective, the reward is conservatively set at 1 point per reverse error. Table 4.10 shows the assessment scheme. The numbers in each cell indicate the weight of an error for a combination of error type and element type. Similar measures have been suggested by Moody [1998] and Storey [1993].

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Modeling Element</th>
<th>Object</th>
<th>Instantiation</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type I: Errors of Omission</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing Elements</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-1, if due to missing objects</td>
</tr>
<tr>
<td><strong>Type II: Errors of Commission</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incorrect Elements</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-1, if due to incorrect objects</td>
</tr>
<tr>
<td>Redundant Elements</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1, if due to redundant objects</td>
</tr>
<tr>
<td>Plausible Elements</td>
<td>+1</td>
<td>N.A.</td>
<td>+1</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Table 4.10 Assessment Scheme**

### 4.5 Empirical Evaluation

The feasibility of using analysis patterns with a naïve approach has already been demonstrated [Purao and Storey 1997a]. The objective of augmenting the naïve approach with the learning mechanisms is so that better designs will be obtained. To test this claim, and thus, assess the effectiveness of the learning mechanisms, an experiment was conducted. A new set of 35 subjects, novice designers, were given multiple tasks that
required using either the naïve or augmented approach for conceptual design. Varied application domains and tasks of different sizes were used. The variation across application domains was important to investigate whether the benefits of learning were transferable. We chose two domains: one in which the mechanisms were trained, and the other, a related, but new, domain, in which the learning mechanisms were not trained. Testing for different size levels was considered important to assess the scalability of the approach.

4.5.1 Research Model and Hypotheses

Because we were primarily interested in evaluating the incremental benefits of learning over the naïve approach to patterns reuse, the experiment was designed to compare the two approaches. Thus, the two approaches were considered as two treatments (following Straub et al [1993]). This enabled us to consider the naïve approach as a control and draw conclusions about the incremental contribution of the learning mechanisms. The base research model for the experiment, therefore, is:

\[ \text{Errors in Design} = f (\text{Design Approach, Domain, Task}) \]
Size), as shown in figure 4.7. The design approaches are A1: Naïve, and A2: Augmented. The domains are D1: Trained, D2: Untrained but Related. The task size levels are S1: Small, S2: Medium and S3: Moderate. Three core hypotheses were posited as shown in table 4.11.

| **H1: Improvement** | **H1**: Designs produced with the augmented approach contain fewer errors compared to those produced using the naïve approach.  
Null: $E(A2)=E(A1)$ | **H1A**: At each size, designs produced with augmented approach contain fewer errors than those produced with naïve approach.  
Null: $E(A2,Sn)=E(A1,Sn) \mid n=1,2,3$ | **H1B**: For each domain, designs produced with augmented approach contain fewer errors than those produced with naïve approach.  
Null: $E(A2,Dk)=E(A1,Dk) \mid k=1,2$ |
| --- | --- | --- | --- |
| **H2: Scalability** | **H2**: The reduction in design errors (from naïve to augmented) does not suffer with an increase in task size.  
Null: $E(A2,S1)-E(A1,S1)=E(A2,S2)-E(A1,S2)=E(A2,S3)-E(A1,S3)$ | **H3: Transferability** | **H3**: The reduction in design errors (from naïve to augmented) is better for the trained domain than for the untrained, related domain.  
Null: $E(A2,D1)-E(A1,D1)=E(A2,D2)-E(A1,D2)$ |

**Table 4.11 Hypotheses Posited**

*Independent Variables.* The research model suggested three independent variables. The first is the variable of interest i.e. the approach, naïve versus augmented. In addition, there are two other independent variables. First, multiple application domains were used: human resource management (D1) and warehouse management (D2). D1 represents a traditional organizational function generally responsible for keeping records such as employee skills, benefits and other records. The learning mechanisms were trained in this domain. 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 related domain (see section
4.2). Second, three levels of tasks, small, medium, and moderate-size 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]) was used as a determinant of system size.

Dependent Variables The quality of designs produced was measured using the assessment scheme (see table 4.10). A set of ideal models, against which the dependent variables (errors) could be measured, was developed following the procedure of section 4.4.4 [Models 2001].

<table>
<thead>
<tr>
<th>Size</th>
<th>Human Resource Management</th>
<th>Warehouse Management</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Classes</td>
<td>Classes plus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Relationships</td>
</tr>
<tr>
<td>Small</td>
<td>Objects: 5</td>
<td>Objects + Relationships: 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>Objects: 9</td>
<td>Objects + Relationships: 18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td>Objects: 13</td>
<td>Objects + Relationships: 26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.12 Comparative Sizes ([Lorenz and Kidd 1994, de Champeaux 1997])

The quality of the generated model was scored as: errors in the generated model/size of the ideal model. It was expected that the naïve model would generate identical results for a given <approach-domain-size> combination unless the subjects failed to follow the steps and/or misunderstood the task. At the other extreme, the ideal models would provide a single benchmark (for a given <approach-domain-size> combination) – representing unaided experts utilizing the patterns. The augmented approach, with the learning mechanisms acting as intelligent assistants to the developers, would provide varied results. The dependent variable, errors in the model generated by the augmented approach could, therefore, be compared against errors in the model
generated by the naïve approach, in order to evaluate the incremental contribution of the former (see figure 4.8).

![Figure 4.8 Expected Behavior of Dependent Variables]

### 4.5.2 Experimental Design and Procedures

A laboratory experiment using a 2 (approach) by 2 (domain) by 3 (task size) was conducted. The three factors resulted in 12 treatment combinations. To allow for multiple measurement possibilities for each subject with different \(\text{<approach–domain–size>}\) combinations, a repeated measures design was chosen [Kerlinger 1986]. Besides requiring fewer experimental subjects, the design provides a control on the differences among various subjects, that is, variability due to differences between subjects can be eliminated from the experimental error [Ramarapu et al 1997].

**Subjects** The subjects were recruited from graduate students enrolled in a large masters program in information systems. The study was briefly described and students willing to participate scheduled to perform the tasks. 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.
**Procedure**  The experiment was administered to individual subjects randomizing the \(<\text{approach-domain-size}>\) combination. The randomization provided direct control for order; the random assignment of subjects to order and treatment ensured that the expected differences between grouping (order and treatment levels) was zero at the time of randomization [Cohen and Cohen 1975]. Instructions were given to the subjects as part of the interface of the prototype [APSARA-Augmented 2000, APSARA-Naïve 2000]. The schedules were staggered to ensure that peer pressure played a minimal role in performance. One of the researchers was present to answer any questions. Both the naïve and the augmented versions of the prototype had an identical user interface, except the additional functionality in the augmented approach and the consequent interactions. The subjects completed four tasks during the sessions, which lasted less than two hours. The results were recorded by the prototype.

### 4.5.3 Results

35 subjects completed 4 tasks to generate a total of 140 design instances, of which 13 were the properly completed, yielding 127 usable instances. Table 4.13 shows the means, number of observations and standard deviations for the dependent variable, design errors (E) by approach (A), task size (S) and domain (D) combination. A lower score indicates better design.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Task Size</th>
<th>Naïve Approach (A1)</th>
<th>Augmented Approach (A2)</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>E:Mean</td>
<td>E:SD</td>
<td>E:Mean</td>
</tr>
<tr>
<td>D1: Human Resource Management</td>
<td>Small (S1)</td>
<td>.8148</td>
<td>0</td>
<td>.4444</td>
</tr>
<tr>
<td>Medium (S2)</td>
<td>.9630</td>
<td>0</td>
<td>.7469</td>
<td>.1126</td>
</tr>
<tr>
<td>Moderate (S3)</td>
<td>1.3718</td>
<td>0</td>
<td>.5531</td>
<td>.2307</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1.0586</td>
<td>N.A.</td>
<td>.6186</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(n=44)</td>
<td></td>
<td>(n=20)</td>
</tr>
<tr>
<td>D2: Warehouse Management</td>
<td>Small (S1)</td>
<td>.1429</td>
<td>0</td>
<td>.1810</td>
</tr>
<tr>
<td>Medium (S2)</td>
<td>.8788</td>
<td>0</td>
<td>.3939</td>
<td>.2736</td>
</tr>
<tr>
<td>Moderate (S3)</td>
<td>.7059</td>
<td>0</td>
<td>.4608</td>
<td>.1258</td>
</tr>
</tbody>
</table>
Table 4.13  Design Error Means for the two Domains

Not surprisingly, the results for the naïve approach, which required following a series of steps, show no variation in the designs produced (SD=0.0). The results from the naïve approach in each case were, thus, used as the baseline (control) against which the results for the augmented approach were compared (similar to [Straub et al 1993]). The augmented approach does show variation in results (see figure 4.9) for different domains or size levels. An ANOVA was conducted to investigate the effects of design factors on the total error rate. Table 4.14 shows the main effects, and the interactions among factors.

<table>
<thead>
<tr>
<th>Df</th>
<th>F-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>195.670***</td>
</tr>
<tr>
<td>1</td>
<td>202.045***</td>
</tr>
<tr>
<td>2</td>
<td>87.212***</td>
</tr>
<tr>
<td>1</td>
<td>22.638***</td>
</tr>
<tr>
<td>2</td>
<td>16.390***</td>
</tr>
<tr>
<td>2</td>
<td>8.972***</td>
</tr>
<tr>
<td>2</td>
<td>29.512***</td>
</tr>
<tr>
<td>115</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>90.863***</td>
</tr>
</tbody>
</table>

* p<.05, ** p<.01, *** p<.001

Table 4.14  ANOVA - Effects of Design Factors on Total Error Rates

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.
Hypothesis H1 was, thus, supported \( (t=6.263, \text{significant at } p<.001) \) (table 4.15). We also expected that designs produced by the augmented approach would continue to be better than those produced with 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 4.15). Hypothesis H1A was, thus, supported at each level. \( (t=2.456, \text{significant at } p<.05 \text{ in } S1; \ t=6.732, \text{significant at } p<.001 \text{ in } S2; \ t=7.542, \text{significant at } p<.001 \text{ in } S3) \) (table 4.15).

<table>
<thead>
<tr>
<th>Approach</th>
<th>Average</th>
<th>Size</th>
<th>Average</th>
<th>Size</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve</td>
<td>0.8359</td>
<td>Small (S1)</td>
<td>0.4788</td>
<td>Medium (S2)</td>
<td>0.5321</td>
</tr>
<tr>
<td>Augmented</td>
<td>0.4497</td>
<td>Moderate (S3)</td>
<td>1.1221</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ t=6.263^{**} \]

\[ t=2.456^{*} \]

\[ t=6.732^{***} \]

\[ t=7.542^{***} \]

* \( p<.05 \), ** \( p<.01 \), *** \( p<.001 \)

Table 4.15 Total Error Rates for Different Sizes under the Two Approaches

The augmented approach, thus, exhibited a propensity for producing designs with fewer errors regardless of size. On the other hand, the naïve approach produced designs with increasing errors (figure 4.9). The augmented approach, therefore, performed better than the naïve approach at each size. This hypothesis, suggesting scalability, was further investigated separately (H2).

The augmented approach also fared well for both trained \((D1: \text{Human Resource Management})\) and the untrained but related domain \((D2: \text{Warehouse Management})\).
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 as well (D2), the augmented approach produced designs with a lower average error (0.3441) than the naïve approach (0.5200). Hypothesis H1B was, therefore, supported (t=8.796, significant at p<.001 in D1; t=4.203, significant at p<.001 in D2) (table 4.16).

<table>
<thead>
<tr>
<th>Approach</th>
<th>Average</th>
<th>Domain</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Trained: Human Resource</td>
<td>Related: Warehouse Management</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Management (D1)</td>
<td>(D2)</td>
</tr>
<tr>
<td>Naïve</td>
<td>0.8359</td>
<td>1.0586</td>
<td>0.5200</td>
</tr>
<tr>
<td>Augmented</td>
<td>0.4497</td>
<td>0.6186</td>
<td>0.3441</td>
</tr>
<tr>
<td>t=6.263***</td>
<td></td>
<td>t=8.796***</td>
<td>t=4.203***</td>
</tr>
</tbody>
</table>

* p<.05, ** p<.01, *** p<.001

Table 4.16 Total Error Rates for Different Domains under the Two Approaches

Hypotheses H1, H1A and H1B were, thus, uniformly supported in the preferred direction. The tests showed that the proposed learning mechanisms significantly reduced errors caused by naïve automation of analysis patterns reuse. The results were consistent across three size levels and for two domains, one in which the learning mechanisms were trained, and another which was related but in which the learning mechanisms were not directly trained.

Test of Hypothesis 2: Scalability

We expected that improvements brought about by the augmented approach would not reduce with increasing size. Table 4.17 shows a comparison of the improvements at different size levels. The augmented approach performs better across the board, at each size, indicated by a positive value at each level. The improvements were computed as pairwise differences, that is, by considering an equal number of cases for each size. For example, the improvement for size S1 was computed by considering each case of size S1
for which the augmented approach was applied, and comparing it against the results of
the naïve approach for that size (0.8148 for D1 and 0.1429 for D2, as shown in table
4.13). The improvement, therefore, is more conservative and more accurate (0.786 as
shown in table 4.17) instead of a simple difference between averages (0.4788 less 0.2562
i.e. 0.2226 from table 4.15). An F-test and a following post hoc test revealed two
significantly different groups (S1 and [S2 and S3]) at p<.01 for the improvements across
size levels (table 4.17).

<table>
<thead>
<tr>
<th>Measure</th>
<th>Size</th>
<th>Small (S1)</th>
<th>Medium (S2)</th>
<th>Moderate (S3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improvement due to Augmented Approach</td>
<td></td>
<td>0.0786</td>
<td>0.3797</td>
<td>0.5128</td>
</tr>
<tr>
<td>Post Hoc Test (p&lt;.05)</td>
<td></td>
<td></td>
<td>Not significant</td>
<td></td>
</tr>
</tbody>
</table>

* p<.05, ** p<.01, *** p<.001

Table 4.17 Comparing Improvements across Size Levels

Even with the conservative computation of differences, improvement due to the
augmented approach shows a significant increase with a move from the small size (S1) to
the medium size (S2). An increase is also seen from medium size (S2) to moderate size
(S3), although it is not significant. The set of hypotheses H2 were, therefore, supported in
the preferred direction. To confirm this, since the numbers appeared to be different
(0.3797 for S2 and 0.5128 for S3), a t-test was performed (t=1.360, p-value .182, not
significant at .05). The tests, thus, indicate that improvement in design quality,
attributable to the augmented approach, does not suffer as the task size increases, i.e. the
augmented approach is scalable within the confines of the laboratory experiment.

To further understand the improvements, we computed the relative improvement
due to the augmented approach. The relative error rates were computed using the
pairwise improvement computation explained above. For example, for the small size (S1), the average relative improvement was 10.61%. Similar computations were performed for medium and moderate size (table 4.18).

<table>
<thead>
<tr>
<th>Approach</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small (S1)</td>
</tr>
<tr>
<td>Relative Improvement due to Augmented Approach</td>
<td>10.61%</td>
</tr>
<tr>
<td>Post Hoc Test (p&lt;.05)</td>
<td>Not significant</td>
</tr>
</tbody>
</table>

* p<.05, ** p<.01, *** p<.001

Table 4.18 Comparing Relative Improvements across Size Levels

The post hoc test (F=7.434, significant at p<.01) revealed that improvement due to the augmented approach did not suffer with an increase in task size. It also confirmed that the relative improvement was significantly different between S1 and [S2 and S3], but not significantly different between S2 and S3, further supporting hypothesis H2 in the preferred direction.

Test of Hypothesis 3: Transferability across Domains

We expected that the improvement brought about by the augmented approach over the naïve approach in the trained domain (D1: Human Resource Management) would be better than that for the domain (D2: Warehouse Management), which did not receive direct training but was related to a domain that did receive training (inventory control). A comparison of improvements brought about by the augmented approach in the two domains reveals significant differences between the two domains as shown in table 4.19. The improvements (0.4579) due to the augmented approach in the trained domain
(D1) are significantly higher than those (0.2615) for the untrained domain (D2) (t=2.168, significant at p<.05) (table 4.19).

<table>
<thead>
<tr>
<th>Measure</th>
<th>Domain</th>
<th>Trained: Human Resource Management (D1)</th>
<th>Related: Warehouse Management (D2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improvement due to Augmented Approach</td>
<td>t=2.168*</td>
<td>0.4579</td>
<td>0.2615</td>
</tr>
</tbody>
</table>

* p<.05, ** p<.01, *** p<.001

Table 4.19 Comparing Improvements under Augmented Approach across Domains

Hypothesis H3 was, therefore, supported in the preferred direction. To understand these improvements, relative improvements were computed using pairwise comparisons similar to table 4.18 for size levels. The results, shown in table 4.20, confirm the above interpretations.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Domain</th>
<th>Trained: Human Resource Management (D1)</th>
<th>Related: Warehouse Management (D2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Improvement due to Augmented Approach</td>
<td>t=1.004</td>
<td>40.08%</td>
<td>31.78%</td>
</tr>
</tbody>
</table>

* p<.05, ** p<.01, *** p<.001

Table 4.20 Comparing Relative Improvements across Domains

The mean improvement for the directly trained domain is higher. The difference between the two domains is, however, not significant (t=1.004, not significant at p<.05). The learning mechanisms, thus, improved the quality of designs produced not only in a domain in which they were directly trained, but also in a domain that was related to another trained domain. The test of relative improvement, therefore, supported hypothesis H3, partially in the preferred direction. The hypotheses posited, along with preferred directions compared to the null hypotheses, and the results are summarized in table 4.21.
### Table 4.21 Summary of Findings

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Preferred Direction</th>
<th>Finding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improvement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H1</td>
<td>$E(A_2)$ $E(A_1)$</td>
<td>Supported</td>
</tr>
<tr>
<td>H1A</td>
<td>$E(A_2, S_n)$ $E(A_1, S_n)</td>
<td>n=1,2,3$</td>
</tr>
<tr>
<td>H1B</td>
<td>$E(A_2, D_k)$ $E(A_1, D_k)</td>
<td>k=1,2$</td>
</tr>
<tr>
<td>Scalability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td>$E(A_2, S_1)$ $E(A_1, S_1)$</td>
<td>Supported</td>
</tr>
<tr>
<td>Transferability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H3</td>
<td>$E(A_2, D_1)$ $E(A_1, D_1)$</td>
<td>Supported</td>
</tr>
</tbody>
</table>

#### 4.5.4 Further Analysis

The data was further analyzed at finer granularity levels as shown in Table 4.22.

