Generating large-scale repositories of reusable artifacts for conceptual design of information systems

Taedong Han a,⁎, Sandeep Purao b, Veda C. Storey c

a Department of Management Information Systems, University of Nevada, Las Vegas, Las Vegas, NV 89154, United States
b College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802, United States
c J. Mack Robinson College of Business, Georgia State University, Atlanta, Georgia 30302-4015, United States

Available online 8 December 2007

Abstract

The design and construction of reusable artifacts is a labor-intensive and demands significant time and effort from expert designers. The up-front investment needed for constructing repositories of reusable artifacts is, therefore, often difficult to justify without immediate benefits. This research proposes a methodology, called the Domain Fragment Creator (DFC), to overcome this problem. It relies on a new type of reusable artifact, called domain fragments that can be generated by examining commonalities and variations in existing designs. The paper describes the methodology and evaluates the quality of the resulting repository using metrics such as domain coverage.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Reusable artifact; Domain fragment; Analysis pattern; Object-oriented design; Conceptual design; Knowledge reuse; Clustering; Automated reuse

1. Introduction

Software reuse involves using previously developed software artifacts for developing new software systems, instead of developing de novo. Common forms of reusable artifact include source code such as low-level utilities (e.g. sort routine), large components (e.g. shopping cart subsystem), or technology-dependent interface elements (e.g. buttons) [11,16,41,44,59]. Software developers realize that reusing artifacts at early stages of the software development life cycle (e.g. design artifacts) can yield much greater benefits than merely reusing source code. However, creating reusable artifacts that can aid the design of information systems is a daunting and labor-intensive task [28,42]. This is particularly true for artifacts that may be reused during the conceptual design of information systems [30]. Reusable design artifacts are typically represented in the form of patterns that can be instantiated and combined in different ways to produce concrete designs. Identifying these patterns, however, is not a trivial matter. They cannot be acquired simply as a by-product of the system development process. Identifying patterns requires a systematic, integrated plan and can demand significant organizational resources such as manpower and money [57]. The most widely used method to identify such patterns is domain engineering (DE) [35]. The goal of DE is to develop patterns that embody a generic solution to common problems within a specific domain by following a series of predefined steps such as...
domain analysis, domain design, and domain implementation [56]. Once an organization decides to identify reusable artifacts, the process can require the dual lifecycle of the SDLC (Software Development Life Cycle) and DE (domain engineering) [50]. Domain engineering can be as time and effort-intensive as SDLC. Cost-effective, predictable and repeatable processes are, therefore, needed to lower domain engineering efforts.

There are also published patterns [8,11,16,18] (called design patterns and analysis patterns) that are available directly from published libraries. These are often identified by respected design experts in the field, and are based on common features regardless of the domain. They provide general solutions, documented in a domain-independent format. Properly used, they can speed up the development process by providing tested, proven templates. Interestingly, because of their domain-independence, reuse them requires significant effort. They also require heavy up-front investments from domain experts or designers [25,42], and can present a steep learning curve for designers interested in reusing them.

The situation above demands continued efforts to find more cost effective strategies to facilitate reuse, particularly at the design stage. The objective of this research, therefore, is to (a) develop a methodology for automatically generating repositories of reusable artifacts for conceptual design, (b) establish its feasibility, and (c) illustrate its scalability. As a precursor to the development of the methodology, we propose a new artifact that we call ‘domain fragments.’ Domain fragments are generated by using both commonalities and variations across existing designs in a domain, which makes them amenable to automation. We leverage this in our methodology, called Domain Fragment Creator (DFC), which includes the use of clustering and graph processing algorithms. To illustrate scalability, we describe the application of the DFC methodology. To evaluate the outcomes, we describe how the domain fragments can be used for a new application design, and develop metrics to assess the quality of the resulting repository. The overall contribution of the research, therefore, is a methodology for generating repositories of reusable artifacts for conceptual design. We use an example from the warehousing domain throughout the paper to illustrate the concepts.

Motivating example: Consider Warehouses R Us, a medium-sized organization that needs to develop an application to support its warehousing operations. Upon reviewing several commercial packages, the company’s managers concluded that none of the packaged applications can meet their requirements without extensive customization. They realize, though, that a warehousing application is not a novel domain and they do not want their IT staff to start designing the application de novo. The CIO of Warehouses R Us would like to obtain a repository of designs that may have been implemented for warehousing applications. A single, grand domain model of the warehousing domain, however, is not a desirable place to start because (a) it requires a significant effort to learn the entire model [1,38], and (b) anchoring behavior [38] can lead to the use of unnecessary features. The need that the CIO expresses can be articulated as a repository of designs within the warehousing operations domain that the application designers can easily search to locate useful designs, which they can then assemble to quickly create an initial design of the desired application.

The remainder of this paper is organized as follows. Section 2 reviews prior research on creating reusable artifacts. Section 3 defines domain fragments. In Section 4, we develop the methodology for generating a repository of domain fragments. Section 5 shows how the approach can be scaled with the help of bootstrapped inputs. Section 6 illustrates the use of domain fragments, and applies metrics to assess the quality of the generated repository. Section 7 concludes with a discussion of the implications for future research and implications for practice.

