The Importance of Retrieval in Creative Design Analogies
Paulo Gomes, Nuno Seco, Francisco C. Pereira, Paulo Paiva, Paulo Carreiro
José L. Ferreira and Carlos Bento
CISUC - Centro de Informática e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informática - Polo II
Universidade de Coimbra - Portugal
pgomes@dei.uc.pt

Abstract
Analogy is an important reasoning process in creative design. It enables the generation of new design artifacts using ideas from domains semantically distant. The retrieval phase is crucial to the generation of creative analogies, without a good set of candidate sources, the success of subsequent phases can be compromised. Two main types of retrieval have been identified: semantic-based retrieval and structural-based retrieval. This paper presents an empirical study on the importance of the analogy retrieval strategy in the domain of software design. We argue that both types of retrieval are important, but they play different roles in the process.

1 Introduction
Creativity is a cognitive process that generates a product said to be creative when satisfying certain kinds of properties [Dasgupta, 1994]. There have been identified at least four components of creativity [Brown, 1989]: creative process, creative product, creative person or entity, and creative situation. Within this model the crucial component is the creative product. Creativity has no meaning except in relation to the creative product. Thus an important issue in developing computational models of creativity is the evaluation of the creative product. Research done in this domain has come up with several methods used for generation of artifacts considered creative [Partridge and Rowe, 1994; Gero, 1994b; Gero and Maher, 1993; Pereira and Cardoso, 2001; Gomes et al., 1996]. Some methods identified in creative reasoning are: cross-domain transfer of ideas (analogy), combination of ideas, and exploration and transformation of conceptual spaces. Most of these methods are used in creative design [Gero, 1994a].

Creative design can be defined as a cognitive process that generates new classes of designs [Gero, 1994b]. A cognitive process resulting into a product will be said to be creative if the product satisfies certain properties or attributes [Dasgupta, 1994]. These properties determine the product’s creativity, thus defining guidelines to the product’s evaluator. There are two main properties identified: novelty and usefulness.

The first thing that comes into mind when talking about creative design is novelty. This is a mandatory characteristic in a creative solution. The creative product must be something different from what the evaluator knows or is expecting. Evaluation of a creative product has to do with the confrontation of two sets of knowledge. One is the information conveyed by the design product and the other is the knowledge that the evaluator possesses or uses in the evaluation. If the knowledge from the design product does not make part of the evaluator knowledge set or is sufficiently different, then it can be said that the product is novel.

During the evaluation of the presumably creative design product, the evaluator uses performance-measuring functions, in order to determine to which extent the design satisfies the problem requirements. These measuring functions make part of the evaluator’s body of knowledge. If minimal performance thresholds are met, then the design product is useful and solves the problem. This is another mandatory characteristic of creative design solutions. The creative design product must be appropriate and useful.

Analogue reasoning [Gentner, 1983; Hall, 1989; Holyoak and Thagard, 1989; Thagard et al., 1990] is a widely used problem solving method. It consists in the transference of knowledge from one domain to a different domain. This transference is based on similarities between past problems and the new problem. The transferred knowledge is then used to generate solutions for the new problem. The main benefit of analogical reasoning is its capability to transfer knowledge between different domains. This cross-domain transfer enables the generation of new creative designs, and can be considered as a creative process.

Analogue reasoning involves at least three phases: identifying candidates for analogy, mapping the candidate sources with the target, and transferring knowledge between source and target. The first phase is crucial to the generation of creative analogies, without a good set of candidate sources the subsequent phases maybe at stake. Two main retrieval strategies can be used: semantic-based retrieval and structural-based retrieval. Some of the most important works on analogical retrieval are described in the next paragraphs.

Gentner and Forbus [Gentner and Forbus, 1991] have developed the MAC/FAC (Many Are Called, Few Are Chosen) approach to analogy retrieval. It comprises a two-stage structural retrieval model, where in the MAC phase a crude idea
of the case structural content is used to retrieve a broad range of cases from memory. Then, in the FAC phase, a complete structural similarity metric is used to eliminate several cases, and to rank the chosen ones.