The fine grain analysis lends further support to the hypotheses above.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Small (S1)</th>
<th>Medium (S2)</th>
<th>Moderate (S3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naïve</td>
<td>0.8148</td>
<td>0.1429</td>
<td>0.9630</td>
</tr>
<tr>
<td></td>
<td>0.3939</td>
<td>0.4608</td>
<td>1.3718</td>
</tr>
<tr>
<td></td>
<td>0.7059</td>
<td></td>
<td>0.7059</td>
</tr>
<tr>
<td>Augmented</td>
<td>0.4444</td>
<td>0.1810</td>
<td>0.7469</td>
</tr>
<tr>
<td></td>
<td>0.1429</td>
<td>0.3939</td>
<td>1.3718</td>
</tr>
<tr>
<td></td>
<td>0.5531</td>
<td>0.4608</td>
<td>1.3718</td>
</tr>
<tr>
<td></td>
<td>0.3939</td>
<td>0.4608</td>
<td>1.3718</td>
</tr>
<tr>
<td></td>
<td>0.7059</td>
<td></td>
<td>0.7059</td>
</tr>
<tr>
<td>Improvement</td>
<td>0.3704 (0.0381)</td>
<td>0.2161 (0.2688)</td>
<td>0.5736 (0.2688)</td>
</tr>
<tr>
<td></td>
<td>0.4849</td>
<td>0.8187</td>
<td>0.2451</td>
</tr>
</tbody>
</table>

| Difference (D1-D2) | 0.4085 | (0.2688) | 0.5736 |
| Difference (D1-D2) | t=3.472** | t=3.269** | t=6.093*** |

* p<.05, ** p<.01, *** p<.001

Key; D1 = Human Resource Management (trained), D2 = Warehouse Control (untrained)

**Table 4.22 Improvements under Augmented Approach for Domains at Size Levels**

The table shows *improvement* in each domain (D1 and D2) for each size (S1, S2 and S3). The improvements are compared to compute the *difference* in improvement between D1 and D2 for each size. There are two anomalous cells in the table, indicated by (negative) values. The first is cell [D2, S1], which shows a negative improvement compared to the naïve approach. An analysis of the logs suggests that a possible reason for this was the designers’ propensity to use the ‘discard’ mechanism (available early in the implemented prototype, see [APSARA-Augmented 2000]) resulting in designs worse than those produced by the naïve approach. The second is cell [(D1-D2),S2], which shows that the improvement in the untrained domain (D2) was better than that for the trained domain (D1). To investigate this further, the results were plotted. Figure 4.10
shows these results separately for the two domains, placed next to each other, using the same scale.

The figures, in turn, show two interesting blips. The first occurs for D1, A1 (naïve), as the error rate jumps from S1 to S2, and then falls for S3. The second occurs for D2, A2 (augmented), as the error rate jumps from S1 to S2 and then falls for S3. Together, these two blips contribute to narrowing the improvement for [D1,S2] and widening the improvement for [D2,S2], resulting in the negative value indicated for the cell [(D1-D2),S2] in table 4.22. Additional tests were conducted and process logs for cases in these cells were further analyzed to locate reasons for this anomaly.

The blip for D1 (figure 4.10A) could not be 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<.05). A replication may be necessary to understand this blip further. The other blip, for D2 (figure 4.10B) was 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 statements manifested in a

Figure 4.10 Errors for the Trained Domain (D1) compared to the Related Domain (D2)
higher proportion of missing objects for S2 (-16 on average) than for S3 (-11 on average). This resulted in a higher average error rate for S2 than that for S3. One specific reason for this was determined as the missing words (e.g. dock) that were required for S2 but not for S3.

### 4.6 Conclusion

We have presented an augmented approach that acts as an intelligent assistant to the designer to facilitate reuse-based conceptual design with analysis patterns. It includes learning mechanisms in two categories: 1) reinforced and supervised learning, and 2) relevance-based and inductive learning. The approach was implemented in a prototype and the learning mechanisms were trained. A rigorous laboratory experiment was performed to assess the value of the augmented approach, over a naïve approach. The findings are very encouraging, indicating that the learning mechanisms result in significant improvements in design quality. They also suggest that the learning mechanisms are scalable within the confines of a research prototype, and may be useful in new, related, but untrained domains.

It is possible to argue that experienced, unaided designers may produce designs that are as good or better. We would anticipate that experienced designers have an intuitive ‘expert system’ that may, in fact, be analogous to the proposed learning mechanisms or knowledge encoded in the analysis patterns. This research may not significantly benefit to this group of designers. However, if inexperienced designers using the approach can produce designs that come close to the quality of designs produced by an experienced designer, then, our proposals would benefit organizations in
the sense that they would not need as many experienced designers. Further studies are
needed to empirically test this possibility. At least two other possible uses of the
approach can be identified. First, it may be used as a training aid for designers to become
familiar with the use of different patterns in different situations. Second, it can be used to
generate alternative designs quickly. These designs may themselves become reusable
artifacts in that domain.

The approach and findings reported in the paper suggest a few avenues for further
investigations and extensions. First, a different approach to evaluating the contribution of
individual learning mechanisms could be undertaken using a technique such as process
tracing. Second, the rules embedded in the structure of a pattern (e.g. cardinality) may
themselves be modified during reuse. Such pattern modifications could further augment
the reuse process. Third, knowledge learned through the augmented approach suggests
that it may be possible to generate larger-grain, domain-specific reusable artifacts by
leveraging this knowledge, improving the design products further. These remain on our
future research agenda.
CHAPTER 5
DESIGN FRAGMENTS: INDUCTIVELY
GENERATED REUSABLE ARTIFACTS

5.1 Introduction

The development of good designs for information systems starts with an understanding of the users requirements. Most designs are still developed “from scratch,” even though a similar design might have been created previously within the same application domain. The employment of reusable artifacts takes advantage of prior design work, with several reusable artifacts having been proposed [Szyperski 1998, Fowler 1997, Coad et al 1995, Gamma et al 1995, Pree 1994, Prieto-Diaz 1993]. Many of these are utilities (e.g. sort routine), large components packages (e.g. shopping cart subsystem), or interface elements (e.g. buttons).

For more critical aspects of systems development, such as conceptual design, there are, unfortunately, few reusable artifacts. Analysis patterns [Coad et al. 1995, Fowler 1997] represent small chunks of domain-independent, recurring solutions. Domain models [Prieto-Diaz 1993] serve as templates for conceptual designs within an application domain. The usefulness and reusability of both these artifacts, however, is limited because analysis patterns have a high level of abstraction, and domain models, a coarse granularity. Furthermore, the creation of these reusable artifacts is an inefficient, manual approach that relies heavily upon domain experts and experienced designers [Prieto-Diaz 1990, Kang et al 1990].

One approach to addressing the problem of creating reusable artifacts is to take advantage of available, object-oriented designs to automatically generate reusable artifacts. First, object-orientation lends itself naturally to domain analysis and widely
reusable components [Kang et al 1990]. Second, the synthetic nature of object-oriented designs may make it relatively easy to decompose object-oriented structures into parts that can be combined in different ways [Graham 2001, Szyperski 1998]. Third, the problem-orientation of object-oriented models [Hoydalsvik and Sindre 1993] makes the components easily communicable to the targets for reuse. Fourth, use of object-orientation creates reusable artifacts for conceptual design, where the potential for reuse is the highest [DOD 1996]. Fifth, the resulting reusable artifacts should be better than the available candidates because, for example, they are less abstract and, therefore, more easily reusable than analysis patterns. They are also more granular, and hence, more useful than domain models.

_The objective of this research, therefore, is to create an approach to inductively generating a repository of a new, demonstrably better, reusable artifact from available object-oriented designs._ The research is significant for two reasons. First, it proposes a new reusable artifact that is superior to existing ones. Second, the research has practical significance because it suggests an alternative to the current effort-intensive practice of building reusable artifacts.

This paper is divided into eight sections. Section 2 reviews prior research on approaches to creating reusable artifacts. Section 3 defines design fragments. Our approach to generating a design fragment repository is presented in section 4. Section 5 illustrates the application of results obtained for design with reuse. The feasibility of the approach is demonstrated in section 6. Larger scale applications that are implemented using a prototype system are discussed in section 7. Finally, section 8 summarizes and concludes the paper.
5.2 Prior Research

5.2.1 Reusable Artifacts

A number of reusable artifacts have been proposed, including templates [Purba 1999], components [Szyperski 1998], frameworks [Johnson and Foote 1988, Fayad et al 1997], analysis patterns [Coad et al 1995, Fowler 1997], and design patterns [Gamma et al 1995, Pree 1994]. These can be classified along two dimensions: granularity and abstraction as shown in figure 5.1.

Granularity is the size of the artifact relative to the application in which it is reused. A fine granularity artifact (e.g. class) covers a small portion of the application; a coarser granularity covers a much larger one. Abstraction refers to the variety of situations in which the artifact may be applicable. An artifact with a high level of abstraction may be useful in multiple domains. Examples are analysis patterns for conceptual design or sort subroutines for detailed design.

Repositories have been created for some types of reusable artifacts (e.g. analysis patterns, design patterns) [Coad et al 1995, Fowler 1997, Pree 1994, Gamma et al 1995]. Templates, frameworks and other code components have also been created [Sun 2000]. Prescriptive approaches for domain engineering [Sodhi and Sodhi 1999, Prieto-Diaz 1990, Lubars 1991, Gomaa 1995] require domain engineers to have a deep knowledge of the domain, consider the boundaries of what objects to include, and identify to what degree they should be abstracted [Lubars, 1991]. Reuse of code components also requires a great deal of effort to manually classify, search, and retrieve them [Mili et al
Thus, a major problem with current approaches to reuse of artifacts is their reliance on human experts, making them practically inefficient.

5.2.2 Evaluating Reusable Artifacts

The value of a reusable artifact lies in how effectively it facilitates reuse-based design. Mili et al. [1995] suggest usability and (re) usefulness are important criteria for judging the effectiveness of a reusable artifact. Usability means the artifact is easy to understand and apply for new systems development; (re)usefulness means the artifact addresses a common need or provides a commonly requested service.

Usability can be interpreted in terms of generic reuse tasks. The first, retrieval, involves search and adaptation. The second, assembly, includes integration with other parts of the design solution, verification of the compatibility of the retrieved artifacts with their environment, and synthesis to create an application design. The usability of an artifact can, therefore, be assessed in terms of ease of retrieval and assembly [Maiden & Sutcliffe 1993] [Purao and Storey 2000].

(Re) usefulness of a reusable artifact can be understood in terms of its granularity and abstractness. Fine-grain artifacts, such as an array of generic objects, can be highly useful, because they are sufficiently generic. Analysis patterns [Coad et al 1995], which are somewhat larger-grain, are also highly useful because they are domain-independent. Individual classes are too fine-grained [Mili et al 1995]. On the other hand, a larger-grain reusable artifact, such as a domain model, may be less useful if it is inflexible. The (re)usefulness of an artifact is balanced by its usability because the generality and applicability of the artifact (its usefulness) may involve abstracting the specifics and
breaking it into component parts that the designer must put back and assemble.

Providing support for the creation and reuse of design artifacts is also an important consideration. Given the effort-intensive nature of these activities, this criterion may be referred to as efficiency with two sub-properties. The first is the effort-intensiveness of creating the artifact. The second is the effort-intensiveness for reusing the artifact.

Table 5.1 summarizes these evaluation criteria and applies them to five best-known artifacts. Any new reusable artifact should, obviously, score better on a significant number of these criteria.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of Retrieval</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Ease of Assembly</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>N.A.</td>
</tr>
<tr>
<td>(Re)usefulness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Granularity</td>
<td>Fine</td>
<td>Medium</td>
<td>Medium</td>
<td>Coarse</td>
<td>Coarse</td>
</tr>
<tr>
<td>Abstractness</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation Effort</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Reuse Effort</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 5.1 Evaluation of Existing Reusable Artifacts

5.3 Design Fragments

A design fragment, is defined as ‘a proper subset of a design that includes objects and relationships’. A design fragment is, thus, domain-specific. The granularity of a design fragment is larger than a single object and smaller than a complete domain model. It captures a combination of objects and relationships, from the numerous possible combinations that may exist in different designs within a domain. Figure 5.2 shows a design fragment using UML modeling notations [Rational 2000]. The four objects and
relationships among them represent a subset of the personnel domain. The relationships capture cardinality and other constraints. The objects contain relevant attributes and methods (not shown).

A design fragment may be combined with other design fragments in different ways to construct a complete design. For example, the design fragment in figure 5.2 may be combined with design fragments for benefits, bonuses, performance or appraisal. A design fragment may also cover multiple subsets of the domain. Thus, a design fragment may consist of objects, relationships, or even other design fragments, contributing to varying levels of granularity. Figure 5.3 shows the meta-model of design fragments, which demonstrates this recursive containment relationship.

Design fragments exhibit a high degree of usability and re-usefulness (table 5.1). Usability is gained from ease of retrieval as a result of the high problem-solution space match that occurs because the artifact is domain specific and uses an object orientation. It also has ease of assembly because of the recursive meta-model of design fragments. The intermediate level of abstraction and medium granularity contribute to high re-usefulness within a domain. Compared to other reusable artifacts available during conceptual design (analysis patterns and domain models), design fragments provide a more affective
alternative (see figure 5.4). To address the third evaluation criterion, efficiency, we devise an approach that supports the construction and reuse of design fragments.

5.3.1 Architecture

This section proposes an architecture for constructing and reusing design fragments. It supports inductive generation of design fragments by exploiting requirement statements and the composition of designs. The design fragments are indexed and stored in a repository. As new requirement statements are encountered, the repository is used to generate designs, which, in turn, are added to the process to grow the repository. Figure 5.5 provides an overview of this approach.

Inputs

The inputs to the building of design fragments consist of requirement statements and prior designs. A systematic approach to generating designs from requirements analysis is, therefore, needed.

We use an automated approach to generating conceptual designs from requirement statements by synthesizing analysis patterns [Purao and Storey 1997a,
1997b]. The automated approach parses simple natural language assertions such as: “a system to track sales at different stores” to identify keywords such as ‘sale’ and ‘store.’ Based upon these, objects and patterns are retrieved, such as object ‘Transaction’ (for sale) and the pattern Place – Transaction. The patterns are then instantiated, using the keywords, for example, 'store – sale' for the pattern Place – Transaction, and synthesized to create the final design. Figure 5.6 summarizes this approach. These synthesized designs along with the requirement statements constitute the inputs.

Building

Design fragments are built using clustering mechanisms. Clustering classifies a set of elements into subsets that share common features or are similar based upon an association measure. A cluster, therefore, is a collection of elements whose intra-cluster similarity is high and inter-cluster similarity is low. Conceptual clustering exploits semantic and syntactic knowledge about the elements. For example, in figure 5.7, points a and y are grouped into one cluster based upon numerical methods. However, with knowledge of...
the circle, it is more appropriate to cluster \( x \) and \( y \) into group A, and \( a \) and \( b \) into group B. In conceptual clustering, it is also possible to construct any type of classification from the given data, based on the focus on the problem [Michalski and Stepp 1983].

The requirement statements are used to cluster designs that address similar contexts and are called *keyword clusters*. Next, the composition of designs is exploited to discover similar structures and semantics to create *design fragments*. Figure 5.8 shows the inputs and outputs generated by the clustering process.

![Figure 5.8 Data model capturing the semantics of clustering process](image)

**Reusing**

The keyword clusters are used to identify relevant design fragments as new requirement statements are encountered. The requirement statements are parsed to identify significant words, which suggest one or more keyword clusters, which, in turn, are used to identify design fragments. The retrieval process can be automated using parsing and matching techniques. Finally, the designer is prompted to assemble the design from some or all of the design fragments retrieved. Table 5.2 summarizes the input, output, and operation of each step.
### Table 5.2 Reusing design fragments

<table>
<thead>
<tr>
<th>Step 1 – Retrieval</th>
<th>Step 2 – Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Requirement statements</td>
</tr>
<tr>
<td>Operation</td>
<td>Retrieve design fragments (automated)</td>
</tr>
<tr>
<td>Output</td>
<td>Design fragments</td>
</tr>
</tbody>
</table>

#### 5.4 Building the Design Fragments

This section presents the methodology for building design fragments. The inputs are: 1) requirement statements in the form of natural language assertions, and 2) prior designs in the domain assembled from instantiated analysis patterns. The process has two phases: keyword clustering and design clustering.
5.4.1 Keyword Clustering

The objective of this phase is to identify and cluster keywords that are similar across multiple requirement statements in order to capture the context. At the end of this phase, each design is mapped against one or more keyword clusters. The multi-step keyword clustering process is described next with an example.

**Step 1: Identify keywords.**

Each requirement statement \(s_i\) is parsed and the stopwords [Stopwords 2001] (e.g., “the, “a,” etc.) removed. The remaining significant words \(\{w_{ij}\}\) are investigated to identify those that occur frequently across requirement statements. These represent keywords in that domain. A designer-defined threshold, called an *occurrence frequency threshold*, determines the relative frequency at which a significant word will be promoted to the status of a ‘keyword’. The algorithm for this process is shown next with a discussion of the selection of appropriate threshold values provided next.

\[
\forall w \in W \text{ DO} \\
\quad \forall s \in S \text{ DO} \\
\quad \quad \text{IF } w \in \{w_j\} \\
\quad \quad \quad \text{frequency}(w)++ \\
\quad \quad \text{ENDIF} \\
\quad \text{ENDDO} \\
\text{ENDDO} \\
\forall w \in W \text{ DO} \\
\quad \text{RelativeFrequency}(w) = \frac{\text{Frequency}(w)}{\text{Count}(S)} \\
\quad \text{IF } \text{relativefrequency}(w) > \text{occurrence frequency threshold} \\
\quad \quad k \leftarrow k \cup w \\
\quad \text{ENDIF} \\
\text{ENDDO}
\]

**Step 2: Compute keyword cohesion.**

*Keyword cohesion* is the relative frequency of the co-occurrence of keywords in requirement statements. Computing all possible combinations of co-occurring keywords
would be an NP-complete problem because answers are verifiable [NIST 2001].

However, to keep the search space manageable, we consider the co-occurrence of pairs of keywords across requirement statements. The algorithm for this process is shown next.