2. Prior research

2.1. Reusable artifacts

A number of artifacts have been proposed for reuse during systems development including: templates [49], components [59], frameworks [15,24], analysis patterns [11,16], and design patterns [18,41]. However, their applicability to conceptual design varies greatly. Three criteria can be used to assess the effectiveness of these artifacts. The first, usability [34], measures the ease with which an artifact can support retrieval (search and adaptation of the artifact for the current design) and assembly (integration of the reusable artifact with other parts of the design) [31,46]. The second, (re)usefulness, is determined by the artifact’s granularity (the size of the artifact relative to the application in which it is reused), and abstractness (domain independence). Artifacts such as analysis patterns [11] can vary in granularity, and can defy simple characterization [7] although they are normally smaller than frameworks, which tend to be coarse as well as domain-specific [51]. A third important criterion, efficiency, is the effort necessary to construct the reusable
artifact [25,34,42,57]. Most approaches to creating reusable artifacts [20,28,42,57] tend to be effort-intensive, requiring input from human experts to identify elements with potential for reuse, and convert these into reusable elements [28]. It is the third criterion, efficiency, that provides the most daunting obstacle to successful reuse.

2.2. Constructing reusable artifacts

One strategy to reduce the effort required to construct reusable artifacts involves examining commonalities and variations in existing applications. The argument is analogous to identifying common components that can function across a product line [14,17,21,27,53,60,62]. The strategy has been elusive in information systems, because it is exceedingly difficult to automate the identification of commonalities across applications.¹ Software products in a given domain do, in fact, contain such commonalities, which can constitute as much as 60% to 70% of an application [33]. Techniques are also available for analyzing conceptual schemata that can be adapted for clustering and abstracting (e.g. [5,9,10]). Maarek et al., [29], and Castano et al., [10] propose techniques for finding similarities among schemata. They also devise techniques for identifying commonalities among legacy applications. For example, they propose methods for coalescing prior research into techniques for analyzing conceptual schemata and identifying ‘reference components’.

Together, these research efforts provide useful starting points for our own research. First, drawing on the notion of common components that can function across a product line [14], we propose that a new form of reusable artifact be created. Second, the archaeological techniques [10,29] allow leveraging such commonalities, we extend these to account for variations. The next two sections address these tasks. First, we propose a new reusable artifact that can reflect both common and variable portions of applications. Next, we develop a methodology to systematically isolate these commonalities and variations into such reusable artifacts, to address the important criterion of efficiency identified above.

3. Domain fragments

We propose a new reusable artifact, called a domain fragment. A domain fragment is a proper subset of the

¹ Existing reusable artifacts (e.g. analysis and design patterns [11,16,18]), do, in fact, represent such commonalities although discovered informally by expert designers drawing on their prior experiences.

---

[Diagram of company and employee relationships with agreement and clause connections]
objects and relationships. The difference between them lies in *granularity*. A domain architecture includes (and may tempt the designers to reuse [38]) a large set of objects and relationships in a domain. Domain fragments, on the other hand, provide proper subsets that can cover different parts of a domain. A designer can account for variations in a domain by combining these fragments in different ways. Unlike domain architectures, which encourage *homogeneity* across applications in a domain, domain fragments provide the designer with the ability to recognize such *variations*. These distinctions suggest that ‘domain fragments,’ as a reusable artifact may be more *usable* and more (re)useful than domain architectures.

The claims related to the benefits of domain fragments compared to analysis patterns and domain architectures, however, are *not* the focus of investigation in this paper. Instead, we focus on a key prerequisite for any possible empirical assessment of such claims: establishing feasibility of constructing domain fragments, i.e., addressing the criterion of *efficiency*. The key question, therefore, is this: ‘Is it possible to generate a large-scale repository of domain fragments without requiring significant effort from expert developers?’ The first step towards answering this question is to develop a methodology for creating domain fragments.

4. DFC: A methodology for constructing domain fragments

The methodology, which we simply call Domain Fragment Creator (DFC), requires two inputs: (a) requirements statements for applications in a domain, and (b) conceptual designs corresponding to these requirements statements. These can be obtained from prior research in automated conceptual design (Purao and Storey [47], Purao et al. [48], Wohed [64]) that allows synthesizing designs from requirements statements. The DFC methodology contains three phases. Phase 1 identifies domain fragments by examining commonalities and variations across available designs. Phase 2 analyzes requirements statements to form requirements clusters. These two phases can occur in parallel. Phase 3 then establishes a mapping between domain fragments and requirements clusters so designers can use them to search through the set of domain fragments generated. Fig. 2 outlines the methodology which we develop in this section. The description is accompanied by a running example which illustrates how each phase will be carried out for the warehousing domain. Each sub-section corresponds to a phase in the methodology. The notation (Fig. 3) captures the constructs used in the development of the methodology, and assists in the description.

4.1. Phase 1: identifying domain fragments

To identify commonalities and variations in the input designs, we propose a procedure that uses iterative partitioning [55]. The procedure examines designs available at each pass to identify the maximal ‘intersection’ [4]. The maximal intersection is operationalized as analysis patterns used and instantiated in the set of available designs. Based on this information, it partitions the set into two subsets; one includes designs that contain the maximal intersection; the other includes designs that do not. The procedure continues recursively until no such intersection is found in a threshold number of designs. The procedure thus creates a binary tree, where each node is marked by the maximal intersection in designs available for consideration in that step. The construction of a domain fragment is then accomplished by pre-order traversal [40] of the binary tree via its different paths, and to different depths. The path traced by each pre-order traversal is used to concatenate the set of patterns \{pattern\}_i at each node visited. Commonalities in designs are, therefore, captured by nodes that are higher in the tree. The paths available from each node and the possibility of stopping at different depths allows for variations.