The Holographic Reduced Representations approach [Plate, 1994] combines semantic and structural similarity. The semantic similarity is based on the common semantic primitives of objects, while the structural similarity is accessed using the various roles of each object. This structural similarity is only a superficial assessment of the structural similarity.

An approach using constraint satisfaction (ARCS - Analogical Retrieval by Constraint Satisfaction [Thagard et al., 1990]) applies three similarity types through the creation of a constraint network. These types are: lexical similarity, structural similarity, and pragmatic similarity.

RADAR developed by Crean and O’Donoghue [Crean and O’Donoghue, 2001], was created to retrieve semantically distant analogies in a computationally tractable manner. It is based in structural properties to retrieve candidate sources, and the main concern of this approach was that distant analogies would not be rejected by the semantic similarity.

In this paper we present an approach to software design using analogy, which is integrated in a Computer Aided Software Engineering (CASE) tool named REBUILDER [Gomes et al., 2002a]. This tool is based on Case-Based Reasoning [Kolodner, 1993] and comprises a Knowledge Base with several types of knowledge, including WordNet [Miller et al., 1990], a lexical resource used in REBUILDER as an ontology. We focus on the retrieval phase of analogical reasoning, by studying the influence of different candidate selection strategies in the novelty and utility of generated artifacts. We have used four combinations of semantic retrieval and structural retrieval, to study the adequacy of each strategy to analogical reasoning.

The next two sections describe REBUILDER and the used knowledge base. These are two fundamental points that have to be addressed in order for the reader to understand the environment in which the analogical reasoning module is used. Then section 4 describes how analogy is performed in REBUILDER, along with the different retrieval strategies. Section 5 describes the experiments performed and it also discusses the results achieved. Section 6 presents other approaches of analogy in design. Finally section 7 concludes this paper.

2 REBUILDER

The approach carried out in REBUILDER uses CBR as the main reasoning framework, enabling the integration of several different types of knowledge. This section provides an overview on the system’s architecture.

Figure 1 shows the architecture of REBUILDER. It comprises four main modules: the UML editor, the knowledge base manager, the knowledge base (KB), and the CBR engine. It also shows the two different user types: software designers, and KB administrators. Software designers use REBUILDER as a CASE tool, and the reuse capabilities of the system. A KB administrator as the function of keeping the KB updated and consistent. The UML editor is the front-end of REBUILDER and the environment where the software designer develops designs. Apart from the usual editor commands to manipulate UML objects, the editor integrates new commands capable of reusing design knowledge.

The KB Manager module is used by the administrator to manage the KB, keeping it consistent and updated. This module comprises all the functionalities of the UML editor, and adds case-base management functions. These are used by the KB administrator to update and modify the KB.

The KB comprises four different parts: the case library, which stores the cases of previous software designs; an index memory used for efficient case retrieval; a data type taxonomy, which is an ontology of the data types used by the system; and WordNet [Miller et al., 1990], which is a general purpose ontology. This module is described in more detail in the next section.

The CBR Engine is the reasoning module of REBUILDER. As the name indicates, it uses the CBR paradigm to establish a reasoning framework. This module comprises five different parts: Retrieval, Design Composition, Analogy, Verification, and Learning. The retrieval module searches the case library for designs or design objects similar to the query. The most similar ones are presented to the user, allowing the designer to reuse these designs or part of them. The analogy module maps designs from the case library, to the query design. The resulting mapping establishes the knowledge transfer from the old design to the query design. The composition module can be used to adapt a past design (or part of it) to the query using design composition. The verification module checks the current design for inconsistencies. The learning module acquires new knowledge from the user interaction, or from the system reasoning. In this paper we focus on the analogy module.
3 Knowledge Base