**Step 3: Generate keyword clusters.**

A keyword cluster is defined as keywords that are common across requirement statements. The clustering process, therefore, begins with keyword pairs that have the highest keyword cohesion. First, sets of requirement statements that include each such keyword pair are identified. Next, the intersection of these sets identifies the requirement statements that include all such keyword pairs. If a complete intersection produces a null set, the process is adjusted to consider subsets. Elements remaining in the set are then subjected to the same steps. The resulting set of (common) keywords represents a keyword cluster to which the corresponding designs are mapped. The mapping is

\[
\forall \ s \in S \text{ DO} \\
\forall k \wedge k \in S \text{ DO} \\
\forall k, k \in S \text{ DO} \\
\text{Co-occurrence}(k, k)++ \\
\text{ENDDO} \\
\text{ENDDO} \\
\text{ENDDO} \\
\forall k, k \in k \text{ DO} \\
\text{Cohesion}(k, k) = \text{Co-occurrence}(k, k)/\text{Count}(S) \\
\text{ENDDO}
\]

\[
\text{cohesion level} = \max(\text{cohesion}(k, k))
\]

\[
\text{WHILE (cohesion level > halt threshold)} \\
\forall (k, k) \text{ s.t. } \text{cohesion}(k, k) = \text{cohesion level} \\
S_i = \{s_i\} \text{ s.t. } k, k \in s_i \\
\text{ENDDO} \\
\forall s_i \\
\text{CommonKeywords} = \{k_i\} \text{ s.t. } k_i \in s_i \wedge k \in s_i \\
\forall s_i \\
\text{IF (count(CommonKeywords) / count(k \in s) > keyword coverage threshold)} \\
\text{Cluster}_{new} = \text{Cluster}_{new} \cup s_i \\
\text{ENDIF} \\
\text{ENDIF} \\
\text{cohesion level} = \text{cohesion level} - 1
\]

\[
\text{ENDWHILE}
\]
controlled by a user-defined, keyword coverage threshold, to ensure that the keyword cluster covers a significant fraction of the requirement statements.

The process repeats for the next keyword cohesion level, creating successive keyword clusters until the cohesion level reaches the halt threshold. Each design may, in this process, be mapped to one or more clusters. The algorithm for this process is shown above.

**An Illustration**

Table 5.3 shows (a) ten requirement statements for a domain (human resource management). Figure 5.9 shows the corresponding design for one of the requirement statements. The complete set of requirements and designs is the input to our example. For keyword clustering, we use the requirement statements.

**Step 1: identifying keywords**

The requirement statements are parsed to significant keywords. Table 5.3 shows the significant words in each requirement statements.

<table>
<thead>
<tr>
<th>Requirement Statement</th>
<th>Significant words</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 To keep company’s employee agreements that include specific items.</td>
<td>Company employee agreements include specific items</td>
</tr>
<tr>
<td>S2 To keep company’s employee agreements that include agreement line items.</td>
<td>Company employee agreements agreement line items</td>
</tr>
<tr>
<td>S3 To keep agreements that include agreement line items, specific items, and performance.</td>
<td>Agreements agreement line items specific items performance</td>
</tr>
</tbody>
</table>
To keep company employee’s agreement line items.  Company employee agreement line items

To keep employee agreements that include agreement line items and specific items.  Employee agreements agreement line items specific items

To keep employee agreements that include specific items and performance.  Employee agreements specific items performance

To keep employee agreements that include specific items and their steps.  Employee agreements specific items steps

To keep employee’s agreement line items and their steps.  Employee agreement line items steps

To keep company worker’s performance and working steps.  Company worker performance working steps

To keep employee’s performance and working steps and to record specific items in contract.  Employee performance working steps specific items contract

Table 5.3  Significant words in requirement statements

Keywords are identified by using an occurrence frequency threshold of, for example, 30%, which means that the keywords occur in at least 30% of the requirement statements. This threshold value is used only for illustration purposes. Other threshold values used in this example are also for illustration, with a more completed discussion on the robustness of these values given below. Keywords are identified as $k_1$: company, $k_2$: employee, $k_3$: agreement, $k_4$: specific item, $k_5$: agreement line item, $k_6$: performance, $k_7$: step. Table 5.4 shows the keywords in each requirement statement.

<table>
<thead>
<tr>
<th>Requirement Statement</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>$k_1,k_2,k_3,k_4$</td>
</tr>
<tr>
<td>S₂</td>
<td>$k_1,k_2,k_3,k_5$</td>
</tr>
<tr>
<td>S₃</td>
<td>$k_3,k_4,k_5,k_6$</td>
</tr>
<tr>
<td>S₄</td>
<td>$k_1,k_2,k_5$</td>
</tr>
<tr>
<td>S₅</td>
<td>$k_2,k_3,k_4,k_5$</td>
</tr>
<tr>
<td>S₆</td>
<td>$k_2,k_3,k_4,k_6$</td>
</tr>
<tr>
<td>S₇</td>
<td>$k_2,k_3,k_4,k_7$</td>
</tr>
<tr>
<td>S₈</td>
<td>$k_2,k_5,k_7$</td>
</tr>
<tr>
<td>S₉</td>
<td>$k_1,k_6,k_7$</td>
</tr>
<tr>
<td>S₁₀</td>
<td>$k_2,k_4,k_6,k_7$</td>
</tr>
</tbody>
</table>

Table 5.4  Keywords in requirement statements

Step 2: Computing Keyword Cohesion
Pairwise co-occurrences of keywords across requirement statements yield keyword cohesion. For example, keywords $k_2$ and $k_3$ appear in $S_1$, $S_2$, $S_5$, $S_6$, and $S_7$, which is in 50% of the statements. Following the algorithm described earlier, table 5.5 shows the keyword cohesion for each pair with the requirement statements mapped to each keyword pair.

<table>
<thead>
<tr>
<th>Keyword Pairs</th>
<th>Keyword Cohesion</th>
<th>Requirement Statements Mapped</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1, k_2$</td>
<td>30%</td>
<td>$S_1$, $S_2$, $S_4$</td>
</tr>
<tr>
<td>$k_2, k_3$</td>
<td>50%</td>
<td>$S_1$, $S_2$, $S_5$, $S_6$, $S_7$</td>
</tr>
<tr>
<td>$k_3, k_4$</td>
<td>50%</td>
<td>$S_1$, $S_3$, $S_5$, $S_6$, $S_7$</td>
</tr>
<tr>
<td>$k_4, k_5$</td>
<td>20%</td>
<td>$S_3$, $S_5$</td>
</tr>
<tr>
<td>$k_5, k_6$</td>
<td>10%</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$k_6, k_7$</td>
<td>20%</td>
<td>$S_9$, $S_{10}$</td>
</tr>
<tr>
<td>$k_1, k_3$</td>
<td>20%</td>
<td>$S_1$, $S_2$</td>
</tr>
<tr>
<td>$k_2, k_4$</td>
<td>50%</td>
<td>$S_1$, $S_5$, $S_6$, $S_7$, $S_{10}$</td>
</tr>
<tr>
<td>$k_3, k_5$</td>
<td>30%</td>
<td>$S_2$, $S_3$, $S_5$</td>
</tr>
<tr>
<td>$k_4, k_6$</td>
<td>30%</td>
<td>$S_3$, $S_6$, $S_{10}$</td>
</tr>
<tr>
<td>$k_5, k_7$</td>
<td>10%</td>
<td>$S_8$</td>
</tr>
<tr>
<td>$k_1, k_4$</td>
<td>10%</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$k_2, k_5$</td>
<td>40%</td>
<td>$S_2$, $S_4$, $S_5$, $S_8$</td>
</tr>
<tr>
<td>$k_3, k_6$</td>
<td>20%</td>
<td>$S_3$, $S_6$</td>
</tr>
<tr>
<td>$k_4, k_7$</td>
<td>20%</td>
<td>$S_7$, $S_{10}$</td>
</tr>
<tr>
<td>$k_1, k_5$</td>
<td>20%</td>
<td>$S_2$, $S_4$</td>
</tr>
<tr>
<td>$k_2, k_6$</td>
<td>20%</td>
<td>$S_6$, $S_{10}$</td>
</tr>
<tr>
<td>$k_3, k_7$</td>
<td>10%</td>
<td>$S_7$</td>
</tr>
<tr>
<td>$k_1, k_6$</td>
<td>10%</td>
<td>$S_9$</td>
</tr>
<tr>
<td>$k_2, k_7$</td>
<td>30%</td>
<td>$S_7$, $S_8$, $S_{10}$</td>
</tr>
<tr>
<td>$k_1, k_7$</td>
<td>10%</td>
<td>$S_9$</td>
</tr>
</tbody>
</table>

Table 5.5 Keyword Cohesion

Step 3. Generating keyword Clusters

The keyword cohesion table shows that keyword pairs $[k_2, k_3]$, $[k_2, k_4]$, and $[k_3, k_4]$, have the highest cohesion rate, 50%. An intersection of requirement statements mapped to each pair yields $S_1$, $S_5$, $S_6$ and $S_7$. Reverting to table 5.4, keywords that are common across these requirement statements are identified ($k_2$, $k_3$, $k_4$). This cluster of
keywords \((k_2, k_3, k_4)\) is compared against the number of keywords in each requirement statement by using a keyword coverage threshold of, for example, 60\%. All requirement statements that satisfy the threshold are mapped to the keyword cluster. In this case, the coverage rate of these keywords in each requirement statement is 75\%, which is over the threshold level. Then, of the requirement statements that remain for each keyword pair (\(S_2\) for \([k_2, k_3]\), \(S_{10}\) for \([k_2, k_4]\), and \(S_3\) for \([k_3, k_4]\)), no further keyword clusters are identified, other than the intersection of requirement statements (\(S_1, S_5, S_6\) and \(S_7\)). The next highest cohesion rate is 40\% for \([k_2, k_3]\). The requirement statements for four designs (\(S_2, S_4, S_5,\) and \(S_8\)) share this pair of keywords (table 5.5). These requirement statements are examined for common keywords (table 5.4) subject to the keyword coverage threshold. The cluster \([k_1, k_2, k_5]\) is shared by two designs (\(S_2\) and \(S_4\)) at 75\% coverage; the cluster \([k_2, k_3, k_5]\) is shared by two designs (\(S_2\) and \(S_5\)) at 75\% coverage, and the cluster \([k_2, k_3]\) is shared by \(S_4\) and \(S_8\) at 67\% coverage. Since all coverage rates are over the threshold level, 60\%, these clusters are identified and the designs mapped to them. The process is repeated at successive cohesion levels, identifying additional clusters. The results appear in table 5.6.

<table>
<thead>
<tr>
<th>Keyword clusters</th>
<th>Requirement statements mapped to the cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1 ((k_2:\text{employee}, k_3:\text{agreement}, k_4:\text{specific item}))</td>
<td>(S_1, S_6, S_7)</td>
</tr>
<tr>
<td>Cluster 2 ((k_1:\text{company}, k_2:\text{employee}, k_5:\text{agreement line item}))</td>
<td>(S_2, S_6)</td>
</tr>
<tr>
<td>Cluster 3 ((k_2:\text{employee}, k_3:\text{agreement}, k_5:\text{agreement line item}))</td>
<td>(S_2, S_5)</td>
</tr>
<tr>
<td>Cluster 4 ((k_2:\text{employee}, k_5:\text{agreement line item}))</td>
<td>(S_4, S_8)</td>
</tr>
<tr>
<td>Cluster 5 ((k_1:\text{company}, k_2:\text{employee}, k_3:\text{agreement}))</td>
<td>(S_7, S_{10})</td>
</tr>
<tr>
<td>Cluster 6 ((k_2:\text{employee}, k_3:\text{specific item}, k_5:\text{step}))</td>
<td>(S_3, S_5)</td>
</tr>
<tr>
<td>Cluster 7 ((k_3:\text{agreement}, k_4:\text{specific item}, k_5:\text{agreement line item}))</td>
<td>(S_3, S_6)</td>
</tr>
<tr>
<td>Cluster 8 ((k_3:\text{agreement}, k_5:\text{specific item, k_6:performance}))</td>
<td>(S_6, S_{10})</td>
</tr>
<tr>
<td>Cluster 9 ((k_2:\text{employee}, k_4:\text{specific item, k_6:performance}))</td>
<td>(S_9)</td>
</tr>
</tbody>
</table>

Table 5.6  Keyword clusters
5.4.2 Design Clustering

The objective of this phase is to identify commonalities in the patterns used and instantiated with keywords; that is, capture the fragments of designs that are suitable for certain contexts. As a result of this phase, fragments that contain instantiated common pattern sets (iCPSs) are identified and mapped against one or more designs. For example, if some designs share two analysis patterns, $p_1$ and $p_2$, with similar instantiations, $p_{i1}$ and $p_{i2}$, then represent the iCPS for those designs.

**Step 1: Identify seed objects set.**

The process begins with identifying analysis patterns that are most frequently used and similarly instantiated across available designs. Objects contained in these patterns constitute the seed set of objects.

**Step 2: Divide designs into groups.**

The designs are divided into two groups: designs that share the seed set and those that do not.
Step 3: Iterate

Within each group, a new seed set is identified that is the most frequently used in that group. Each group of designs is, thus, further divided again into two groups based upon the presence or absence of the new seed set. The process continues recursively until there is no shared seed set in a predetermined (*design remainder threshold*) number of designs. The recursive process results in a tree-like structure where each node contains a seed set and each path terminates in a leaf node. A design fragment, then, represents the accumulation of seed object sets that can be identified by traversing a path. The accompanying box shows the algorithm.

∀ \( d \in D \) DO
  \( p \leftarrow \text{Find the most frequently used pattern (a seed pattern) among designs in } \{D_k\} \)
  \( n(D_k) \leftarrow \text{Count the number of designs in } \{D_k\} \)
  IF \( n(D_k) > \text{threshold} \)
    \( \text{CPS}_k = \text{CPS}_{k-1} + p \)
    \( \{D_k\} \leftarrow \{D_k\} \setminus \{p \notin D_k\} \)
    Recur to the top of the do-while loop
  \{D_k\} \leftarrow \{D_k\} \setminus \{p \notin D_k\} \)
  ELSE
    Cluster(\{D_k\}, \text{CPS}_k) \leftarrow \text{Register } \{D_k\} \text{ as a new cluster of designs represented by } \text{CPS}_k
  END IF
  WHILE \( n(D_k) > \text{design remainder threshold} \)
ENDDO
An Illustration

Continuing the example, figure 5.10 (along with figure 5.9) shows the available designs. Each represents instantiated and synthesized analysis patterns, as shown in table 5.7, from the requirements in table 5.3.

![Designs Diagram]

Figure 5.10 Sample designs assembled from analysis patterns
Step 1: Identifying the seed object set.

Table 5.7 shows the analysis patterns used and their instantiations in each design.

From these, the object set ‘employee – agreement’ instantiated from the pattern

*Participant – Transaction* represents the most frequently used object set. This is identified as the seed object set.

<table>
<thead>
<tr>
<th>Designs</th>
<th>Instantiated Analysis Patterns (legend: [Pattern, instantiation])</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>[Actor-Participant company-employee], [Participant-Transaction employee-agreement], [Transaction-Place agreement-department], [Transaction-Subsequent Transaction agreement-subsequent agreement], [Transaction-Transaction line item agreement-agreement line item], [Transaction-Specific item agreement-specific item]</td>
</tr>
<tr>
<td>D2</td>
<td>[Actor-Participant company-employee], [Participant-Transaction employee-agreement], [Transaction-Subsequent Transaction agreement-subsequent agreement], [Transaction-Transaction line item agreement-agreement line item], [Transaction-Specific item agreement-specific item], [Plan-Step performance-step], [Plan-Plan Execution performance-performance execution]</td>
</tr>
<tr>
<td>D3</td>
<td>[Actor-Participant company-employee], [Participant-Transaction employee-agreement], [Transaction-Subsequent Transaction agreement-subsequent agreement], [Transaction-Transaction line item agreement-agreement line item], [Transaction-Specific item agreement-skilled], [Transaction-Specific item agreement-knowledge], [Specific item-Specific line item skill-skill line item], [Specific item-item skill-item], [Specific item-Hierarchical item skill-hierarchical skill item], [Specific item-Specific line item knowledge-knowledge line item], [Specific item-item knowledge-item], [Specific item-Hierarchical item knowledge-hierarchical knowledge item]</td>
</tr>
<tr>
<td>D4</td>
<td>[Actor-Participant company-employee], [Participant-Transaction employee-contract]</td>
</tr>
<tr>
<td>D5</td>
<td>[Participant-Transaction employee-contract], [Transaction-Place contract-department], [Transaction-Transaction line item contract-contract line item]</td>
</tr>
<tr>
<td>D6</td>
<td>[Participant-Transaction employee-agreement], [Transaction-Subsequent Transaction agreement-subsequent agreement], [Transaction-Transaction line item contract-contract line item], [Transaction-Specific item agreement-specific item]</td>
</tr>
<tr>
<td>D7</td>
<td>[Actor-Participant company-employee], [Participant-Transaction employee-agreement], [Transaction-Place agreement-department], [Transaction-Subsequent Transaction agreement-subsequent agreement], [Transaction-Transaction line item agreement-agreement line item], [Transaction-Specific item agreement-specific item]</td>
</tr>
<tr>
<td>D8</td>
<td>[Actor-Participant company-employee], [Participant-Transaction employee-agreement], [Transaction-Place agreement-department], [Transaction-Subsequent Transaction agreement-subsequent agreement], [Transaction-Transaction line item agreement-agreement line item], [Transaction-Specific item agreement-specific item]</td>
</tr>
<tr>
<td>D9</td>
<td>[Participant-Transaction employee-agreement], [Transaction-Subsequent Transaction agreement-subsequent agreement], [Transaction-Transaction line item agreement-agreement line item], [Transaction-Specific item agreement-specific item]</td>
</tr>
<tr>
<td>D10</td>
<td>[Participant-Transaction employee-contract], [Transaction-Transaction line item contract-contract line item]</td>
</tr>
</tbody>
</table>

*Table 5.7 Instantiated analysis patterns in each design*
Step 2: Dividing designs into groups.

The designs are divided into two groups – one that includes the seed object set \(\{D_k\mid p_{s0} \in D_k\}\) and another that does not \(\{D_k\mid p_{s0} \notin D_k\}\). The first group consists of seven designs \((D_1, D_2, D_3, D_6, D_7, D_8, D_9)\) that include the seed set. The second group consists of three designs \((D_4, D_5, D_{10})\) that do not include the seed set.