*Example:* For the warehousing domain, the inputs may contain a number designs (e.g. \(D=30\)). The designs are parsed to identify instantiated analysis patterns...
A binary tree is created, starting with a root node, which is marked by the pattern that is shared by most designs (e.g. pattern $i = 'group-member'$ instantiated as 'bin-item'). The designs are separated into two sets, one containing this pattern, and the other not. Each set is examined to identify the pattern that is shared by most designs (e.g. pattern $j = 'transaction-subsequentTransaction'$ instantiated as 'order-delivery'), and split again. The resulting tree is then traversed starting with the root node (i.e. pattern $= 'group-member'$) to construct different pattern-sets. A particular set $\{pattern_i\}$ might include 'group-member' instantiated as 'bin-item' and 'transaction-subsequentTransaction' instantiated as 'order-delivery').

Appendix A shows the procedure which includes two sub-procedures: one for creating a binary tree, and another for constructing domain fragments by traversing the tree. As shown, the latter procedure makes the assumption that a leaf node must be reached. This assumption may be relaxed, allowing the procedure to define domain fragments at varying depths. The domain fragments constructed in this manner, however, need to be associated with features of the application domain. These ‘search criteria’ are identified in the next phase of the methodology by analyzing the requirements statements.

4.2. Phase 2: identifying requirements clusters

To identify requirements clusters, we adapt and extend the concept of a vector space model (VSM) [54] (from automated document processing) that is used to cluster similar documents by computing ‘term vectors.’ We identify requirements clusters, which are analogous to these ‘term vectors’ across sets of requirements statements to capture commonalities among requirements. The procedure does not require a specific format for the requirement so requirements stated as simple natural language assertions are sufficient. Our three-step procedure for creating requirement clusters includes: (a) identifying keywords in each requirements statement, (b) computing keyword-pair co-occurrence, and (c) creating requirements clusters using keyword-pair co-occurrences as centroids.

The first step, identifying keywords, ensures that the clustering algorithm utilizes important words in the requirements statements. This requires application of two filters. The first parses each requirements statement, stmtc, to remove stopwords [58]. The second filter computes relative frequency of the remaining words across all requirements statements. Significant words with higher than a threshold 3 importance, weights, are promoted to the status of a ‘keyword.’

The second step, computing keyword pair co-occurrence, calculates a measure of keyword co-occurrence across requirements statements. We adopt the cosine algorithm from Salton [54] for this purpose. The co-occurrence measure between each pair of keywords is used as centroids during the clustering process.

The third step, generating requirements clusters, uses the centroids created in the previous step to separate requirements statements into clusters using an iterative procedure. The procedure starts with the keyword pair that has the highest co-occurrence measure. Treating this pair as the seed, all keyword pairs connected to the seed pair (up to a threshold 4 keyword co-occurrence value) are located. This ensures that the complete linkage criterion [54] in the cluster. The procedure repeats, creating successive clusters until the co-occurrence level reaches a halt threshold 5. Each requirements statement

---

3 Minimum relative frequency for promoting a word to a keyword.
4 Threshold for adding keyword-pairs into a requirements cluster.
5 Halt threshold for stopping the requirements cluster generation process.
can, therefore, contribute to the definition of one or more requirements clusters.

**Example:** For the warehousing domain, the inputs may contain a number of requirements statement (e.g. \( S = 30 \)). The statements are parsed to identify keywords, and the frequency of co-occurrence for each pair of keywords is computed (e.g. keyword \(_e\) and keyword \(_f\) = product + delivery may occur 22 times, aisle + shipment may occur 15 times, and product + on-hand may occur 7 times). Starting with the most frequently occurring keyword-pair (e.g. product + delivery), cluster \(_m\) is constructed by incrementally adding connected keywords (e.g. the keyword on-hand may be added because it is connected with the keyword product) until a threshold is reached (e.g. co-occurrence frequency of at least 5). The procedure repeats, starting with the next most frequently co-occurring pair (e.g. product + shipment).

An example of a requirements cluster that may result from this procedure is a set of keywords e.g. \{product, delivery, on-hand, inventory, location\}.

Appendix A shows the computations for promoting words to keywords, and the keyword co-occurrence measure. The requirements clusters identified in this manner represent features in the domain (i.e. possible search criteria) that need to be associated with domain fragments generated in the first phase. The next phase of the methodology performs this mapping.

### 4.3. Phase 3: mapping requirements clusters to domain fragments

The final phase maps requirements clusters to domain fragments. The mapping is mediated by the requirements statements and the corresponding designs. The mapping serves as an index that future designers can use to retrieve domain fragments from the repository.

**Example:** For the warehousing domain, the first phase may have produced a number of domain fragments (e.g. fragment\(_k\) may include a particular set \{pattern\(_i\)\} = ‘group-member’ instantiated as ‘bin-item’ and ‘transaction-subsequentTransaction’ instantiated as ‘order-delivery’). The second phase may have produced a number of requirements clusters (e.g. cluster\(_m\) may include a set of keywords \{keyword\(_e\)\} = product, delivery, on-hand, inventory, location). Using the requirements statements and designs as mediators, fragment\(_k\) may be mapped against cluster\(_m\).

Appendix A outlines the algorithm to perform this mapping. The methodology outlined so far describes how domain fragments may be constructed and mapped against requirements clusters for later retrieval. A significant concern, however, remains. To be useful, the DFC methodology must be scalable, to allow generation of large-scale repositories of domain fragments. Obstacles to scalability include: (a) non-availability of inputs, (b) determination of appropriate threshold values, and (c) efficiency of algorithms. The next section investigates these concerns.