The KB comprises the WordNet ontology, a case library, the case indexes and the data type taxonomy. In REBUILDER a case describes a software design, which is represented in UML through the use of class diagrams. Figure 2 shows an example of a class diagram representing part of an educational system. Nodes are classes, with name, attributes and methods. Links represent relations between classes. Conceptually a case in REBUILDER comprises: a name used to identify the case within the case library; the main package, which is an object that comprises all the objects that describe the main class diagram; and the file name where the case is stored. UML class diagram objects considered in REBUILDER are: packages, classes, interfaces and relations. A package is a UML object used for grouping other UML objects. A class describes an entity in UML and it corresponds to a concept described by attributes at a structural level, and by methods at a behavioral level. Interfaces have only methods, since they describe a protocol of communication for a specific class. A relation describes a relationship between two UML objects.

WordNet is used in REBUILDER as a common sense ontology. It uses a differential theory where concept meanings are represented by symbols that enable a theorist to distinguish among them. Symbols are words, and concept meanings are called synsets. A synset is a concept represented by one or more words. If more than one word can be used to represent a synset, then they are called synonyms. The same word can have more than one different meaning (polysemy). WordNet is built around the concept of synset. Basically it comprises a list of word synsets, and different semantic relations between synsets. The first part is a list of words, each one with a list of synsets that the word represents. The second part, is a set of semantic relations between synsets, like is-a relations, part-of relations, and other relations. REBUILDER uses the word synset list and four semantic relations: is-a, part-of, substance-of, and member-of. Synsets are classified in four different types: nouns, verbs, adjectives, and adverbs. REBUILDER uses synsets for categorization of software objects. Each object has a context synset which represents the object meaning. The object’s context synset can be used for computing object similarity (using the WordNet semantic relations), or it can be used as a case index, allowing the rapid access to objects with the same classification.

As cases can be large, they are stored in files, which makes case access slower then if they were in memory. To solve this problem we use case indexes. These provide a way to access the relevant case parts for retrieval without having to read all the case files from disk. Each object in a case is used as an index. REBUILDER uses the context synset of each object to index the case in WordNet. This way REBUILDER can retrieve a complete case, using the case root package, or it can retrieve only a subset of case objects, using the objects’ indexes. This allows REBUILDER to provide the user the possibility to retrieve not only packages, but also classes and interfaces. To illustrate this approach, suppose that the class diagram of Figure 2 represents Case1. Figure 3 presents part of the WordNet structure and case indexes. As can be seen, WordNet relations are of the types is-a, part-of and member-of, while the index relation relates a case object (squared boxes) with a WordNet synset (rounded boxes). For instance Case1 has one package called School (the one presented in Figure 2), which is indexed by synset School. It has also a class with the same name and categorization, indexed by the same synset, making also this class available for retrieval.

The data type taxonomy is a hierarchy of data types used in REBUILDER. Data types are used in the definition of attributes and parameters. The data taxonomy is used to compute the conceptual distance between two data types.

4 Analogical Reasoning

Analogical reasoning is used in REBUILDER to suggest class diagrams to the designer, based on a query diagram. The analogy process has three steps:

- Identify candidate diagrams for analogy.
- Map the candidate diagrams.
- Create new diagrams, by knowledge transfer between the candidate diagram and the query.
Candidate ranking receives a set of cases, which are then ranked based on a metric. This metric incorporates several criteria, which basically assess semantic and structural similarity. Three different similarity metrics are used: one that combines semantic and structure evaluation, a second one that uses an independence measure, and another that evaluates structural properties.

The first similarity metric compares two UML class diagrams based on four items: diagram synset (the package’s synset, which is a semantic evaluation) and diagram structure (combines structural evaluation of the diagram with object synset distances, is both structural and semantical) for a detailed description of this metric see [Gomes et al., 2002a].