Step 3: Iterating

Within each group, the next most frequently shared seed object set is identified, and the grouping is repeated. We trace the first group in this illustration. This group of designs shares three more patterns, ‘agreement -- agreement line item’ \((p_{s1})\), ‘agreement – subsequent agreement’ \((p_{s2})\) and ‘agreement – specific item’ \((p_{s3})\). Continuing, six designs \((D_1, D_2, D_3, D_7, D_8, D_9)\) share ‘company – employee’ \((p_{s4})\). Of these, three \((D_1, D_7, D_8)\) share ‘department – agreement’ \((p_{s5})\). The process continues until there are no more patterns shared by a group of designs, where the number of designs satisfies the design remainder threshold. Tracing the path to the leaf then identifies the design fragment. For the above, the design fragment consists of patterns \(p_{s1}\) to \(p_{s5}\). Table 5.8 shows the design fragment \((DF)\). The surviving designs at the leaf \((D_1, D_7\) and \(D_8)\) are mapped to the design fragment. Table 5.8 shows all fragments and their mappings. The design fragments may be cumulative, following the meta-model (figure 5.3), and the mapping may be many-to-many, although the results below do not illustrate this.

<table>
<thead>
<tr>
<th>Design Fragment</th>
<th>Instantiated Analysis Patterns in the Fragment</th>
<th>Corresponding Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF(_1)</td>
<td>[employee-contract]</td>
<td>(D_1)</td>
</tr>
<tr>
<td>DF(_2)</td>
<td>[employee-contract], [contract-contract line item]</td>
<td>(D_5, D_{10})</td>
</tr>
<tr>
<td>DF(_3)</td>
<td>[employee-agreement], [agreement-subsequent agreement], [agreement-agreement line item], [agreement-specific item]</td>
<td>(D_6)</td>
</tr>
</tbody>
</table>
The steps outlined above generate design fragments that may be reused for new designs. Keyword-based clustering provides an access mechanism to these design fragments that an approach to design with reuse can exploit. New requirement statements can be parsed to identify significant words, from which keywords are identified. These keyword are eventually clustered to represent the given requirement statements. Designs that are mapped to this keyword cluster can then be examined and corresponding design fragments retrieved. The retrieval of appropriate design fragments can, therefore, be automated.

To demonstrate the reuse of design fragments created from our example, consider three cases from the personnel management domain. Table 5.9 shows the results from the retrieval process. The process consists of two simple steps: identifying keyword clusters, and retrieving design fragments. Using the clusters, corresponding designs are identified (table 5.6). The design fragment mapped against each design identified is then retrieved.
benefits being used by each employee.”

| 2 | company, employee | Cluster 2 | D_2 | DF_4 |
|   |                  |          | D_4 | DF_1 |
|   |                  | Cluster 4 | D_1 | DF_5 |
|   |                  |          | D_2 | DF_4 |

**Table 5.9 Results of the Design Fragment Retrieval Process**

**Step 1: Identifying keyword clusters**

The process begins by parsing the new requirement statement to identify keywords, which generates significant words. Among the significant words in case 1, for example, the keywords ‘employee’ and ‘step’ are identified. To map these keywords to keyword cluster(s), a new requirement coverage threshold determines the relevance. For example, keyword cluster 6 (table 5.6) is relevant because the identified keywords (employee, step) cover 67% of the keyword cluster (above a new requirement coverage threshold of, say, 60%).

Multiple clusters may be found to be relevant. Table 5.9, for instance, shows two clusters for case 2. The algorithm for identifying keyword clusters is given to the right.
Step 2: Retrieving design fragments

Keyword cluster 6 suggests two designs (D_7 and D_{10}), which in turn, suggest design fragment 5 and design fragment 2. The data model (figure 5.8) and tables 5.6 and 5.8 show the access paths followed for this retrieval, with the algorithm shown above. To generate the final design, the designer may choose one or more of the suggested design fragments, fully or in part. For example, if a designer selects to combine DF_2 and DF_5, the result would appear as shown in figure 5.11.

\[
\forall d_k \in D_k \\
\text{if } (d_k \in \text{ClusterCPS}) \\
\text{SuggestedCPS} = \text{ClusterCPS} \\
\text{ENDIF} \\
\text{ENDDO}
\]

5.6 Simulating the Creation and Reuse of a Design Fragment Repository

The viability of design fragments depends upon the availability of a reasonably large number of conceptual models in a domain, which is difficult to obtain. Therefore, we can use simulation to help us generate enough surrogate designs that can be used as

Figure 5.12 Simulation process
inputs. An overview of the simulation process is shown in figure 5.12. To demonstrate the usefulness of our approach, design fragment repositories were built for two domains, warehouse management and human resource management. These were selected because an initial set of keywords was available. The following describes the steps and results obtained for the first domain.

5.6.1 Generating Inputs

A set of available keywords for the warehouse domain was obtained from prior research. [Purao et al, 2001], augmented with additional keywords collected from relevant web sites. Web sites were selected since they provided an independent source of relevant keywords. The 37 keywords collected from the web sites were then supplemented by 31 synonyms acquired from a thesaurus to result in the 68 keywords shown in Table 5.10.

<table>
<thead>
<tr>
<th>Original keywords (37)</th>
<th>Synonyms (31)</th>
</tr>
</thead>
<tbody>
<tr>
<td>loading_dock, conveyor, order, item, warehouse, inventory, record, account, registration, catalog, inventory_tracking, frequent_item, rare_item, backorder, shipment, delivery, bin, shelf, supply, manager, management, product, report, cargo, asset, equipment, barcode, deck, forklift, manufacturer, part, compound_part, plan, sale, retail, wholesale, package</td>
<td>dock, article, storehouse, stockroom, store, depot, recurrent_item, regular_item, repeated_item, everyday_item, uncommon_item, infrequent_item, consignment, load, storage_bin, silo, container, rack, stock, director, supervisor, merchandise, freight, apparatus, tool, floor, producer, firm, maker, company, component</td>
</tr>
</tbody>
</table>

**Table 5.10 Pool of keywords in warehouse management domain**

A set of keywords from this pool is a surrogate for a requirement statement because it is, in essence, a requirement statement with the stop words removed. Thus, subsets of keywords from this pool were randomly selected to generate the input requirement statements. The number of keywords (i.e., size of requirement statement) in
each subset was varied between 4 and 11 following prior research [Purao et al., 2001]. The range [4 – 11], however, may be changed without affecting the execution of steps.

5.6.2 Keyword Clustering

The first step is to generate keyword clusters, where each cluster represents common fragments of requirements. An important task for the clustering process is the identification of good threshold values. Two thresholds are required for keyword clustering. The first, a halt threshold, terminates the iterative process of keyword cluster identification. A value of median cohesion rate was considered appropriate for this threshold because it would exploit keyword-pairs with higher-than-median cohesion (see, for example, table 5.5). The second, a keyword coverage threshold reflects how much of the requirement statements are covered. Fixing the value of the first threshold, multiple simulation runs were carried out by varying the value of the second threshold between 30% and 70% (with 70% indicating that a significant fraction of the requirement statements are met). Table 5.11 shows these thresholds and values.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Purpose</th>
<th>Value/Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowest cohesion level (halt threshold)</td>
<td>To stop the process of generating keyword clusters</td>
<td>Median cohesion rate</td>
</tr>
<tr>
<td>Minimum coverage of design (keyword coverage threshold)</td>
<td>To determine viability of a keyword cluster in terms of coverage of requirements</td>
<td>30% - 70%</td>
</tr>
</tbody>
</table>

**Table 5.11 Thresholds for keyword clustering**

Table 5.12 shows the results for varying the number of input keyword sets and the keyword coverage threshold.

<table>
<thead>
<tr>
<th>Input keyword sets</th>
<th>Keyword coverage threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30%</td>
</tr>
<tr>
<td>1000</td>
<td>7103</td>
</tr>
<tr>
<td>2000</td>
<td>20911</td>
</tr>
</tbody>
</table>

100
Table 5.12 Number of keyword clusters generated

<table>
<thead>
<tr>
<th>Input</th>
<th>Keyword clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>35513 24167 8149 1190 76</td>
</tr>
<tr>
<td>4000</td>
<td>50648 36275 13265 2165 132</td>
</tr>
<tr>
<td>5000</td>
<td>76892 60025 19767 3214 238</td>
</tr>
<tr>
<td>6000</td>
<td>102641 98524 26592 4397 359</td>
</tr>
<tr>
<td>7000</td>
<td>154986 118892 30368 4956 446</td>
</tr>
<tr>
<td>8000</td>
<td>175032 142860 39667 6024 545</td>
</tr>
<tr>
<td>9000</td>
<td>220569 175367 47633 7506 621</td>
</tr>
<tr>
<td>10000</td>
<td>292465 205106 59042 8871 818</td>
</tr>
</tbody>
</table>

The results are shown in figure 5.13. The number of keyword clusters generated represents a trade-off. A higher number represents the possibility that appropriate design fragments are available for new applications. Too few keyword clusters could result in not finding any match against new applications. In this sense, we may want a low keyword coverage threshold because it generates more keyword clusters. On the other hand, stricter (higher) keyword coverage threshold value may increases the quality of the suggested design fragments because each keyword cluster is defined with a greater number of shared keywords across the design. The resulting keyword clusters should, therefore, be more valuable to a new requirement...
statement. It is, therefore, necessary to identify a value of the *keyword coverage threshold* that balances the number of keyword clusters against their quality.

Based on the values for the keyword coverage threshold, we observe that for higher coverage, the number of keyword clusters decreases rapidly for a given level of input keyword sets. At threshold values of 30% and 40%, the number of clusters is considerably high and climbs rapidly as seen in figure 5.13. At values of 60% and 70%, the number of clusters is considerably low and climbs slowly. More interestingly, the result of 60% does not show the proper distribution of designs that maps to keyword clusters. As shown in table 5.13, at a threshold value of 60%, approximately 40-50% of the input designs do not map to any clusters, compared to only about 15% at a threshold value of 50%. Also the average number of keyword clusters to which a design is mapped is about 1 at 60%, and over 6 at 50%. Thus, threshold values of 60% and above are not desirable. A threshold of 50% is most appropriate because it generates a large number of high quality keyword clusters.

<table>
<thead>
<tr>
<th>Input keyword sets (designs)</th>
<th>At keyword coverage threshold of 50%</th>
<th>At keyword coverage threshold of 60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average number of designs in a cluster</td>
<td>Average number of clusters to which a design is mapped</td>
<td>Number of designs that are not mapped to any cluster</td>
</tr>
<tr>
<td>3000</td>
<td>2.19</td>
<td>5.93</td>
</tr>
<tr>
<td>4000</td>
<td>2.21</td>
<td>7.35</td>
</tr>
<tr>
<td>5000</td>
<td>2.22</td>
<td>8.77</td>
</tr>
<tr>
<td>6000</td>
<td>2.24</td>
<td>9.92</td>
</tr>
</tbody>
</table>

*Table 5.13 Design distribution at keyword coverage threshold of 50% and 60%*
5.6.3 Design Clustering

Each randomly generated keyword set was used to generate a conceptual design following the automated approach of figure 5.6. The number of objects in these designs was varied between 8 and 30 and the number of relationships between 5 and 39 (see figure 5.14). The automated approach produced some designs that included disconnected sets of objects and relationships because they were based on instantiation and synthesis of available patterns, which were also not fully connected. Similar to keyword clustering, the important decision during design clustering is the design remainder threshold value. The design remainder threshold determines the condition for terminating the recursive process of design clustering. Four different values were used as shown in table 5.14.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Purpose</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of designs remaining to be processed (design remainder threshold)</td>
<td>To stop clustering when few designs remain.</td>
<td>5 – 20</td>
</tr>
</tbody>
</table>

Table 5.14 Parameter and threshold in design clustering simulation

Several simulation runs were carried out varying this threshold with the results shown in table 5.15.

<table>
<thead>
<tr>
<th>Input designs</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>107</td>
<td>53</td>
<td>39</td>
<td>31</td>
</tr>
</tbody>
</table>
The results show trends similar to those for keyword clustering. As the number of designs increases, the number of generated design fragments increases (table 5.15 and figure 5.15). The number of fragments is considerably smaller than the number of keyword clusters (table 5.12) since designs are generated from fewer analysis patterns. Deciding the value for the design remainder threshold is straightforward because it does not require a trade-off between the number of design clusters and the quality of the design fragment. The greater the number of design clusters, the more precise the design fragments. A lower threshold means that the clustering process continues until few designs remain. Thus, we choose 5 as the design remainder threshold.

<table>
<thead>
<tr>
<th>Input designs</th>
<th>Number of design fragments generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>211 108 74 59</td>
</tr>
<tr>
<td>3000</td>
<td>318 162 103 87</td>
</tr>
<tr>
<td>4000</td>
<td>416 201 138 113</td>
</tr>
<tr>
<td>5000</td>
<td>507 268 148 145</td>
</tr>
<tr>
<td>6000</td>
<td>604 322 206 168</td>
</tr>
<tr>
<td>7000</td>
<td>729 389 258 195</td>
</tr>
<tr>
<td>8000</td>
<td>787 430 289 231</td>
</tr>
<tr>
<td>9000</td>
<td>884 482 316 256</td>
</tr>
<tr>
<td>10000</td>
<td>985 589 364 288</td>
</tr>
</tbody>
</table>

Table 5.15 Number of design fragments generated

Figure 5.15 Fragments Generated (at varying thresholds and inputs)
5.6.4 Retrieving Design Fragments

To illustrate the retrieval of design fragments, new keyword sets (to indicate new sets of requirement statements) were generated. The number of keywords in each new set was varied between 4 and 7. Keywords were randomly selected from the 68 keywords (table 5.10). A mapping algorithm identified keyword clusters and design fragments for the new application. The values of the new requirements coverage threshold for the design fragments retrieval are shown in table 5.16 and the results of the simulation shown in tables 5.17, 5.18, 5.19, and 5.20. This threshold determines similarity between the new requirements and any design cluster(s) matched.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Purpose</th>
<th>Range of value</th>
</tr>
</thead>
<tbody>
<tr>
<td>New requirement coverage threshold</td>
<td>Portion of shared keywords in the keyword set of new application and in the matching keyword cluster to find matching design fragments – how much similarity between keywords in the new application and keywords in keyword clusters</td>
<td>50% – 80%</td>
</tr>
</tbody>
</table>

Table 5.16 Parameter and threshold in reuse simulation

<table>
<thead>
<tr>
<th># of input designs</th>
<th># of new keywords</th>
<th>1000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># of suggested design fragments</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3-16</td>
<td>27-139</td>
<td>59-148</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>11-23</td>
<td>65-141</td>
<td>21-107</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0-18</td>
<td>55-235</td>
<td>154-210</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0-24</td>
<td>11-99</td>
<td>77-147</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not shown due to too many results

Table 5.17 Retrieving design fragments (threshold = 50%)

<table>
<thead>
<tr>
<th># of input designs</th>
<th># of new keywords</th>
<th>1000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># of suggested design fragments</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2-4</td>
<td>20-41</td>
<td>36-124</td>
<td>76-141</td>
<td>86-120</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2-11</td>
<td>5-86</td>
<td>4-26</td>
<td>36-165</td>
<td>65-240</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3-11</td>
<td>5-21</td>
<td>10-24</td>
<td>6-74</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0-5</td>
<td>0-3</td>
<td>0-9</td>
<td>0-8</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.18 Retrieving design fragments (threshold = 60%)

<table>
<thead>
<tr>
<th># of input designs</th>
<th># of new keywords</th>
<th>1000</th>
<th>6000</th>
<th>7000</th>
<th>8000</th>
<th>9000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># of suggested design fragments</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2-4</td>
<td>2-4</td>
<td>6-38</td>
<td>36-38</td>
<td>17-24</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2-4</td>
<td>2-6</td>
<td>8-24</td>
<td>16-45</td>
<td>6-34</td>
<td></td>
</tr>
</tbody>
</table>
Tables 5.17 through 5.20 show the number of suggested design fragments for different sizes (number of keywords). As the threshold is set higher, the number of design fragment suggested is lower. This is understandable since the number of inputs to our simulation (1000 to 9000) is a small fraction of all possible combinations of 68 keywords for 4 to 11 elements; that is, the keyword clusters in the repository cannot cover all possible new cases.

This prompts us to search for a heuristic solution. For example, at the 80% level of the new requirement coverage threshold, it does not appear to be easy to get enough suggestions for most cases. At 50%, it seems much easier to have suggestions, but it provides too many suggestions, which is not practical either. Thus, either 60% (table 5.18) or 70% (table 5.19) of the new requirement coverage threshold level appears to provide a good trade-off in terms of the number of suggestions. However, even at these new requirement coverage threshold levels, there are too many suggestions for the cases with less than 6 keywords. The solution to this problem could be to flexibly apply threshold values depending on the number of designs in the repository and the number of keywords in a new application. For example, if we have 7000 input designs with 4
keywords in the new application, we may apply an 80% threshold level (table 5.20), but for 5-6 keywords, we can apply a 60% threshold level (table 5.18).

It also appears to be difficult to consistently obtain suggestions with more than 8 keywords (not shown in tables), which is more than 80% of keywords that a design can maximally have in the current simulation set. It appears that we need far more than 9000 designs to make any suggestions for those cases, which may be infeasible or undesirable. A possible solution is to allow designers to divide their requirement statement into small pieces, each of which includes a moderate number of keywords (say less than 7).

5.7 An Application

This section illustrates the actual reuse of design fragments. Starting from new requirement statements, we retrieve and combine design fragments to generate a new design. This is illustrated on two different domains: warehouse management and human resource management. Two new requirement statements for each domain were obtained from prior research [Purao et al., 2001] as shown in table 5.21. The minimum cohesion threshold was set at the median cohesion rate and the keyword coverage threshold rate set at 50%. The design remainder threshold was set at 5. 6000 designs were used as input. The design fragment repository described in the prior section was used for the warehouse management domain. The design fragment repository for human resource management domain was created with 53 keywords (40 original and 13 synonyms).

<table>
<thead>
<tr>
<th>Domain</th>
<th>Case</th>
<th>Requirement Statement (keywords in bold face)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse Management</td>
<td>W1</td>
<td>The warehouse management system will track different items, each of which has a unique number. There are several docks in the warehouse and a network of conveyor belts. The most frequently ordered items are kept closest on hand for picking efficiency. The system will also be responsible for tracking items in the inventory and will be able to order or reorder items when their level reaches a predetermined level.</td>
</tr>
</tbody>
</table>
The warehouse management system will be responsible for assembling orders as required into one or more shipments. The warehouse contains many different stock items, which are kept in different bins and shelves. The deliveries received by the warehouse are recorded by the system, which increases the stock levels appropriately. Any stockouts are recorded by the system as backorders and when new stock arrives, the backorders are filled.