### 5. Applying DFC to generate a large-scale repository of domain fragments

The most significant obstacle to applying DFC is the non-availability of appropriate inputs because it requires: (a) identification of a large set of requirements statements and applications in a domain, and (b) comparable representation of their conceptual designs. Few organizations are willing to share this valuable information. Even if their reluctance can be overcome, it is difficult to represent the designs using a common language. To overcome this obstacle, we propose to bootstrap [63] additional keyword subsets as surrogates for requirements statements [22,47,48,64]. Using these, we build conceptual designs using analysis patterns following approaches suggested by [46,47]. The requirements statements and conceptual designs, then, provide the inputs necessary to generate domain fragments and requirements clusters. Fig. 4 shows the process. We continue the example from the warehousing management domain to illustrate the outcomes.

**Example:** The keyword pool for the warehousing domain was constructed by identifying terms from...
ontologies, supplemented by those extracted from web sites of: (a) organizations in this domain, and (b) prior research [37]; and finally amplified with synonyms from a thesaurus. 37 terms were identified and amplified by 31 synonyms resulting in a pool of 68 total keywords.

Original keywords ($n=37$)
- 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

Synonyms ($n=31$)
- 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

Subsets of keywords were randomly selected from this pool. The size of each subset varied between 4 and 11 keywords following prior research [48]. The subsets generated (up to 10,000) represented an insignificant fraction of the possible subsets from the keyword pool i.e. nC68, n=4..11, i.e. 1.88157E+12. Each keyword subset was used to build a conceptual design. The resulting designs contained between 8 and 30 objects, and between 5 and 39 relationships.

The second obstacle for application of the DFC methodology was determining appropriate threshold values. Table 1 below shows initial threshold values based on results from smaller experiments.

Table 1
Initial threshold values for applying the DFC methodology

<table>
<thead>
<tr>
<th>Phase</th>
<th>Threshold</th>
<th>Initial values</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary tree creation in Phase 1</td>
<td>For halting binary tree creation</td>
<td>5 to 20 designs</td>
<td>Allow most designs to participate in the tree</td>
</tr>
<tr>
<td>Generating requirements clusters in Phase 2</td>
<td>Relative frequency for promoting to keyword</td>
<td>Median±5</td>
<td>Presence in greater than median number of requirements statements</td>
</tr>
<tr>
<td></td>
<td>For stopping the clustering process</td>
<td>Median±5</td>
<td>Clustering pairs higher than median co-occurrence</td>
</tr>
<tr>
<td></td>
<td>For adding keyword-pairs to cluster</td>
<td>(50±20) % coverage</td>
<td>A band of 20% around 50% coverage, i.e. clusters cover at least half the requirements statements</td>
</tr>
</tbody>
</table>

These threshold values were subjected to significant variations during application of the methodology. The threshold values were revised during application of the DFC methodology. We describe below the results obtained for each phase, following the warehousing domain example, based on 1000 to 6000 bootstrapped inputs. The revised threshold values and our rationale for the revisions are explained within the context of the example.

5.1. Applying phase 1 of DFC: identifying domain fragments

The first phase was executed with minimal designer intervention. A low initial threshold value (5) to halt binary tree creation ensured that the clustering process will continue, until few designs remain.

Example: Using the inputs generated for the warehousing domain, a number of simulation runs were conducted, varying the inputs from 1000 to 6000. The number of domain fragments generated with these inputs varied from 30 to 600 based on the number of inputs and varied threshold values. Table 2 shows the results obtained.

As seen in Table 2, the threshold value was systematically revised upwards to 20 to examine the effects. Each cell in the table shows the number of domain fragments generated during a simulation for the number of input designs (left column) using a certain halt threshold (top row). The results indicate that there was a negative correlation between the halt threshold and the number of domain fragments. A lower threshold value resulted in a greater number of domain fragments. One possible undesirable consequence of this is that a search may result in an unduly large number of possible domain fragments. Deciding the appropriate threshold value for

Table 2
Number of domain fragments generated

<table>
<thead>
<tr>
<th>Input designs</th>
<th>Halt threshold for tree creation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1000</td>
<td>107</td>
</tr>
<tr>
<td>2000</td>
<td>211</td>
</tr>
<tr>
<td>3000</td>
<td>318</td>
</tr>
<tr>
<td>4000</td>
<td>416</td>
</tr>
<tr>
<td>5000</td>
<td>507</td>
</tr>
<tr>
<td>6000</td>
<td>604</td>
</tr>
</tbody>
</table>
this phase, therefore, requires taking into account the next two phases, where requirements clusters are created and mapped against domain fragments.

### 5.2. Applying phase 2 of DFC: identifying requirements clusters

This phase was also executed with minimal designer intervention, varying the number of inputs between 1000 and 6000. Initial threshold values used for this phase appear in Table 1.

**Example:** Using the inputs generated for the warehousing domain, a number of simulation runs were conducted. The number of requirements clusters generated varied from just 18 to more than 100,000 depending upon the number of inputs and the threshold values. Table 3 shows the results obtained.

The keyword coverage threshold was varied systematically between 30% and 70%. Table 3 shows the number of requirements clusters generated during a simulation run for the number of input keyword sets (left column) using a certain requirements coverage threshold (top row). A significant negative correlation was observed between the number of requirements clusters and the requirements coverage threshold. Varying the threshold from 30% to 70% resulted in a rapid drop of requirements clusters (e.g., from 7,103 to 18 in row 1, Table 3). The number of requirements clusters generated, therefore, demonstrates a trade-off. A large number of clusters (corresponding to lower threshold values) provides several possibilities for searching domain fragments. Fewer clusters (corresponding to higher threshold values) provide possibly more precise search criteria, but also the threat that no criteria may be available to locate appropriate domain fragments. To help make this trade-off, one further analysis was performed (see Table 4).

At the 60% threshold, a large fraction of the requirements statements (38 to 56%) are not associated with any clusters, compared to just a few (12 to 18%) at a threshold value of 50%. Lower threshold values do not necessarily improve this coverage, but sacrifice precision. For the keywords bootstrapped for this exercise, a threshold of 50% is, therefore, desirable as it generates a large number of high quality requirements clusters.

### 5.3. Applying phase 3 of DFC: mapping requirement clusters to domain fragments

The last phase of the methodology was executed to establish a mapping between domain fragments and requirements clusters without any designer intervention. No further threshold values are required for this mapping.

**Example:** Mapping the domain fragments and requirements clusters created in the previous two phases resulted in a repository of design fragments for the warehousing domain. Fig. 5 describes the contents of the repository for one specific run based on 4000 inputs. This repository contains 416 domain fragments that can be retrieved with the help of 13,265 requirements clusters.

Further analysis of the repository, following multiple simulation runs, is shown in Table 5. It shows the results (average number of domain fragments mapped to each requirements cluster) as well as the overall results (mapping across conceptual designs and domain fragments). For example, the first row shows that, with 3000 input designs utilized for generation of the domain fragment repository, the designer for a new application would expect to be provided with $5.93 \times 2.18 \approx 13$ domain fragments.

<table>
<thead>
<tr>
<th>Input keyword sets</th>
<th>30% Coverage</th>
<th>40% Coverage</th>
<th>50% Coverage</th>
<th>60% Coverage</th>
<th>70% Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1069</td>
<td>165</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>4245</td>
<td>650</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>8149</td>
<td>1190</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>13,265</td>
<td>2165</td>
<td>132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>19,767</td>
<td>3214</td>
<td>238</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6000</td>
<td>26,592</td>
<td>4397</td>
<td>359</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* First threshold held constant at co-occurrence median.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Number of requirements statements not associated with any requirements cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Threshold 30%</td>
</tr>
<tr>
<td>3000</td>
<td>493 (16.43%)</td>
</tr>
<tr>
<td>4000</td>
<td>591 (14.78%)</td>
</tr>
<tr>
<td>5000</td>
<td>640 (12.80%)</td>
</tr>
<tr>
<td>6000</td>
<td>809 (13.48%)</td>
</tr>
</tbody>
</table>
The simulation runs, and analyses reported above suggested threshold values for applying the DFC methodology that are summarized in Table 6 below along with the rationale for selecting the value.

The results demonstrate that the three obstacles identified could be overcome: (a) inputs necessary for the DFC methodology can be bootstrapped, (b) threshold values for applying the DFC methodology can be determined, and (c) algorithms contained in DFC are scalable. As a result, the methodology can be executed with minimal demands on expert designers, which directly addresses the key concern of efficiency. The results illustrate that it is, indeed, possible to generate such large-scale repositories of reusable artifacts without significant up-front investment. The next section illustrates the use of the repository for design of new applications, and analyzes the quality of the repository using metrics devised for this purpose.

6. Evaluation and illustration

To assess the quality of domain fragments generated, we devise two simple metrics: (a) domain coverage: whether the domain fragments, in the aggregate, provide adequate coverage of the domain, and (b) access paths: whether the repository provides sufficient access paths to reach a given domain fragment.

The potential space of any domain (e.g. warehouse management or human resource or accounting or order management [19]) is likely to be quite large and subject to considerable debate. A comprehensive elaboration of the constructs in a domain is also likely to lead to debates related to whether certain constructs are within or outside the scope. Our proposed metric for domain coverage, therefore, does not attempt to measure the quality of the domain fragment repository against this unattainable ideal. Batra [7], for example, describes the possibility that a small number of templates may be sufficient to provide domain coverage. Instead, we suggest a metric that directly evaluates the quality of the process that has lead to these domain fragments. In particular, the domain coverage metric assesses whether the requirements statements that were provided as inputs to the DFC methodology are reflected in the resulting domain fragments. Table 5, for example, shows that less than 15% of the requirements statements are not directly associated with any requirements clusters. Clearly, a lower number suggests more effective domain coverage. The metric can be further improved by adjusting values of parameters during application of the DFC methodology. These adjustments, however, may come at the cost of other qualities of the domain fragments (e.g. too many fragments) as described during application of the methodology. The repository, generated using the parameter choices described above, therefore, does provide adequate coverage of the domain.