The system will be responsible for storing personal information of our employees. It will be used to record performance evaluation. The system will also track benefits, maintain skill profiles and may need to match employees to different projects based on skill requirements Finally, employee benefits will also be recorded.

We need to develop a personnel tracking application to locate personnel based on skills, to keep track of employee turnover to schedule work shifts, to compute employee benefits. The system will also create and classify employee skills to allow allocation of skills to departments as required. In addition to the usual requirements about leave records and employee benefits, it will also record training sessions attended by employees as well as expense authorizations.

### Table 5.21 Requirement statements

Design fragments retrieved for each case are shown in table 5.22 and figures 5.18 through 5.21. The second case of each domain has a large number of keywords (11 and 13 respectively). They are, therefore, divided into two small sets of keywords. Each set generates design fragments that are combined to create the final designs. Examples of the final designs generated by combining design fragments are shown in figures 5.16 and 5.17.

<table>
<thead>
<tr>
<th>Case</th>
<th>Number of keywords</th>
<th>Number of keywords divided</th>
<th>Number of design fragments suggested</th>
<th>Number of objects in design fragments</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>7</td>
<td></td>
<td>4</td>
<td>7, 9, 9, 12</td>
</tr>
<tr>
<td>W2</td>
<td>11</td>
<td>5 and 6</td>
<td>5</td>
<td>13, 11, 13, 6, 14</td>
</tr>
<tr>
<td>H1</td>
<td>7</td>
<td></td>
<td>5</td>
<td>6, 7, 8, 10, 12</td>
</tr>
<tr>
<td>H2</td>
<td>13</td>
<td>6 and 7</td>
<td>4</td>
<td>10, 10, 11, 11</td>
</tr>
</tbody>
</table>

### Table 5.22 Size of design fragments suggested

At 60% of new requirement coverage threshold level, all cases suggested a proper number of design fragments. The number of objects in the suggested design fragments varied from 6 to 14. As shown in appendices A through D, suggested design fragments
for each case covered most of keywords in the requirement statements for each case and generated other reasonable objects.

![Figure 5.16. Final design generated from design fragments (W1)](image)

![Figure 5.17. Final design generated from design fragments (H2)](image)

### 5.8 Conclusion

Design fragments have been proposed as a reusable artifact that is highly (re)useful with a higher level of granularity and a lower level of abstractness than previously developed reusable artifacts. They are inductively built by identifying
common parts of existing designs by a series of algorithms. The built-in algorithms make
the automation of both building and retrieving of design fragments possible, and
consequently reduce human efforts in building, retrieving, and reusing reusable artifacts.
This increases usability and (re)usefulness of design fragments. Table 5.23 shows
evaluation of design fragments.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Evaluation of Design Fragments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>Build-in mechanisms to retrieve – through keyword clusters and design clusters; natural language processing</td>
</tr>
<tr>
<td>Ease of Retrieval</td>
<td>Easy to superimpose; domain specificity; graphical representation</td>
</tr>
<tr>
<td>Ease of Assembly</td>
<td>Easy to superimpose; domain specificity; graphical representation</td>
</tr>
<tr>
<td>(Re)usefulness</td>
<td>Medium or above medium granularity yielding high productivity</td>
</tr>
<tr>
<td>Abstractness</td>
<td>Low abstraction and domain specificity making minimal demands on designers’ adaptation and integration efforts</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Creation Effort: Minimal effort to create following automated creation</td>
</tr>
<tr>
<td></td>
<td>Reuse Effort: Minimal effort for reuse following automated retrieval</td>
</tr>
</tbody>
</table>

Table 5.23 Evaluation of Design Fragments

A methodology has been proposed for the development of design fragments that
simulates stereotypical designs based upon keywords to form a repository of design
fragments. Retrieved design fragments may be combined by a designer to produce a
design for new requirement statements.

Design fragments should prove to be useful based upon our criteria for evaluation
of reusable artifacts (table 5.1). As demonstrated, in terms of usability, they score high on
ease of retrieval and ease of assembly. In terms of (re)usefulness, a design fragment is of
medium granularity and has a relatively low level of abstraction. For efficiency, it scores
medium on creation effort and low on reuse effort. It, thus, appears from our initial
research, that, as a new reusable artifact, design fragments score better than previously
created artifacts on a number of significant criteria, as desired.
To illustrate the feasibility of the employment of the design fragment artifact, the methodology was applied to several design problems, two of which have been shown in detail. Future research will include application to other domains and testing on real world problems. Additional research will test the adoption intention of designers based upon a comparison of design fragments to other reusable artifacts such as analysis patterns.
Figure 5.18. Suggested design fragments for case W1
(Un-instantiated objects in bold face)
Figure 5.19. Suggested design fragments for case W2
(Un-instantiated objects in bold face)
Figure 5.20. Suggested design fragments for case H1
(Un-instantiated objects in bold face)
Figure 5.21. Suggested design fragments for case H2
(Un-instantiated objects in bold face)

- **Design Fragment 1**
  - Subsequent Transaction
  - Participant
  - manager
  - training
  - specific item
  - training line item
  - file
  - resume
  - line item

- **Design Fragment 2**
  - Subsequent Transaction
  - Participant
  - specific item
  - authorization
  - leave record
  - record
  - line item

- **Design Fragment 3**
  - Subsequent Transaction
  - Participant
  - specific item
  - record
  - location
  - line item
  - skill
  - turnover

- **Design Fragment 4**
  - authorization line item
  - employee
  - record
  - specific item
  - record line item
  - location
  - skill
  - line item
  - item
CHAPTER 6
EXPLORATION OF ADOPTION INTENTION OF DESIGN FRAGMENTS

This chapter explores a preliminary model to test adoption intention of prospective systems developers with respect to design fragments. As described in Chapter 5, the three intrinsic properties of design fragments are usability, (re)usefulness, and efficiency. (Re)usefulness indicates whether the artifact addresses a common need and provides a commonly requested service. Two dimensions that contribute to the (re)usefulness of an artifact (from literature on reuse) are: granularity and abstraction. Usability indicates whether the artifact is easy to understand and apply for new system development. Two dimensions that contribute to the usability of an artifact (from literature on reuse) are: ease of retrieval and ease of assembly. Finally, efficiency refers to the effort involved for creation or reuse of the artifact. This is operationalized as the support available for creation of the artifact and the support available during the reuse-based design with the artifact. The methodology and simulation processes for creating the repository and retrieving the design fragments described in chapter 5 addresses the criteria of efficiency. This exploration, therefore, focuses on assessing the other two properties: usability and (re)usefulness.

6.1 Research Model

To operationalize the constructs of interest, reusability and (re)usefulness, we draw on the established constructs of perceived usefulness and perceived ease of use from the Technology Acceptance Model (TAM) [Davis, 1989]. A related interest in operationalizing the constructs in this manner is that following TAM, high levels of
perceived usefulness and perceived ease of use can be used to predict adoption intention.

Another key advantage is that TAM’s explanatory power and measurement validity are supported by many studies in different empirical settings characterized by user group, technology, and organizational context [Agarwal and Prasad, 2000; Chau, 1996; Venkatesh, 2000; Lucas and Spitler, 1999]. The validity of measurement scales for TAM has also been scrutinized [Doll et al., 1998; Straub, 1989].

Accordingly, we consider (re)usefulness and usability as external variables that are antecedents of perceived usefulness and perceived ease of use respectively. There have been many studies exploring antecedents to usefulness and ease of use for different information technologies [Lederer et al., 2000; Venkatesh, 2000; Moon and Kim, 2001; Chau, 1996; Agarwal and Prasad, 2000]. Venkatesh [2000] adopted ‘anchor’ and ‘adjustment’. Four constructs (computer self-efficacy, facilitating conditions, computer anxiety, and computer playfulness) were used for anchor and two constructs (perceived enjoyment and objective usability) for adjustment. Agarwal and Prasad [1999] tested the influence of individual differences such as role, tenure, education, experience, and training on the beliefs about usefulness and ease of use. Chau [1996] studied external factors such as implementation gap and transitional support on usefulness and ease of use. In the current study, we focus on how two intrinsic characteristics of design fragments – reusability and reusefulness – would act as antecedents of perceived usefulness and perceived ease of use, which may eventually provide the basis on acceptance of design fragments. Figure 6.1 shows the research model.
6.2 Hypotheses

To understand the relative advantage of our proposed reusable artifact, design fragment, we compare it to an existing reusable artifact, analysis pattern. We hypothesize that reusing analysis patterns would be neither easy nor useful compared to our proposal, design fragments. Thus, the first set of hypotheses directly compares design fragments with analysis patterns in terms of each variable in the model. We expect that design fragments will be perceived as less granular and less abstract, that is, more useful. We also expect that they will be perceived as easier to retrieve and assemble, that is, more usable.

**Hypotheses Set 1:** Design fragments are better than analysis patterns in (re)usefulness and usability, and therefore perceived usefulness and perceived ease of use.

**H1A:** (Re)usefulness of analysis pattern < (Re)usefulness of design fragment
  - **H1A1:** Granularity of analysis pattern < Granularity of design fragment
  - **H1A2:** Abstractness of design fragment < Abstractness of analysis pattern

**H1B:** Usability of analysis pattern < Usability design fragment
  - **H1B1:** Ease of Retrieval of analysis pattern < Ease of Retrieval design fragment
  - **H1B2:** Ease of Assembly of analysis pattern < Ease of Assembly design fragment
H1C: Perceived usefulness of analysis pattern < Perceived usefulness of design fragment
H1D: Perceived ease of use of analysis pattern < Perceived ease of use design fragment

The next hypotheses set examines the influence of design fragment properties on the user’s beliefs about usefulness and ease of use. We expect that the levels of granularity and abstractness of design fragments will be correlated to the user’s beliefs about usefulness, and reusability.

**Hypotheses Set 2**: (Re)usefulness and usability of design fragments are positively correlated to the corresponding constructs, perceived usefulness and perceived ease of use.

**H2A**: (Re)usefulness of design fragments is positively related to perceived usefulness.
**H2B**: Usability of design fragments is positively related to perceived ease of use.

### 6.3 Instrument Development and Experiment

Use of TAM is advantageous because of its well-researched and validated measurements. Preliminary measurements for perceived usefulness and perceived ease of use were obtained from prior studies [Davis, 1989; Hu et al., 1999] to formulate the questionnaire items, using a seven-point Likert scale with anchors ranging from strongly agree to strongly disagree. On the other hand, since design fragment is a new kind of artifact, there have not been available question items for constructs for (re)usefulness and reusability of design fragments. A review panel consisting of three design experts was used, instead, to evaluate the face and content validity of the instruments. To ensure desired balance of the items in the questionnaire, several question items were negated to invite the attention of subjects. Question items related to each construct were grouped to avoid possible measurement errors from confusions and irritation among subjects [Davis
and Venkatesh, 1996]. After a pilot test, the questionnaires were modified and the review panel reexamined the formatted survey instruments to ensure its layout and wording. Following these rounds, two instruments were developed for the experiment. One included question items about designing without assistance and the other about designing with reuse of analysis patterns and design fragments. The instruments are provided in the Appendix J.

The experiment consisted of repeated trials. Each subject performed three tasks: create a design without assistance, create a design by reusing analysis patterns, and create a design by reusing design fragments. The subjects were provided information about analysis patterns and design fragments before the experiment. Task one was done first for all subjects to avoid a possible learning bias. After the first task, the first instrument was administered. The subjects were then randomly divided into two groups. One group performed task two first and then task three, the other group did task three first and then task two, to avoid a possible order bias. Requirement statements were designed in three domains, to ensure that the subjects performed each task in different domains to avoid the possible effect of domain familiarity on the performance. After task three, the second instrument was administered. This instrument included identical question items for analysis patterns and design patterns. The experiment and survey process is depicted in Figure 6.2. The tasks are provided in the Appendix K.
6.4 Results and Data Analysis

Of total of fifty cases acquired, seventeen cases were not usable due to excessive missing data and inconsistencies, yielding thirty-three usable cases. And first, measurement validity in terms of reliability was evaluated. Specifically, reliability was evaluated using Cronbach’s alpha. Constructs adopted from TAM – perceived usefulness and perceived ease of use – showed high alpha values (over 0.80), indicating high reliability, in all tasks as expected since they are from the validated instrument.

However, new constructs of (re)usefulness and usability did not show high reliability. Granularity and abstractness showed low reliability in all tasks. Ease of retrieval and ease of assembly showed high reliability in unaided design and design with analysis patterns, but not in design with design fragments. This indicates that items of (re)usefulness and reusability in the instrument need to be refined in the continuing research. Construct validity of the instrument was evaluated by examining convergent and discriminant validity using principal component factor analysis. Again, constructs from TAM – perceived usefulness and perceived ease of use – showed convergence and discrimination to some degree, especially in design with design fragments. The new variables did not show clear convergence and discrimination. The reason could be that,
first, the sample size is not enough for performing factor analysis, and second, there were considerable number of unmotivated subjects, thought from the fact that 34% of data acquired were unusable. Acquiring more data and motivating subject are other points to be addressed in the further study.

To test the first set of hypotheses, t-tests were performed. The results show that design fragments are better than analysis patterns in usability (ease of retrieval and ease of assembly), perceived usefulness, and perceived ease of use (see table 6.1). Average scores of design fragments in these constructs are significantly higher than those of analysis patterns. Thus hypotheses H1B1, H1B2, H1C, and H1D are accepted. However, differences in (re)usefulness (granularity and abstractness) are not significant. Thus, hypotheses H1A1 and H1A2 are not accepted. This indicates that the user do not seem to feel difference between design fragments and analysis patterns in granularity and abstractness, but somehow they perceive that design fragments are more useful than analysis patterns regardless of granularity and abstractness. Eventually, this might positively influence the acceptance of design fragments. The antecedents of this may not, however, be granularity and abstractness.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Reusable Artifact</th>
<th>Mean</th>
<th>t</th>
<th>Hypothesis Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granularity</td>
<td>Analysis Pattern</td>
<td>4.2656</td>
<td>.077</td>
<td>H1A1 Not Accepted</td>
</tr>
<tr>
<td></td>
<td>Design Fragment</td>
<td>4.2500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abstractness</td>
<td>Analysis Pattern</td>
<td>3.7424</td>
<td>1.672</td>
<td>H1A2 Not Accepted</td>
</tr>
<tr>
<td></td>
<td>Design Fragment</td>
<td>4.1515</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of retrieval</td>
<td>Analysis Pattern</td>
<td>3.5606</td>
<td>3.803**</td>
<td>H1B1 Accepted</td>
</tr>
<tr>
<td></td>
<td>Design Fragment</td>
<td>4.5758</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of assembly</td>
<td>Analysis Pattern</td>
<td>3.3030</td>
<td>3.513**</td>
<td>H1B2 Accepted</td>
</tr>
<tr>
<td></td>
<td>Design Fragment</td>
<td>4.2121</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perceived usefulness</td>
<td>Analysis Pattern</td>
<td>4.0949</td>
<td>3.623**</td>
<td>H1C Accepted</td>
</tr>
<tr>
<td></td>
<td>Design Fragment</td>
<td>4.8424</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perceived ease of use</td>
<td>Analysis Pattern</td>
<td>3.4502</td>
<td>3.666**</td>
<td>H1D Accepted</td>
</tr>
<tr>
<td></td>
<td>Design Fragment</td>
<td>4.2294</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1  t-test results for hypotheses set 1
The second set of hypotheses examines the influence of design fragment properties on independent variables of TAM, (re)usefulness on perceived usefulness and usability on perceived ease of use. As seen in Table 6.2, (re)usefulness is not significantly related to perceived usefulness as expected from the test result of hypotheses set 1. Thus, hypothesis H2A is not accepted. Meanwhile, hypothesis H2B is accepted (F-value=7.731, p<.001). Of usability properties of design fragments, however, ease of retrieval is significantly and positively related to perceived ease of use (β=.419, t-value=2.303, p<.05), but ease of assembly is not. This may reflect that assembly of given design fragments still require some efforts, although it is significantly easier than using analysis patterns. Meanwhile, retrieval of appropriate design fragment is fully automated and the user appears to perceive the easiness.

<table>
<thead>
<tr>
<th>Model</th>
<th>R²</th>
<th>β</th>
<th>F</th>
<th>t</th>
<th>Hypothesis Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU = GRAN + ABS + errors</td>
<td>.009</td>
<td>.128</td>
<td>.098</td>
<td>.500</td>
<td>H2A Not Accepted</td>
</tr>
<tr>
<td>GRAN</td>
<td>.098</td>
<td>.500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABS</td>
<td>-.019</td>
<td>-.099</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEOU = ER + EA + errors</td>
<td>.340</td>
<td>7.731**</td>
<td>.419</td>
<td>2.303*</td>
<td>H2B Accepted</td>
</tr>
<tr>
<td>ER</td>
<td>.419</td>
<td>2.303*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td>.230</td>
<td>1.262</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2  Regression results for hypotheses set 2

Overall, design fragments are better than analysis patterns on at least usability property and the user perceives so both on usefulness and ease of use. The regression results show that the (re)usefulness (granularity and abstractness) does not seem to affect perceived usefulness. But usability (ease of retrieval and ease of assembly) seems to be important for perceived ease of use. The fact that the user perceive that design fragments are more useful and easy to use, and that reusability of design fragments is proven that it
affects one variable (perceived ease of use) of TAM would increase the possibility of acceptance of design fragments in the future. Research is necessary to further investigate these constructs and their role as antecedents of TAM to reliably predict the adoption intentions of prospective developers.
CHAPTER 7
CONCLUSION

This study developed, implemented, and tested a set of related approaches to facilitate system design with reuse. The research was done in three phases. In the first phase, a naive approach of designing object-oriented systems with reuse of analysis patterns was augmented with learning mechanisms. The augmented approach was validated across domains and fall different task sizes through a lab experiment. The results showed the augmented approach is better than the naive approach, scalable over different task sizes, and transferable across domains. Designs synthesized by analysis patterns in this phase provided the opportunity of creating a new kind of reusable artifact – design fragment. In phase two, design fragments were identified from designs generated in phase one, and organized to be efficiently retrieved for new applications. Two clustering mechanisms were developed for the process. Feasibility of this phase was assessed and demonstrated by simulation. Finally in phase three, the adoption possibility of design fragments was explored by extending TAM. Overall results were positive, which indicates design fragments are likely to be used by developers.