The second metric involves access paths that allow a designer to search for and retrieve domain fragments available in the repository. Providing a single access path to a domain fragment can be problematic because that

<table>
<thead>
<tr>
<th>Inputs</th>
<th># requirements → requirements cluster</th>
<th># requirement clusters → requirement</th>
<th># designs → domain fragment</th>
<th># domain fragments → designs</th>
<th># domain fragments → requirements cluster</th>
<th># requirements cluster → domain fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>2.19</td>
<td>5.93</td>
<td>9.40</td>
<td>0.9963</td>
<td>2.18</td>
<td>55.74</td>
</tr>
<tr>
<td>4000</td>
<td>2.21</td>
<td>7.35</td>
<td>9.58</td>
<td>0.9968</td>
<td>2.20</td>
<td>70.41</td>
</tr>
<tr>
<td>5000</td>
<td>2.22</td>
<td>9.77</td>
<td>10.04</td>
<td>0.9978</td>
<td>2.22</td>
<td>88.05</td>
</tr>
<tr>
<td>6000</td>
<td>2.24</td>
<td>9.92</td>
<td>9.91</td>
<td>0.9973</td>
<td>2.23</td>
<td>98.31</td>
</tr>
</tbody>
</table>

Fig. 5. Repository of reusable domain fragments for the warehousing domain.
fragment may be useful in a number of design situations. The search criteria that can lead to a domain fragment, therefore, need to account for the multitude of configurations that may be encountered for different variations in a domain. As a result, our proposed metric values the multiplicity of access paths to retrieve a given domain fragment. Fig. 5, which characterizes the repository generated, shows that the available domain fragments (416) can be searched using a large number of search criteria (13265); on average by about 32 access paths.

The two metrics, domain coverage and access paths provide a preliminary indication of the quality of repository. Additional measures such as perceived ease of use and usefulness are possible (e.g. following the technology acceptance model [13]), but have not been included in the current manuscript because they require perceptual measures based on potential deployment. The final arbiter of the value of the domain fragments is clearly whether they will benefit the conceptual design of systems in organizations. A validation of this nature will, necessarily, require an approach similar to an integrated design research and action research approach outlined by [12], which is beyond the scope of this research.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Threshold</th>
<th>Initial values</th>
<th>Suggested value</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary tree creation in Phase 1</td>
<td>For halting binary tree creation</td>
<td>5–20 designs</td>
<td>5 designs</td>
<td>Allow most designs to participate in the tree</td>
</tr>
<tr>
<td>Generating requirements clusters in Phase 2</td>
<td>Relative frequency for promoting to keyword</td>
<td>Median±5</td>
<td>Median</td>
<td>Presence in greater than median number of requirements statements</td>
</tr>
<tr>
<td></td>
<td>For stopping the clustering process</td>
<td>Median±5</td>
<td>Median</td>
<td>Clustering pairs higher than median co-occurrence</td>
</tr>
<tr>
<td></td>
<td>For adding keyword-pairs to cluster</td>
<td>(50±20)% coverage</td>
<td>50% coverage</td>
<td>The clusters should cover at least half the requirements statements</td>
</tr>
</tbody>
</table>

Fig. 6. Domain fragments retrieved using keywords in the requirements statements.
6.1. An illustration

In lieu of an organizational evaluation, we provide an example of use of domain fragments. The illustration serves as a scenario of use as suggested by Hevner et al., [23]. To do this, we return to the motivating example followed throughout the paper.

Example: The initial, partial, specification of the requirements of an application for Warehouses R Us includes the following. “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.”

To construct a design for this requirements statement, the designer first searches the repository for domain fragments based on keywords in the requirements. Fig. 6 shows the fragments retrieved.

The figure does not show cardinalities of relationships. Class names in bold indicate classes included in analysis patterns that were not instantiated for the warehousing domain. The fragments retrieved contain several overlapping objects. For example, the objects “warehouse-warehouseLineItem” are repeated across Fragments 1, 3 and 4. To create the combined design, the designer needs to exercise judgment to: (a) exclude any unnecessary relationships, (b) exclude any duplicated objects, and (c) rename objects as necessary. The combination requires identification of common classes so that they can be superimposed, a step that may be automated partially.

Fig. 7 shows the outcome of this step: an initial design for the application. The designer may refine this design further based on prior knowledge of the domain or other experience. Constructing the initial design however, does not require such expertise and may also be generated by a novice designer by reusing the domain fragments. The example demonstrates that prima facie, the domain fragments are useful for providing designers, both experts and novices alike, a quick first approximation of a conceptual design for the new design situation.

7. Discussion

Traditionally, conceptual design of information systems has been considered a highly creative activity, where human developers are indispensable. Perhaps as a result of the scarcity of expert developers, practitioners report that the difficult task of conceptual modeling often falls into disuse within their organizations [61]. Building on work towards greater understanding of developer behaviors, such as the use of schemata [43], recurring structures [6] and codification of knowledge in terms of patterns [11,16,18], researchers are generating approaches that can help developers during the conceptual design process [29,45], thereby reducing the dependence on experienced developers. Building a repository of reusable domain fragments repository involves explication and codification of human developers’ knowledge — a
major obstacle in facilitating reuse of knowledge [3,26,32,37].

In response to this broad set of concerns, we have proposed a new artifact, called a domain fragment, that is appropriate for reuse during the conceptual design phase of the system development process. We have provided a definition of the artifact, and described a methodology for generating a repository of domain fragments. The research has shown that it is possible to identify domain fragments by examining commonalities and variations across applications in a domain, and that domain fragments, identified in this manner, are likely to be useful for conceptual design. First, they are easier to retrieve because they contain a search mechanism in the form of requirements clusters. Second, they are coarser than individual objects; yet not as large as domain models. Third, they are domain-specific, and therefore, easier to understand and reuse. Finally, minimal investment of effort is needed from expert designers to generate a repository of domain fragments, following the DFC methodology.