7.1 Contributions

This study contributes in several aspects over three phases. First, it proposed learning mechanisms to augment the naive approach to reuse-based design and provided empirical evidence that the augmented approach improves reuse-based design. The empirical test showed that the augmented approach is scalable over task sizes and transferable across domains. Second, this study developed a new kind of reusable
artifact, called design fragment. Compared to existing reusable artifacts, design fragments provide relatively high granularity and, at the same time, low abstractness, which indicates that they are easy to reuse and still able to yield higher productivity. Built-in support for building and retrieving makes design fragments more efficient. Finally, this study developed constructs extending TAM, which can be antecedents of adoption intention to design fragments by developers.
APPENDIX A: A SAMPLE DESIGN GENERATED WITH THE NAÏVE APPROACH

Input to the Design Process: “a system to track sales at different stores”

Task 1: Identification of Significant Words
Words: system, sale, different, store

Task 2: Identification of Objects
Objects: Transaction, Place, Container. Designer Removes TransactionLineItem

Task 3: Retrieval of Relevant Patterns
8 Patterns Retrieved

Task 4: Instantiation of Retrieved Patterns
4 Patterns Instantiated Fully, 4 More Instantiated Partially.

Task 5: Synthesis of Instantiated Patterns
Patterns Synthesized using Shared Objects (such as Participant-Transaction and Place-Transaction), and Objects superimposed using similar instantiations, such as Store (Place and Container).

The Resulting Design with the Naïve Approach is shown below, along with all the attributes and methods for the generic classes suggested by the patterns.
APPENDIX B: TASK DESCRIPTIONS USED FOR THE EXPERIMENT

The task descriptions were compiled using a composite of fragments from statements generated by a users during early testing (see section 4.4.3). The low complexity task statement was designed to provide partial coverage of knowledge contained in the naïve approach, whereas the medium complexity task statement was designed to provide an almost complete coverage of knowledge contained in the augmented approach. Finally, the high complexity task statement was designed to go beyond the knowledge available even in the augmented approach (see figure 4-5).

Domain: Warehouse Management

Low Complexity: We need to develop an automated system to control the conveyor and route each item to the loading dock we assign to its order.

Medium Complexity: The warehouse management system will track different items, each of which has a unique number. There are several docks in the warehouse and a network of conveyor belts. The most frequently ordered items are kept closest on hand for picking efficiency. The system will also be responsible for tracking items in the inventory and will be able to order or reorder items when their level reaches a predetermined level.

High Complexity: The warehouse management system will be responsible for assembling orders as required into one or more shipments. The warehouse contains many different stock items, which are kept in different bins and shelves. The stock items have a shelf life and may need to be discarded and replenished if they are not shipped before the shelf life is over. The deliveries received by the warehouse are recorded by the system, which increases the stock levels appropriately. Any stockouts are recorded by the system as backorders and when new stock arrives, the backorders are filled. Shipments of backorders are then recorded along with the original orders.

Domain: Human Resource Management

Low Complexity: We need to develop a system to keep track of our different employees, their qualifications, their leave records and benefits.

Medium Complexity: The system will be responsible for storing resume & personal information of our employees. It will be used to record & report performance evaluation which may be used later in promotion decisions. The system will also track benefits, maintain skill profiles and may need to match employees to different projects based on skill requirements. Finally, employee benefits will also be recorded.

High Complexity: A personnel tracking application to locate personnel based on skills, to keep track of employee turnover, to schedule work shifts, to compute employee benefits. The system will also create and classify employee skills to allow allocation of skills to departments as required. It will also track candidates and their resumes in a resume bank.
In addition to the usual requirements about leave records and employee benefits, it will also record training sessions attended by employees as well as expense authorizations and actual expense statements of employees.
APPENDIX C: ASSESSMENT SCHEME TO EVALUATE DESIGN
PRODUCT QUALITY

The table below shows the assessment scheme used to measure errors in the designs generated. The numbers in each cell indicate the weight of an error for a combination of error type and element type. Since these numbers reflect the relative importance of each modeling element these scores provide a relative ranking among models.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Modeling Element</th>
<th>Object</th>
<th>Instantiation</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type I: Errors of Omission</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing Elements</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-1, if due to missing objects</td>
</tr>
<tr>
<td>Type II: Errors of Commission</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incorrect Elements</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-1, if due to incorrect objects</td>
</tr>
<tr>
<td>Redundant Elements</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1, if due to redundant objects</td>
</tr>
<tr>
<td>Plausible Elements</td>
<td>+1</td>
<td>N.A.</td>
<td>+1</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Weights of error types and modeling elements

The assessment scheme is based on the two commonly used error types - type I (errors of omission) and type II (errors of commission). It identifies four kinds of primitive errors - missing, incorrect, redundant and plausible. It also distinguishes between major and minor errors and identifies reverse errors. Major errors seriously affect the quality of a model, and are penalized at 3 points per error. Minor errors have less of an effect on the quality of a model than major errors, and, therefore, are assigned a weight of 1 point per error. To avoid double penalization, some errors are penalized at a lower weight if they were caused by other errors. Finally, reverse errors represent reasonable elements in the generated model that were not anticipated by the ideal model. Since point since the decision between incorrect and plausible is subjective, the reverse error is conservatively assigned 1 point.

Consider, for example, an ideal model composed of 8 objects and 7 associations. The size of the ideal model, then, is \((8 + 7) \times 3 = 45\). Suppose also that a generated model contains 9 objects and 6 associations with errors as below. The design will be scored using the above assessment scheme as:

\[
\begin{align*}
\text{Total error score} &= -21 \\
\text{Error rate is computed} &\quad (21/45) \times 100 = 0.4667
\end{align*}
\]
Thank you for agreeing to participate in this Research Study!

Your Task

Your task involves constructing a preliminary design of an application system. For performing this task, you will use an experimental tool, APSARA: Automated Pattern Retrieval and Synthesis Assistant.

The Tool

APSARA contains several patterns with which it assists you, the designer. For example, one of its patterns is 'Transaction-Subsequent Transaction.' With this, it may suggest to you specific parts of your model such as 'Order-Shipmont' or 'Reservation-Rental' or 'Accident-Ticket' depending upon your application domain.

The Steps

The Tool is organized as a series of steps. As each step is done, you will have the opportunity to adjust the results generated. At the end of each step, there will be a link that you can use to proceed to the next step. Remember, APSARA is a research prototype. That means it is not robust. Please follow the instructions carefully. If you have any questions, please ask the Research Coordinator.

Preliminaries

The Research Coordinator will collect the Informed Consent Form from you and get you started.

Start

Step 1: Enter System Requirements

1. The Research Coordinator will give you a TXT file containing the system requirements.
2. Follow directions in the applet below.
3. Click the ‘Parse Requirements’ button.
4. Move on to the next step by selecting ‘Next Step’ link.

Step 2: Identify Keywords

The requirements statement is parsed to identify keywords.

1. To adjust the results, follow directions in the applet below.
2. Click the ‘Identify Objects’ button.
3. Move on to the next step by selecting ‘Next Step’ link.
Step 3: Identify Objects

Based on the keywords identified, objects are retrieved from the patterns-base. The keywords ||| objects are separated by vertical lines.

1. To adjust the results, follow directions in the applet below.
2. Click the 'Identify Patterns' button.
3. Move on to the next step by selecting 'Next Step' link.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Object Identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>warehouse</td>
<td>place</td>
</tr>
<tr>
<td>warehouse</td>
<td>container</td>
</tr>
<tr>
<td>management</td>
<td>transaction</td>
</tr>
<tr>
<td>management</td>
<td>plan</td>
</tr>
</tbody>
</table>

To delete keyword-object pair, select it and use the 'Delete Keyword-Object' button.

Add any new keyword in the box below, select corresponding object from the list below, and click 'Add' button.

Add

---

Next Step (Did you click on the [Identify Objects] button above first?)

---

Step 4: Identify Patterns

Based on the objects identified, patterns are retrieved from the patterns-base. The keywords ||| objects ||| patterns are separated by vertical lines.

1. To adjust the results, follow directions in the applet below.
2. Click the 'Instantiate Patterns' button.
3. Move on to the next step by selecting 'Next Step' link.

### Step 5: Instantiate Patterns

The patterns retrieved are instantiated, fully or partially, using the keywords that lead to their retrieval. The keyword || object || patterns instantiated are separated by vertical lines.

1. To adjust the results, follow directions in the applet below.
2. Click the 'Identify Unique Patterns' button.
3. Move on to the next step by selecting 'Next Step' link.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Object Identified</th>
<th>Patterns Identified</th>
</tr>
</thead>
</table>
| warehouse||place||place-transaction
| warehouse||container||container-content
| warehouse||container||container-content-lineitem
| management||transaction||participant-transaction

**Next Step** (Did you click on the [Instantiate Patterns] button above first?)
Step 6: Identify Unique Patterns

The redundant patterns and instantiations are removed. The results are shown as pattern instantiation ||| pattern retrieved, separated by vertical lines.

1. To adjust the results, follow directions in the applet below.
2. Click the 'Apply Superimpose' button.
3. Move on to the next step by selecting 'Next Step' link.
Step 7: Superimpose Patterns

Patterns with corresponding instantiations are superimposed.

1. To adjust the results, follow directions in the applet below.
2. Move on to the next step by selecting 'Next Step' link.
Step 8: Confirm Your Design

1. Type date and your initials followed by the example number at the bottom of the TXT file.
2. For example, William Jefferson Clinton working on example 22 at 10 Nov would be Nov10WJC22.
3. Click the 'Confirm' button.

Done? (Did you click on the [Save] button above first?)

1. If there are additional problems to solve, please turn off the program and start the next session.
2. If you are done, thanks for participating in the study. Please consult the Research Coordinator to complete the process and ensure your registration for the Best Design Award.
APPENDIX E. PHASE 1 EXPERIMENT INTERFACE FOR NAIVE APPROACH

http://cis.gsu.edu/~than/apsara/apsara-naive.html

Thank you for agreeing to participate in this Research Study!

Your Task

Your task involves a constructing a preliminary design of an application system. For performing this task, you will use an experimental tool, APSARA: Automated Pattern Retrieval and Synthesis Assistant.

The Tool

APSARA contains several patterns with which it assists you, the designer. For example, one of its patterns is 'Transaction-Subsequent Transaction.' With this, it may suggest to you specific parts of your model such as 'Order-Shipment' or 'Reservation-Rental' or 'Accident-Ticket' depending upon your application domain.

The Steps

The Tool is organized as a series of steps. As each step is done, you will have the opportunity to adjust the results generated. At the end of each step, there will be a link that you can use to proceed to the next step. Remember, APSARA is a research prototype. That means it is not robust. Please follow the instructions carefully. If you have any questions, please ask the Research Coordinator.

Preliminaries

The Research Coordinator will collect the Informed Consent Form from you and get you started.

Start

Step 1: Enter System Requirements

1. The Research Coordinator will give you a TXT file containing the system requirements.
2. Follow directions in the applet below.
3. Click the ‘Parse Requirements’ button.
4. Move on to the next step by selecting ‘Next Step’ link.

The warehouse management system will be responsible for assembling orders as required into one or more shipments. The warehouse contains many different stock items which are kept in different bin and shelf. The stock items have a shelf life and may need to be discarded and replenished if they are not shipped before the shelf life is over. The delivery received by the warehouse are recorded by the system which increases the stock level appropriately. Any stockout are recorded by the system as backorder and when new stock arrives the backorder are filled. Shipment of backorder are then recorded along with the original order.

Step 2: Identify Keywords

The requirements statement is parsed to identify keywords.

1. Click the ‘Identify Objects’ button.
2. Move on to the next step by selecting ‘Next Step’ link.

Step 3: Identify Objects

Based on the keywords identified, objects are retrieved from the patterns-base. The keywords are separated by vertical lines.
1. Click the 'Identify Patterns' button.
2. Move on to the next step by selecting 'Next Step' link.

### Step 4: Identify Patterns

Based on the objects identified, patterns are retrieved from the patterns-base. The keywords ||| objects ||| patterns are separated by vertical lines.

1. Click the 'Instantiate Patterns' button.
2. Move on to the next step by selecting 'Next Step' link.

### Step 5: Instantiate Patterns

The patterns retrieved are instantiated, fully or partially, using the keywords that lead to their retrieval. The keyword ||| object ||| pattern instantiated are separated by vertical lines.

1. Click the 'Identify Unique Patterns' button.
2. Move on to the next step by selecting 'Next Step' link.
Step 6: Identify Unique Patterns

The redundant patterns and instantiations are removed. The results are shown as pattern instantiation ||| pattern retrieved, separated by vertical lines.

1. Click the 'Apply Superimpose' button.
2. Move on to the next step by selecting 'Next Step' link.

Step 7: Superimpose Patterns

Patterns with corresponding instantiations are superimposed.

1. Move on to the next step by selecting 'Next Step' link.
Step 8: Confirm Your Design

1. Type date and your initials followed by the example number at the bottom of the TXT file.
2. For example, William Jefferson Clinton working on example 22 at 10 Nov would be Nov10WJC22.
3. Click the 'Confirm' button.

Done? (Did you click on the [Save] button above first?)

1. If there are additional problems to solve, please turn off the program and start the next session.
2. If you are done, thanks for participating in the study. Please consult the Research Coordinator to complete the process and ensure your registration for the Best Design Award.
APPENDIX F. USAGE HISTORY DATABASE ERD

http://cis.gsu.edu/~than/apsara/db/
APPENDIX G. SOLUTION TECHNIQUES

This appendix describes solution techniques used to solve the research problems posed.

G.1 Machine Learning

Machine learning is the key to machine intelligence just as learning is the key to human intelligence. Progress in machine learning has become central to the development of the field of artificial intelligence (AI) as a whole and affects almost all of its sub-areas. Many algorithms have been invented that are effective for certain types of learning tasks, and a theoretical understanding of learning is beginning to emerge. Many practical computer programs have been developed to exhibit useful types of learning, and significant commercial applications have begun to appear.

Classifications of the paradigms in the field of machine learning are slightly different among researchers, but the major classifications are not largely different. The most sophisticated classification is probably the one by Michalski and Kodratoff (1990). They classify learning process based on the primary purpose of the learning method, the type of input information, the type of primary inference employed, and the role of the learner’s prior knowledge. Other papers that discuss taxonomies are Michalski (1986), Carbonell et al. (1983), and Briscoe and Caelli (1996). This research follows Briscoe and Caelli’s taxonomy, which are organized around the underlying learning strategies used by various systems. Table G.1 shows the taxonomy of machine learning adopted from (Briscoe & Caelli, 1996).
Symbolic Empirical Learning (SEL)
- Supervised (*Learning From Examples*)
  - Decision Trees
  - Star Methodology
  - Version Spaces
  - Least Generalization
  - Inductive Logic Programming
- Unsupervised (*Learning From Observation and Discovery*)
  - Conceptual Clustering
  - Discovery
  - Reinforcement Learning

Analytical Learning / Explanation-Based Learning (EBL)
- Learning Composite Rules
- Learning Search Control Knowledge

Integrated Learning Systems
- Combining various learning techniques
- Learning Apprentice Systems
- Learning by Analogy

Table G.1 Taxonomy of the machine learning adopted and modified from Briscoe & Caelli (1996)

Two main paradigms are symbolic empirical learning (SEL) and explanation-based learning (EBL). SEL is mainly consistent with inductive learning methods in other papers. EBL is consistent with analytical learning or deductive learning. SEL (or inductive learning) involves the creation of general symbolic concept descriptions whose structures are unknown a priori. The concept descriptions that are learned are created on the basis of given examples or facts. SEL methods are empirical in that they require little or no background knowledge or domain knowledge. On the other hand, EBL (or deductive learning) relies mainly on pre-existing background knowledge and not on examples. It uses existing domain knowledge to explain and generalize concepts and theories in order to improve the efficiency of a system.
G.1.1 Symbolic Empirical Learning (SEL)

SEL is further subdivided into supervised learning (or learning from examples) and unsupervised learning (or learning from observation). In supervised learning, the output classifications are supplied by an external teacher. With unsupervised learning, the examples are not preclassified. There is no information about what the correct outputs are. The learning procedure is required to find commonalties and regularities in the data. Whereas the success of supervised learning algorithms is readily evaluated by applying the output hypothesis to a set of test examples and seeing if they are correctly classified, the success of unsupervised learning may be determined by examining the test examples and seeing if they exhibit the same regularity that was discovered in the training examples.

G.1.1.1 Supervised Learning

G.1.1.2 Unsupervised Learning

Unsupervised learning is divided into two groups. The first group collects examples into clusters sharing some commonality. The second group is based on methods traditionally used by scientists and inventors to make discoveries. An example of the former is conceptual clustering (Michalski & Stepp, 1983; Hanson, 1990; Stepp & Michalski, 1986; Gennari et al., 1989; McKusick & Thompson, 1990; Hadzikadic & Yun, 1989). The second group, called machine discovery, extends across a broad range of techniques. At one end of the scale, empirical discovery attempts to find general laws to describe observations, whereas the more ambitious theory formation tries to go beyond mere description to include more detailed explanations for the discovered laws. To date, most discovery systems have concentrated on empirical discovery, which uses relatively simple algorithms in knowledge poor environments.

G.1.1.2.1 Conceptual Clustering

Conceptual clustering is an unsupervised paradigm for induction that attempts to find appropriate classifications for a set of examples. Traditional clustering algorithms, such as cluster analysis and numerical taxonomy, classify a set of objects into sets that share common features. “Natural classes” are found from elements of a set (Michalski, 1983). Clustering is based on close associations or shared characteristics, that is, a similarity measure. The measure of similarity is usually defined as a proximity measure in a multidimensional feature space. Clusters represent collections of elements whose intra-cluster similarity is high and inter-cluster similarity is low. In statistical clustering, this similarity can be decided by a numerical distance. The presence of irrelevant
features may distort this measure of similarity. Traditional numerical methods of eliminating irrelevant attributes of classes, such as factor analysis or multidimensional scaling, are inadequate for nominal (categorical) attributes. Also, numerical measures fail to take account of any contextual information.

Figure G.1 An Example of Conceptual Clustering

Clustering can be conceptual using semantic and syntactic knowledge about the elements. For example, in Figure G.1, the point y may be numerically clustered into the circle B; however, with knowledge of the circle, it is more appropriately clustered in A (Michalski, 1986).

Michalski and Stepp (1983) define conceptual clustering as “a process of constructing a concept network that characterizes a collection of objects, with nodes marked by concepts describing object classes, and links marked by the relationships between classes.” Conceptual clustering attempts to overcome the problems of traditional numerical clustering methods.

CLUSTER/2 (Michalski & Stepp, 1983) is a conceptual clustering system that belongs to the AQ star methodology family. For a given set of objects, attributes, and background knowledge, CLUSTER/2 system finds a hierarchy of object classes, in that each class is described by a single conjunctive concept. CLUSTER/S (Stepp & Michalski, 1986) extends CLUSTER/2. The representation language of objects and classes is upgraded to annotated predicate calculus, which allows for structural descriptions. COBWEB (Gennari et al., 1989; McKusick & Thompson, 1990) forms concepts by grouping objects with similar attributes. COBWEB represents clusters as a probability distribution over the space of attribute values, generating a hierarchical
classification tree. UNIMEM (Gennari et al., 1989, Lebowitz, 1987) is an incremental conceptual clustering algorithm, capable of handling large numbers of examples. Instances are attribute-value pairs, with numeric, ordinal, and simple hierarchical attributes. Multiple values are allowed for the nominal attributes, and missing values are permitted. Concepts are represented as conjunctions of attribute-value pairs, with each attribute value having an associated integer that measures a predictability score, that is, how well the feature can be predicted given an instance of the concept. UNIMEM organizes concepts into a hierarchical memory structure. Hadzikadic and Yun (1989) describe an incremental conceptual clustering system INC, which is based on evidence from human learning research suggesting that the members of categories that are considered the most prototypical are those with the most attributes in common with other members of the category, and those with the least attributes in common with other categories. Another variant of conceptual clustering is LEW (Constant et al., 1990). Whereas most clustering algorithms attempt to cluster concepts, LEW tries to cluster rules.

G.1.1.2.2 Learning by Discovery and Heuristics

Another form of unsupervised learning is discoveries. The methods include both symbolic (qualitative) discovery (e.g., AM, EURISKO) and numerical (quantitative) discovery (e.g., BACON). Most of systems in this category use heuristics. A heuristic is defined as “a rule of thumb, strategy, trick, simplification, or any other kind of device that drastically limits search for solutions in large problem spaces. Heuristics do not guarantee optimal solutions. In other words, a heuristic is a rule of thumb or judgmental
technique that leads to a solution some of the time but provides no guarantee of success. Heuristics play an important role in search strategies because of the exponential number to polynomial number and, thereby, obtain a solution in a tolerable amount of time. Bergadano and Giordana (1988) distinguish three kinds of heuristics:

**Statistical Heuristics**: From the statistical definition of a “good” description, it is possible to derive an estimate of the probability of moving toward a discriminant concept description.

**Domain Independent Heuristics**: We could define some theory of learning, or more simply, a set of criteria suggesting how to drive the search process in order to minimize the effort or to find more reliable and less complex concept descriptions.

**Domain Specific Heuristics**: All the *a priori* knowledge available for a particular domain can, in principle, be exploited in order to formulate such heuristics.

AM (Lenat, 1978; 1982; 1983) was developed to explore the process of learning by discovery. Heuristics were used to develop new knowledge. Given initial set theory and search heuristics, AM was able to rediscover many important concepts from set theory and number theory. It showed that a small set of general heuristics can guide a non-trivial discovery process. AM works with a fixed set of heuristics, with no way of improving or adding to these rules. Eventually, the efficacy of these rules gradually declines. EURISKO (Lenat, 1983) addresses this issue by incorporating the discovery of heuristics themselves. It extends the AM system to include the discovery of heuristics.
themselves\textsuperscript{6}. EURISKO uses the processes of specialization or generalization of existing heuristics, as well as using analogy to create new heuristics. It suggests experiments to try, proposes plausible synthesis and modifications of heuristics, and detects implausible constructs and actions.

The BACON suite of systems (BACON.1 through BACON.6) (Langley et al., 1983; 1986) were an attempt to apply the AI notation of search through a problem space, and the use of heuristics to direct that search, to the discovery of (scientific) numeric law. The strengths of BACON are its ability to discover simple laws relating real-valued variables, its ability to combine existing terms to create new terms, and its ability to recast the training instances on the basis of the developing hypotheses. However, it does have limitations. It has difficulty with irrelevant variables, and is unable to perform any clustering of the data or to derive multiple equations to describe different subsets of the data. The authors of BACON have developed several programs that examine other aspects of scientific discovery, specifically qualitative discovery. These systems are GLAUBER, STAHL, and DALTON (Langley et al., 1986).

G.1.1.2.3 Reinforcement Learning

Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The two most important features of reinforcement learning are trial-and-error search and delayed reward. Reinforcement learning is defined not by characterizing learning algorithms, but by characterizing a learning problem. Any algorithm that is well suited to solving that problem we consider is to be a reinforcement learning algorithm. The basic idea is simply to capture the most

\textsuperscript{6} Heuristics are used to develop new heuristics.
important aspects of the real problem facing a learning agent interacting with its environment to achieve a goal. Reinforcement learning is different from supervised learning. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In uncharted territory, an agent must be able to learn from its own experience.

G.1.2 Analytical Learning / Explanation-Based Learning (EBL)

The second major approach in machine learning is EBL (or analytical, or deductive learning). This learning paradigm corresponds to knowledge enhancement, in which the system improves its performance by exploiting its current knowledge more effectively, either by learning improved composite rules or by learning to find an appropriate rule more efficiently. With EBL, the justification for a new, learned concept is logically derived from background knowledge. The main advantage of this learning is that as little as one example, together with the background knowledge, is sufficient to acquire an operational concept description. Methods in this paradigm are EBG, EGGS, GENESIS, BAGGER2, and so on.

G.1.3 Integrated Learning Systems

Both of the two major paradigms used in machine learning have difficulties when used alone. On the one hand, SEL systems operate by finding regularities within a set of training examples, but cannot readily incorporate any background knowledge that is available. EBL systems, on the other hand, use the available domain theory to explain a single example, and in so doing form a general description of the class of the examples.
However, the reliance of EBL on background knowledge is too strong, requiring correct domain theories to for learned concepts. Thus the integrated learning systems combine SEL and EBL techniques to gain strengths of both and to overcome the weakness of either system used individually. The early integrated systems allowed only attribute-value pairs, or unary predicates. Some examples of early systems include OCCAM, IOE, and ML-SMART. Later systems were able to use function-free Horn clauses to extend the concept description language to first-order logic. These systems include FOCL, EITHER, and FORTE.

Another approach in this category is learning by analogy. Learning by analogy involves transferring certain properties of the base knowledge (a concept, a procedure, etc.) into the target knowledge. Such a transfer includes elements of both inductive and deductive learning. To recognize (or discover) an analogy, one needs to detect a similarity or common relations between the base and the target knowledge and hypothesize that it might extend beyond the compared relations. This is primarily inductive process. Once these common relations have been determined, then the transfer of properties from the base to the target knowledge is deductive. Illustrative examples of analogical concept learning are described in (Winston, 1982). A theoretical framework and a method for analogical problem solving are presented in (Carbonell, 1986).

G.2 Processing the Semantics

G.2.1 Natural Language Processing (NLP)

Natural language processing (NLP) is very large and difficult field. Though the application of NLP is limited to parsing in this research, the potential of using more
sophisticated NLP techniques in the future is large. NLP inevitably involves machine learning techniques since the machine should understand a natural language. However, in this section, we focus on the definitions and semantic aspects of NLP.

G.2.1.1 Grammars and Languages

A language can be considered as a set of strings of finite or infinite length, where a string is constructed by concatenating basic atomic elements called symbols. A grammar is a formal specification of the sentence structures that are allowable in the language. One of the basic issues of NLP is how to represent natural languages with grammar in a mathematical sense. A grammar $G$ is formally defined as

$$ G = (v_n, v_t, s, p) $$

where $v_n$ is a set of nonterminal symbols, $v_t$ a set of terminal symbols, $s$ is a starting symbol, and $p$ is a finite set of productions or rewrite rules. The production rules from $P$ represent the grammatical relationship of elements (words) of a sentence. More constrained languages (formal programming language) have been studied through the use of similar grammars, including the Chomsky classes of languages (1965). Chomsky defined a hierarchy of grammars he called types 0, 1, 2, and 3. Type 0 is the most general. Type 1 is context-sensitive grammars. Type 2 is context-free grammars. And type 3 is a finite state or regular grammar. Fillmore (1968) extended Chomsky’s grammar with case grammars to identify more correct meaning of a sentence. A case relates to the semantic role that a noun phrase plays with respect to verbs and adjectives. Other grammars are systemic grammars (Winograd, 1972) and semantic grammars (Hendrix et al. 1978).
G.2.1.2 Parsing

The process of determining the syntactical structure of a sentence is known as parsing. Parsing is the process of analyzing a sentence by taking it apart word-by-word and determining its structure from its constituent parts and subparts. To determine the meaning of a word, a parser must have access to a lexicon, which is a dictionary of words where each word contains some syntactic, semantic, and possibly some pragmatic information. The information in the lexicon is needed to help determine the function and meaning of the words in a sentence. The organization and entries of a lexicon will vary from one implementation to another.

Parsers may be designed to process a sentence using either a top-down or a bottom-up approach. A top-down parser begins by hypothesizing a sentence and successively predicting lower level constituents until individual preterminal symbols are written. A bottom-up parser, on the other hand, begins with the actual words appearing in the sentence and is, therefore, data-driven. Parsers may also be classified as deterministic or nondeterministic depending on the parsing strategy employed (Patterson, 1990). A deterministic parser permits only one choice for each word category. Thus, care must be taken to make correct test choices at each stage of the parsing since, in deterministic parses, the parser cannot backtrack to an alternative choice when incorrect test choice is accepted from some state. Nondeterministic parses permit different arcs to be labeled with the same test. The parser must guess at the proper constituent and then backtrack if the guess is later proven to be wrong. For example, for a sentence “The strong bear the loads,” if the deterministic parser chose to recognize ‘strong’ as an
adjective and ‘bear’ as a noun, the parse would fail, since there is no verb following bear. A nondeterministic parser, on the other hand, would simply recover by backtracking when failure was detected and then taking another arc that accepted ‘strong’ as a noun. Some researchers prefer to use deterministic parsing since they feel that it more closely models the way humans parse input sentences.

G.2.1.3 Semantic Analysis and Representation Structures

Finally, the most difficult stage in NLP is the semantic interpretation: how the final semantic knowledge structures are created to satisfy the requirements of a knowledge base used to represent some particular world model. The semantic structures must account for all aspects of meaning in what is known as the domain, context, and the task. Semantic interpretations require that sentences be transformed into coherent expressions in the form of first-order predicate logic, associative networks, frames, or script-like structures that can be manipulated by the understanding program.

There are a number of different approaches to the transformation problem. One approach is first to perform a syntactic analysis and produce a tree-like structure using a parser, and then to use a semantic analyzer to produce a semantic structure. Another approach is to transform the sentence directly into the target structures with little syntactic analysis. Such approaches typically depend on the use of constraints given by semantic grammars or place strong reliance on the use of key words to extract meaning. Between these two extremes, are approaches which perform syntactic and semantic analyses concurrently, using semantic information to guide parse, and the structure learned through the syntactical analysis is used to help determine meaning.
Some successful natural language understanding systems include LIFER (Hendrix, 1978) and SHRDLU (Winograd, 1972).
APPENDIX H. ADDITIONAL STATISTICS FOR PHASE 1 ANALYSIS

- Outliers have been removed.
- Data have been standardized (in other words, changed into percentage) since the current abstract values do not show the scalability for the complexity levels. For example, data #4 in HR11 (Missing:-6 Wrong/Redundant:-8 Plausible:2 Total:-12) is converted into percentage to a certain norm value. Since the ideal model of HR11 has 5 objects and 4 associations, each is counted as 3 points. So the norm value for HR11 is 27 points (3 points x 9). So the standardized data is now M:-.22 M/P:-.15 W/R:-.3 T:-.44 (P is not converted since P only is not meaningful. Instead M/P is standardized.) Consequently, standardized data show some scalability. The rationale behind using standardized value could be that we must discount the degree of complexity for errors. For example, when a designer made 1 error in a 5 object model and 4 errors in a 20 object model, should 1 error and 4 errors be considered same or different? If we think the designer must identify all objects in any model (if the number of errors is a matter), they are different. 4 errors are more than 1 error. But if we think the degree of complexity should be considered, then error rates are same. Both are 20%. This designer did same performance in both models.

Several MANOVA tests were performed with various cases, as above, of dependent variable for both abstract values and standardized values. In the body text of the dissertation, only total error rates (in other words, T with standardized values) are described. Though, the results of other cases are similar: all significant results for main effect, but for interaction effect too, which means we need to interpret the results situation by situation. The following is all the results including one described in the body text.

MANOVA

First, a Multivariate Analysis of Variance (MANOVA) was conducted to investigate the effects of three design factors (approach, domain, and complexity) on the three types of error rates – Missing/Plausible, Wrong/Redundant, and Total. The results (Table 0) show that the system design elements significantly influence the error rate (as a whole). However, since interaction effects among factors are also significant, we cannot interpret the main effect directly. Instead, the effect of each factor is examined at the combined level of other factors.

Table 0. MANOVA Table - Effects of Design Factors on Error Rates

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>F-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Effects:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td>2</td>
<td>67.90***</td>
</tr>
</tbody>
</table>

159
To determine the effects of the design factors on the specific error rate, a series of ANOVA were conducted on the three error rates respectively. Overall, the analysis indicated important main effects and interactions. One must be cautious when examining main effects in the presence of significant interaction. To interpret interactions clearly, a series of t-tests or another set of ANOVAs with multiple comparison tests based on the number of treatment groups of interests to investigate group differences.

**Missing/Plausible Error**

Table 1 shows the ANOVA results. The overall model reveals significant differences in Missing/Plausible error scores between the system design elements ($F=21.84$, df=11, $p<.001$). That is, approach, domain, and complexity of a system design altogether do account for differences in Missing/Plausible error rate. However, although all of the main effects are significant, there are also significant 2-way and 3-way interactions among factors. Due to interaction effects, therefore, the effect of each design element on M/P error rate could not be examined separately. Instead, it was investigated in the combination of the other two design elements for the groups of interest only (See t-test section).

Table 1. ANOVA Table - Effects of Design Factors on Missing/Plausible Error Rates

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>F-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main Effects:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td>1</td>
<td>50.35***</td>
</tr>
<tr>
<td>Domain</td>
<td>1</td>
<td>6.96**</td>
</tr>
<tr>
<td>Complexity</td>
<td>1</td>
<td>69.25***</td>
</tr>
<tr>
<td><strong>2-way Interactions:</strong></td>
<td></td>
<td>27.13***</td>
</tr>
<tr>
<td>Approach*Domain</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

* $p<.05$  ** $p<.01$  *** $p<.001$
Wrong/Redundant Error

To test the effects of the system design elements on Wrong/Redundant Error, an ANOVA is conducted. The results shown in Table 2 indicated that W/R Error is significantly different based on the system design elements. There was no significant 3-way interaction among three design factors. In addition, there was no significant interaction between domain and complexity. Individual t-tests other than these interactions were performed for the interesting groups (See t-test section).

Table 2. ANOVA Table - Effects of Design Factors on Wrong/Redundant Error Rates

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>F-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main Effects:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td>1</td>
<td>126.32***</td>
</tr>
<tr>
<td>Domain</td>
<td>1</td>
<td>67.98***</td>
</tr>
<tr>
<td>Complexity</td>
<td>2</td>
<td>32.26***</td>
</tr>
<tr>
<td><strong>2-way Interactions:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach*Domain</td>
<td>1</td>
<td>19.30***</td>
</tr>
<tr>
<td>Approach*Complexity</td>
<td>2</td>
<td>56.42***</td>
</tr>
<tr>
<td>Domain*Complexity</td>
<td>2</td>
<td>2.23</td>
</tr>
<tr>
<td><strong>3-way Interaction:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach<em>Domain</em>Complexity</td>
<td>2</td>
<td>.775</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>11</td>
<td>65.26***</td>
</tr>
</tbody>
</table>

* p<.05   ** p<.01   *** p<.001

Total Error

As presented in Table 3, an ANOVA confirmed that the system design elements have an effect on Total Error Scores. The overall model was significant ($F=65.26$, df=11, $p<.001$). There were also 2-way and 3-way interactions among factors.

Table 3. ANOVA Table - Effects of Design Factors on Total Error Rates

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>F-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main Effects:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>2-way Interactions:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach*Domain</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Approach*Complexity</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Domain*Complexity</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>3-way Interaction:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approach<em>Domain</em>Complexity</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

* p<.05   ** p<.01   *** p<.001
Main Effects:
- Approach 1 39.81***
- Domain 1 41.48***
- Complexity 1 127.05***

2-way Interactions:
- Approach*Domain 1 65.61***
- Approach*Complexity 2 54.75***
- Domain*Complexity 2 19.74***

3-way Interaction:
- Approach*Domain*Complexity 2 7.13**

Model 11 41.56***

* p<.05    ** p<.01    *** p<.001

T-Test

{- naïve approach A: augmented approach}

Table 4 group shows the difference between N and A in each domain and complexity.

Table 4 group. T-Tests with standardized scores (to investigate group differences with three way interactions)

Total (Standardized)

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>N: .0740</td>
<td>N: -.3333</td>
<td>N: -.8974</td>
</tr>
<tr>
<td></td>
<td>A: -.3519</td>
<td>A: -.4424</td>
<td>A: -.5055</td>
</tr>
<tr>
<td>WH</td>
<td>N: -.2381</td>
<td>N: -1.0606</td>
<td>N: -.1019</td>
</tr>
<tr>
<td></td>
<td>A: -.2429</td>
<td>A: -.4978</td>
<td>A: -.4265</td>
</tr>
</tbody>
</table>

When complex, A is better than N in both domains. In simple, difference in WH is not significant, and N is better than A in HR. It shows that A helps designer getting better result when the problem is complex. This result is contributed mainly by wrong/redundant errors (see the table below). Freedom to add/remove objects based on A’s suggestion seems to work positively to get rid of unnecessary objects in complex problem, but result in a little excessive addition in simple problem. This is backed up by the result in Table 5 group (total error rate). It shows that A is better than N in complex regardless of domain, and worse than N in simple regardless of domain. The result is mixed in medium. However, it still shows this trend. Worse A in HR medium is also due to some excessive addition of objects. Overall, in medium and complex,
wrong/redundant errors are reduced in A, and missing errors are rather a little increased (but much less effective than w/r error decrease).