The decisive test of the utility for domain fragments will, of course, be whether systems designers will reuse them, and whether they will contribute to superior outcomes from the system design process. A convincing answer to this question will require an empirical investigation of whether and how domain fragments contribute to the conceptual design process in organizational contexts. This paper has laid the groundwork for future empirical studies. Our arguments suggest two specific claims may be investigated in empirical investigation. First, our work suggests that domain fragments are likely to require minimal effort from the designers to understand and reuse. Second, the repository includes a retrieval mechanism that makes it easier to identify appropriate domain fragments, given a simple description of requirements. A counter-example for both claims is analysis patterns, which are domain-independent, and do not provide a retrieval mechanism.

7.1. Contribution

The key contributions of this research are, therefore, two-fold. First, we have defined domain fragments; and have outlined a methodology for creating them. The methodology leverages the semantics and structure embedded in existing reusable artifacts such as analysis patterns. The mechanisms and procedures build upon established algorithms such as document clustering and graph-structure manipulation. A chronic problem that plagues the software industry is the absence of approaches to find and adapt appropriate components for users’ needs. To solve this problem, standards for components specification such as Reusable Asset Specification (RAS) [52] are being developed. This research suggests a possible extension to these specifications in the form of automated techniques for identifying and indexing reusable artifacts.

Second, we have demonstrated how the methodology can be scaled to generate large repositories of domain fragments. At a pragmatic level, this addresses a critical need, namely, creating a reusable artifact for conceptual design without requiring significant effort from expert developers. While it may be difficult to provide a benchmark in terms of costs for our approach, we argue that of the two elements of costs, computational and human, the DFC methodology directly deals with the latter, which is a greater contributor to total costs. Our approach can be applied in a CASE tool, so that novice designers can easily generate a conceptual model just by providing requirements statements such as use cases. Prior studies of reusable artifacts have focused on in-house approaches to reuse during this stage of the development process (e.g. [9,65]) with consequent limitations imposed by the small number of designs within a domain. To the best of our knowledge, no CASE tool contains a mechanism to identify reusable artifacts and create their repository at design stage. We have overcome this limitation by suggesting a way to automatically bootstrap a large number of inputs that may be used by the DFC methodology to build large-scale repositories of reusable domain fragments. Automatically generating such a repository overcomes a major problem of inefficient manual approaches that rely on domain experts and experienced designers, who are scarce. To ensure that other researchers and practitioners can benefit from and build on research reported in this paper, we plan to make our approach, algorithms, and implementation available similar to other initiatives such as the Pattern Book Open Source [39].

As with all research, the outcomes must be viewed with some caveats, which represent opportunities for future research. First, the implementation used to demonstrate the application of the DFC methodology may be extended to test additional ranges and combinations of parameters. Second, the bootstrapped requirements statements and conceptual designs are restricted to lexicons and published sources following prior research, and therefore, may be augmented with keywords extracted from real-world examples of
applications. The reported results are likely to improve further with these additions. Finally, our assessments of the outcomes have included demonstration of examples and assessment of the quality of the repository with metrics such as domain coverage and access paths. These may be extended with other possibilities such as a panel of expert developers as judges, empirical analyses following constructs such as usefulness, and integrated design research/action research approaches. These evaluation possibilities remain part of our future research agenda.

Acknowledgement

An earlier version of this research was presented at the International Conference on Conceptual Modeling. This research was supported by the J. Mack Robinson College of Business, Georgia State University.

Appendix A. Notations and algorithms

Notations

\begin{itemize}
  \item \textbf{Requirements statements}
    \begin{itemize}
      \item word_{a,b} \in W \quad \text{set of significant words, } a,b \in \ldots W
      \item stmt_{c,d} \in S \quad \text{set of requirements statements, } c,d \in \ldots S
      \item word_{c,d} \subseteq stmt_{c,d} \quad \text{significant word } w_{a} \text{ is part of requirements statement } stmt_{c,d}
      \item keyword_{c,d} \in KW \quad \text{keywords in a domain, } KW \subseteq W; e_{f} \subseteq 1\ldots KW
      \item member_{c,d} \quad \text{keyword is part of requirements statements } stmt_{c,d}, \text{ member}_{c,d} = [0,1]
    \end{itemize}
  \item \textbf{Designs}
    \begin{itemize}
      \item design_{g,h} \in D \quad \text{set of designs, } g,h \in \ldots D
      \item pattern_{i,j} \in P \quad \text{set of analysis patterns instantiated to create the designs, } i,j \in \ldots P
    \end{itemize}
  \item \textbf{Domain fragments}
    \begin{itemize}
      \item fragment_{k,l} \in DF \quad \text{set of domain fragments, } k,l \in \ldots DF
    \end{itemize}
  \item \textbf{Requirements clusters}
    \begin{itemize}
      \item cluster_{m,n} \in V \quad \text{set of requirements clusters, } m,n \in \ldots V
    \end{itemize}
\end{itemize}

Algorithm for phase 1: Identifying and constructing domain fragments

\begin{enumerate}
  \item Let \text{PreviousNodeSet} = \emptyset
  \item Let \text{CurrentNodeSet} = D
  \item Start Procedure: \textit{Augment binary tree}
    \begin{itemize}
      \item Let \text{NodeSet}_{(with)} = \text{NodeSet}_{(without)} = \emptyset
      \item \forall \text{design}_{g,h} \in \text{CurrentNodeSet}
        \begin{itemize}
          \item // Find patterns most common to these designs
            \item Identify maximal set \{pattern_{i}\} such that
              \begin{itemize}
                \item pattern_{i} \in \text{design}_{g} \cap \text{design}_{h}
              \end{itemize}
            \item If \text{PreviousNodeSet} \neq \emptyset
              \begin{itemize}
                \item pattern_{i} \in \text{PreviousNodeSet}, \{pattern_{i}\} \wedge
                  \exists \text{pattern}_{j} \in \text{PreviousNodeSet}. \{pattern_{j}\}
              \end{itemize}
          \item Endif
        \end{itemize}
        \item Mark \text{CurrentNodeSet} with \{pattern_{i}\}
    \end{itemize}
  \item \forall \text{design}_{g,h} \in \text{CurrentNodeSet}
    \begin{itemize}
      \item // Divide the set of designs into two subsets based on the patterns
        \item Mark current node with \{pattern_{i}\}
        \item If \{pattern_{i}\} \subseteq \text{design}_{g}
          \begin{itemize}
            \item Let \text{NodeSet}_{(with,1)} = \text{NodeSet}_{(with,1)} \cup \text{design}_{g}
          \end{itemize}
        \item Else
    \end{itemize}
\end{enumerate}
Let $\text{NodeSet}(\text{without},2) = \text{NodeSet}(\text{without},2) \cup \text{design}_g$

Endif

If $|\text{NodeSet}(\text{with})| > \text{Halt threshold}$
Let $\text{PreviousNodeSet} = \text{CurrentNodeSet}$
Let $\text{CurrentNodeSet} = \text{NodeSet}(\text{with})$
Execute Procedure ‘Augment binary tree’

Endif

If $|\text{NodeSet}(\text{without})| > \text{Halt threshold}$
Let $\text{PreviousNodeSet} = \text{CurrentNodeSet}$
Let $\text{CurrentNodeSet} = \text{NodeSet}(\text{without})$
Execute Procedure ‘Augment binary tree’

Endif

End Procedure

Let $x=0$

Start Procedure: Create domain fragment
Perform pre-order traversal of the binary tree generated above
If $|\text{NodeSet}(x,1)| > \text{designer-determined threshold}$
Let $\text{CurrentDomainFragment}_{x} = \text{CurrentDomainFragment}_{x} \cup \text{CurrentNodeSet}. \{\text{pattern}_j\} = \text{NodeSet}(x,2)$
Else
Let $x=x+1$
Let $\text{CurrentDomainFragment}_{x} = \emptyset$
Endif

End Procedure

Computations for phase 2:

Promoting a word to a keyword

$\text{weigh}_{ac} \text{ in stmt}_c = |	ext{Frequency(word}_a) \text{ where word}_a \subset \text{stmt}_c| * \log(|\text{S}|/|\text{Frequency(stmt}_c \in \text{S} |)$, where $\text{word}_a \subset \text{stmt}_c$)

Note: the computation follows the rationale described in [52].

Computing keyword affinity

$$\text{Keyword affinity} = \frac{\sum_{c=1}^{S} \text{weigh}_{ec} \times \text{weigh}_{fd}}{\sqrt{\sum_{c=1}^{S} \text{weigh}_{ec}^2 \times \sum_{d=1}^{S} \text{weigh}_{fd}^2}}$$

Algorithm for phase 3: Mapping requirements clusters to domain fragments

Start Procedure: Map domain fragments to requirement clusters

∀ fragment$_k$, fragment$_l$$ \in \text{DF} $
If fragment$_k \subset \text{design}_g$
Let DesignSet += design$_g$
∀ design$_g$, design$_h$$ \in \text{S}$
If design$_g$ = stmt$_c$
Let StmtSet += stmt$_c$
∀ stmt$_c$, stmt$_d$$ \in \text{StmtSet}$
If cluster$_m$ = stmt$_c$
Let fragment$_k \leftrightarrow \text{cluster}_m$
Endif

Endif

End Procedure
References


[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.


Taedong Han is an Assistant Professor of Management Information Systems at the University of Nevada Las Vegas. He received his B.S. in Business Administration from Yonsei University, Korea, M.S. in MIS from University of Arizona, and Ph.D. in Computer Information Systems from Georgia State University. His research interests include software reuse, object-oriented systems development, web data mining, electronic commerce, and database management. His research has been published in Information Systems Research, Journal of Management Information Systems, and Journal of Convention and Event Tourism, and presented in IS conferences such as WITS, ER, HICSS, AMCIS, and ITCC.

Sandeep Purao’s research deals with the design, evolution and management of complex techno-organizational systems, blending research methods from social science and software engineering. His research has been published in several journals including ISR, CACM, DSS, ACM Computing Surveys, and presented in conference including ICIS, WITS, ER and HICSS. After completing his PhD in Management Information Systems from the University of Wisconsin-Milwaukee, he was on the faculty at the Computer Information Systems department at Georgia State University, before moving to the inter-disciplinary College of Information Sciences and Technology at Penn State University. He currently serves as the Graduate Student Adviser at the College.

Veda C. Storey has research interests in database management systems, intelligent systems, and Semantic Web, and ontology development. Her research has been published in various journals including ACM Transactions on Database Systems, IEEE Transactions on Knowledge and Data Engineering, Information Systems Research, Management Information Systems Quarterly, and Decision Support Systems. She has served on the editorial board of several journals including Information Systems Research, MIS Quarterly, and Decision Support Systems.