**Missing (Standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
</table>
| **HR**     | N: -.2222  
A: -.2778  (NS) | N: -.2407  
A: -.4198  (S) | N: -.3846  
A: -.4524  (NS) |
| **WH**     | N: -.2381  
A: -.2476  (NS) | N: -.4848  
A: -.4221  (NS) | N: -.2157  
A: -.3873  (S) |

**Wrong/Redundant (Standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
</table>
| **HR**     | N: .0000  
A: -.2130  (S) | N: -.3148  
A: -.1481  (S) | N: -.7179  
A: -.1209  (S) |
| **WH**     | N: -.3810  
A: -.2857  (NS) | N: -.7273  
A: -.3052  (S) | N: -.9412  
A: -.2010  (S) |

**Missing + Plausible (Standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
</table>
| **HR**     | N: .0740  
A: -.1389  (S) | N: -.019  
A: -.2942  (S) | N: -.1795  
A: -.3846  (S) |
| **WH**     | N: .1429  
A: -.0428  (NS) | N: -.3333  
A: -.1926  (S) | N: -.078  
A: -.2255  (NS) |

Table 5 group. T-tests with standardized scores: to investigate group differences with 2-way interactions

**Missing (standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
</table>
| N: -.2302  
A: -.2562  (NS) | N: -.3286  
A: -.4212  (S) | N: -.3213  
A: -.4176  (S) |

**Wrong/Redundant (standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
</table>
| N: -.1905  
A: -.2649  (NS) | N: -.4633  
A: -.2437  (S) | N: -.8017  
A: -.1636  (S) |
Missing+Plausible (standardized)

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>.1085</td>
<td>-.1319</td>
<td>-.1416</td>
</tr>
<tr>
<td>A:</td>
<td>-.0091 (S)</td>
<td>-.2324 (S)</td>
<td>-.2997 (S)</td>
</tr>
</tbody>
</table>

Total (standardized)

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>-.082</td>
<td>-.5952</td>
<td>-.9433</td>
</tr>
<tr>
<td>A:</td>
<td>-.2740 (S)</td>
<td>-.4761 (NS)</td>
<td>-.4633 (S)</td>
</tr>
</tbody>
</table>

Table 6 group shows the difference between N and A in each domain regardless of complexity. The result (total error) shows that there is no difference in HR, and A is better than N in WH. We may say that in online processing domain, A is better than N. [or that there might be an effect between approaches based on domain due to the mixed result in different domains. Possibly in some domains, A is better than N, and in some domains it doesn’t work as intended.]

Table 6 group. T-tests with standardized scores: to investigate group differences with 2-way interactions

Missing

<table>
<thead>
<tr>
<th></th>
<th>HR</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>-.2843</td>
<td>-.3032</td>
</tr>
<tr>
<td>A:</td>
<td>-.4028 (S)</td>
<td>-.3589 (NS)</td>
</tr>
</tbody>
</table>

Wrong/Redundant

<table>
<thead>
<tr>
<th></th>
<th>HR</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>-.3592</td>
<td>-.6441</td>
</tr>
<tr>
<td>A:</td>
<td>-.1516 (S)</td>
<td>-.2731 (S)</td>
</tr>
</tbody>
</table>

Missing+Plausible

<table>
<thead>
<tr>
<th></th>
<th>HR</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>-.0460</td>
<td>-.062</td>
</tr>
<tr>
<td>A:</td>
<td>-.2948 (S)</td>
<td>-.1273 (NS)</td>
</tr>
</tbody>
</table>

Total

<table>
<thead>
<tr>
<th></th>
<th>HR</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>-.4053</td>
<td>-.7038</td>
</tr>
</tbody>
</table>
Scalability of A is shown. Table 7 group (total error) shows that there is no difference among complexity in each domain. In other words, errors rates of augmented approach are same among different complexity levels.

Table 7 group. ANOVA (3way): Augmented with Standardized Scores

Total (standardized)

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>-.3519</td>
</tr>
<tr>
<td>Medium</td>
<td>-.4424</td>
</tr>
</tbody>
</table>
| Complex   | -.5055| (NS)
| WH        |       |
| Simple    | -.2429|
| Medium    | -.4978|
| Complex   | -.4265| (NS)

Missing (standardized)

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>-.2778*a</td>
</tr>
<tr>
<td>Medium</td>
<td>-.4198*b</td>
</tr>
</tbody>
</table>
| Complex   | -.4524*b | (S)
| WH        |       |
| Simple    | -.2476*a |
| Medium    | -.4221*b |
| Complex   | -.3873*b | (S)

Wrong/Redundant (standardized)

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>-.2130</td>
</tr>
<tr>
<td>Medium</td>
<td>-.1481</td>
</tr>
</tbody>
</table>
| Complex   | -.1209| (NS)
| WH        |       |
| Simple    | -.2857|
| Medium    | -.3052|
| Complex   | -.2010| (NS)

Missing+Plausible (standardized)

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>-.1389*a</td>
</tr>
<tr>
<td>Medium</td>
<td>-.2942*b</td>
</tr>
</tbody>
</table>
Table 8 group shows there is difference between simple and medium/complex (a is one kind, b is another kind).

Table 8 group. One-way ANOVA: A-SMC (with Standardized Scores)

**Missing (standardized)**

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple: -.2562 *a</td>
<td></td>
</tr>
<tr>
<td>Medium: -.4212*b</td>
<td></td>
</tr>
<tr>
<td>Complex: -.4176*b</td>
<td>(S)</td>
</tr>
</tbody>
</table>

**Wrong/Redundant(standardized)**

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple: -.2649</td>
<td></td>
</tr>
<tr>
<td>Medium: -.2437</td>
<td></td>
</tr>
<tr>
<td>Complex: -.1636</td>
<td>(NS)</td>
</tr>
</tbody>
</table>

**Missing+Plausible (standardized)**

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple: -.0091*a</td>
<td></td>
</tr>
<tr>
<td>Medium: -.2324*b</td>
<td></td>
</tr>
<tr>
<td>Complex: -.2997*b</td>
<td>(S)</td>
</tr>
</tbody>
</table>

**Total (standardized)**

<table>
<thead>
<tr>
<th>Augmented</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple: -.2740*a</td>
<td></td>
</tr>
<tr>
<td>Medium: -.4761*b</td>
<td></td>
</tr>
<tr>
<td>Complex: -.4633*b</td>
<td>(S)</td>
</tr>
</tbody>
</table>

Table 9 group also shows that there is no difference of A between domains in each complexity level. In other words, augmented approach is scalable between domains in each complexity level. Table 10 group also shows no difference of A among domains regardless of complexity level.
Table 9 group. T-tests with standardized scores (3-way): (Augmented) with three way interactions

**Missing (standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR: -0.2778, WH: -0.2476</td>
<td>HR: -0.4198, WH: -0.4221</td>
<td>HR: -0.4524, WH: -0.3873</td>
</tr>
<tr>
<td></td>
<td>(NS)</td>
<td>(NS)</td>
<td>(NS)</td>
</tr>
</tbody>
</table>

**Wrong/Redundant (standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR: -0.2130, WH: -0.2857</td>
<td>HR: -0.1481, WH: -0.3052</td>
<td>HR: -0.1209, WH: -0.2010</td>
</tr>
<tr>
<td></td>
<td>(NS)</td>
<td>(NS)</td>
<td>(NS)</td>
</tr>
</tbody>
</table>

**Missing+Plausible (standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR: -0.1389, WH: 0.0428</td>
<td>HR: -0.2942, WH: -0.1926</td>
<td>HR: -0.3846, WH: -0.2255</td>
</tr>
<tr>
<td></td>
<td>(S)</td>
<td>(NS)</td>
<td>(NS)</td>
</tr>
</tbody>
</table>

**Total (standardized)**

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Medium</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR: -0.3519, WH: -0.2429</td>
<td>HR: -0.4424, WH: -0.4978</td>
<td>HR: -0.5055, WH: -0.4265</td>
</tr>
<tr>
<td></td>
<td>(NS)</td>
<td>(NS)</td>
<td>(NS)</td>
</tr>
</tbody>
</table>

Table 10 group. T-tests (2way): D Group (Augmented)
**Overall, there exist the effects of three considering factors on the design. However, we need to be careful to interpret the results since there are also interaction effects among factors (see MANOVA and ANOVA tables; Table 0 through Table 3).**

### Scalability

Total error rate in Table 8 group shows the difference (no scalability) between simple and medium/complex, regardless of domain. So we see scalability in each domain. The results show that the augmented approach has scalability over complexity level in each domain (see total error rate in Table 7 group).

### Domain

The results show that the augmented approach is not affected and well adjusted by different domain inputs. It does not show any differences in error rates among domains (total in Table 10 group), even in each complexity level (total in Table 9 group).

### Comparison to Naïve Approach

We compared the augmented approach to the naïve approach in several ways; in each domain by complexity level (total and W/R in Table 5 group), in each complexity level regardless of domain (total in Table 6 group), and in each domain regardless of complexity level (total in Table 7 group).
APPENDIX I. ADDITIONAL PLOTS FOR PHASE 1 ANALYSIS

Augmented vs. Naïve for each domain

Figure 4.9 (Augmented vs. Naïve in general, p.66) is more specialized in each domain. HR shows same pattern as figure 4.9. But WH shows different pattern: Naïve is better than Augmented in both medium and high complexity level.
Augmented vs. Naïve for each complexity level
Various error rates in each complexity level

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Error Rate</th>
<th>Error Rate</th>
<th>Error Rate</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>0.3519</td>
<td>0.2857</td>
<td>0.2429</td>
<td>0.1389</td>
</tr>
<tr>
<td>Medium</td>
<td>0.4424</td>
<td>0.3052</td>
<td>0.2942</td>
<td>0.1481</td>
</tr>
<tr>
<td>High</td>
<td>0.4978</td>
<td>0.3846</td>
<td>0.201</td>
<td>0.1209</td>
</tr>
</tbody>
</table>
W/R error rates of each approach in each complexity level
M/P error rates of each approach in each complexity level
W/R error rates of each approach in each complex and each domain
M/P error rates of each approach in each complexity level and each domain
Questionnaire 1

Please answer the following questions drawing on your experience of creating the conceptual models.

Your Name: ____________________________

<table>
<thead>
<tr>
<th>Statement</th>
<th>Disagree</th>
<th>Slightly</th>
<th>Neither</th>
<th>Slightly</th>
<th>Quite</th>
<th>Strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual objects and relationships are too small to deal with during modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual objects and relationships are too large to deal with during modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual objects and relationships are too abstract to deal with during modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual objects and relationships are too specific to deal with during modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It was easy to generate appropriate objects and relationships.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The objects and relationships that I generated were sufficient.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It was easy to decide how the objects and relationships should be connected.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connecting objects and relationships was sufficient to generate the model.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objects and relationships enable me to perform modeling quickly.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objects and relationships improve my modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objects and relationships increase my productivity in modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objects and relationships do not enhance my modeling effectiveness.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objects and relationships make modeling easier.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objects and relationships are not useful for modeling.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using objects and relationships requires a great deal of mental effort.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using objects and relationships is often frustrating.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It is difficult to learn how to use objects and relationships.
I find it easy to get objects and relationships to do what I need to do.
I find that objects and relationships are flexible to use.
It is easy for me to become skillful in using objects and relationships.
I find objects and relationships easy to use.

**Questionnaire 2**

Please answer the following questions drawing on your experience of creating the conceptual models.

Your Name: ______________________________

<table>
<thead>
<tr>
<th></th>
<th>Disagree</th>
<th>Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis patterns are too small to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design fragments are too small to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis patterns are too large to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design fragments are too large to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis patterns are too abstract to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design fragments are too abstract to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis patterns are too specific to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design fragments are too specific to deal with during conceptual design.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>It was easy to identify analysis patterns that are relevant.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>It was easy to identify design fragments that</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The analysis patterns were sufficient to generate the conceptual model. The design fragments were sufficient to generate the conceptual model.

It was easy to decide how the analysis patterns should be combined. It was easy to decide how design fragments should be combined.

Combining analysis patterns was sufficient to generate the conceptual model. Combining design fragments was sufficient to generate the conceptual model.

Analysis patterns enable me to perform conceptual modeling quickly. Design fragments enable me to perform conceptual modeling quickly.

Analysis patterns improve my conceptual modeling. Design fragments improve my conceptual modeling.

Analysis patterns increase my productivity in conceptual modeling. Design fragments increase my productivity in conceptual modeling.

Analysis patterns do not enhance my conceptual modeling effectiveness. Design fragments do not enhance my conceptual modeling effectiveness.

Analysis patterns make conceptual modeling easier. Design fragments make conceptual modeling easier.

Analysis patterns are not useful for conceptual modeling. Design fragments are not useful for conceptual modeling.

Using analysis patterns requires a great deal of mental effort. Using design fragments requires a great deal of mental effort.

Using analysis patterns is frustrating. Using design fragments is frustrating.
<table>
<thead>
<tr>
<th>It is difficult to learn how to use analysis patterns.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>It is difficult to learn how to use design fragments.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I find it easy to get analysis patterns to do what I need to do.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I find it easy to get design fragments to do what I need to do.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I find that analysis patterns are flexible to use.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I find that design fragments are flexible to use.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>It is easy for me to become skillful in using analysis patterns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is easy for me to become skillful in using design fragments.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I find analysis patterns easy to use.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I find design fragments easy to use.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I intend to use analysis patterns whenever possible.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I intend to use design fragments whenever possible.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I intend to use analysis patterns frequently.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I intend to use design fragments frequently.</td>
</tr>
</tbody>
</table>

### DEMOGRAPHIC INFORMATION

All responses to this questionnaire are strictly confidential; only summary findings will be reported. Thank you for your assistance in this project.

Your Name: _______________________________________

Gender: Male _____ Female _____

Age: Under 25_____ 26 - 30_____ 31 - 35____ 36-40_____ 41 and over_____

Work Experience: None_____ < 5 years_____ 5 to 10 years_____ over 10 years_____ 

System design experience: None_____ < 5 years_____ 5 to 10 years_____ over 10 years_____ 

179
How knowledgeable are you about the application domain: human resource management? (1: None, 7: Very much)?

1____ 2____ 3____ 4____ 5____ 6____ 7____

How knowledgeable are you about the application domain: retail management? (1: None, 7: Very much)?

1____ 2____ 3____ 4____ 5____ 6____ 7____

How knowledgeable are you about the application domain: warehouse management? (1: None, 7: Very much)?

1____ 2____ 3____ 4____ 5____ 6____ 7____

How knowledgeable are you about conceptual modeling? (1: None, 7: Very much)?

1____ 2____ 3____ 4____ 5____ 6____ 7____

How knowledgeable are you about analysis patterns? (1: None, 7: Very much)?

1____ 2____ 3____ 4____ 5____ 6____ 7____

How knowledgeable are you about design fragments? (1: None, 7: Very much)?

1____ 2____ 3____ 4____ 5____ 6____ 7____
APPENDIX K. TASKS FOR PHASE 3 EXPERIMENT

DRAWING CONCEPTUAL MODELS

Your task is to draw conceptual models for applications in different domains.

Definition

A conceptual model contains objects and relationships in the domain of interest, specified with accepted notational conventions that help to perceive, organize, and specify the proposed system. A simple example for a small application is shown below.

Sample Requirement Statement

“The project management system will keep information on projects based on required skills and employees assigned to a specific project. It also keeps track of employees’ skills. Employees on the project are supervised by Managers.”

Sample Conceptual Model

manager manages employee works on project

has skill requires
MODE 1: UNAIDED

Draw a conceptual model for the requirements statement below.

Requirement Statement

“The retail sales management system tracks sales transactions as items are sold and adjusts inventory levels. It supports different payment modes such as cash, credit, check, and debit. In addition to the usual requirements of various accounting and management reports, it allows input and tracking of coupons, rebates, and refunds.”

How to create a conceptual model without any assistance

1. Find objects and relationships that you think are useful.
2. Connect them to generate your model.

Your Name: ________________
MODE AP: REUSING ANALYSIS PATTERNS

Draw a conceptual model for the requirements statement below with the help of analysis patterns to identify objects and relationships. The analysis patterns are available to you on the desktop at http://www.unlv.edu/faculty/than/DFpaper2/ap.html.

**Requirement Statement**

“The personnel tracking application locates personnel based on skills, keeps track of employee turnover, schedules work shifts, and computes employee benefits. In addition to the usual requirements about leave records and employee benefits, it records training sessions attended by employees as well as expense authorizations.”

**How to create a conceptual model with analysis patterns**

1. Select analysis patterns that you think are useful.
2. Apply them to your requirements – replace the object name with your object name.
3. Connect them to generate your model.

**Your Name:** ________________________
MODE DF: REUSING DESIGN FRAGMENTS

Draw a conceptual model for the requirements statement below with the help of design fragments. The design fragments suitable for the requirements are available to you on the desktop at http://www.unlv.edu/faculty/than/DFpaper2/df.html.

Requirement Statement

“The warehouse management system is responsible for assembling orders as required into one or more shipments. The warehouse contains many different stock items, which are stored in different bins and shelves. In addition to the usual requirements of recording deliveries received by the warehouse, it records any stockouts, which are tracked as backorders are filled.”

How to use

1. Select any part(s) or whole of the design fragments that you think are useful.
2. Connect or use them at different places in your model.

Your Name: __________________________
REFERENCES


mateo, CA.

Emergence of Hypertext and Problem Solving. Decision Sciences 28(4). November, 825-
850.


In T. J. Biggerstaff & A. J. Perlis (Eds.), Software Reusability, Vol. I (pp. 125-139).
ACM Press.


Conceptual Modeling, Databases, and CASE: An Integrated View of Information
Systems Development. Loucopolous and Zicari (Eds). Wiley & Sons, CA.

expert systems analysts: What do we know and what do we do? Journal of Management
Information Systems, 15(1), Summer, 9-50.


McGraw-Hill.

Sorumgard, L. S., Sindre, G., & Stokke, F. (1993). Experiences from application of a
faceted classification scheme. Selected papers from the 2nd Int’l Workshop on Software
Reusability Advances in Software (pp. 116-124). ReuseLucca, Italy.

State of the Art of ‘Reuse’ in Object-Oriented Development. (1998). The International
User Group Council Joint Project on Object-Oriented Technology. Available at

Classifications of Structured Objects. In Machine Learning: An Artificial Intelligence
Approach by Michalski, R. S., J. G. Carbonell, and T. M. Mitchell (Eds.), Vol. 2, Morgan
Kaufmann, Los Altos, CA, 471-498.