The independence measure is specific to UML and evaluates the independence of each object in the class diagram based on the structural relations of diagram objects. Each class or interface in the class diagram is assigned a score that reflects the independence level of that object from other objects in the diagram, thus yielding a sense of importance of that object in the diagram (for more details see [reference omitted]).

The third similarity metric is based on the structural properties of the diagrams being compared. This metric is based on the work of O’Donoghue [O’Donoghue and Crean, 2002]. The main idea of this metric is using the diagram’s graph-like properties to determine the structural similarity of two diagrams. O’Donoghue argues that structural similarity is more important for the identification of good analogical candidates than semantic similarity [O’Donoghue and Crean, 2002]. But because the best structural similarity can only be assessed using structural mapping, which is computational expensive, he has used structural properties to assess the structural similarity, which are fast to assess [Crean and O’Donoghue, 2002].

The structural properties that we have used are:

- Number of loops in diagram.
- Average loop size.
- Number of classes.
- Number of interfaces.
- Number of generalizations.
- Number of realizations.
- Number of associations.
- Number of dependencies.
- Independence metric comparison, is an UML specific metric that assesses the independence degree of objects (see [Gomes et al., 2002a]).

Figure 4: The retrieval algorithm used in REBUILDER.

```plaintext
ObjFound ← ∅
Psynset ← Get context synset of QObj
Psynsets ← {Psynset}
ObjsExplored ← ∅
WHILE (#ObjsFound < N) AND (Psynsets ≠ ∅) DO
    Synset ← Remove first element of Psynsets
    ObjsExplored ← ObjsExplored + Synset
    SubSynsets ← Get Synset hyponyms (subordinates)
    SuperSynsets ← Get Synset hypernyms (superordinates)
    Psynsets ← SubSynsets ∪ ObjsExplored
    −Psynsets
    SuperSynsets ← SuperSynsets − ObjsExplored
    −Psynsets
    Psynsets ← Add SubSynsets to the end of Psynsets
    Psynsets ← Add SuperSynsets to the end of Psynsets
    Objects ← Get all objects indexed by Synset
    ObjectList ← Objects ∩ ObjectList
    ObjsFound ← ObjectList ∪ ObjsFound
ENDWHILE
ObjsFound ← Rank ObjsFound by similarity
RETURN Select the first N elements from ObjsFound
```

4.1 Candidate Selection

Cases are selected from the case library to be used as source diagrams. The selected candidates must be appropriate, otherwise the whole mapping phase can be at risk. Four alternative strategies for candidate selection can be used in REBUILDER. To better describe these four strategies, candidate selection is decomposed in two subtasks: retrieval and ranking. While retrieval should be a fast comparison task yielding a subset of candidate cases, ranking is a more complex and computational demanding phase where cases are ordered accordingly to a defined criterion.

Candidate retrieval returns a subset of cases from the case library. This should be a computational inexpensive task, with the main purpose of identifying a first set of cases that are possible candidates for analogy mapping with the target problem. Case retrieval is based on the WordNet structure, which is used as an indexing structure. The retrieval algorithm uses the classifications of the target problem object as the initial search probe in WordNet. For example, if the designer selects a package as the target problem, the retrieval algorithm uses the package’s synset as the initial search probe. Then the algorithm checks if there are any packages indexes associated with the WordNet node of that synset. If there are enough indexes, the algorithm stops and returns them. Otherwise, it explores the synset nodes adjacent to the initial one, searching for package indexes until the number of found indexes reaches the number of objects that the user wants to be retrieved. Suppose that the N best objects are to be retrieved, QObj is the query object, and ObjectList is the universe of objects that can be retrieved (usually ObjectList comprises all the library cases), the retrieval algorithm is depicted in figure 4.

Candidate ranking receives a set of cases, which are then ranked based on a metric. This metric incorporates several criteria, which basically assess semantic and structural sim-
Table 1: The retrieval strategies implemented in RE-BUILDER.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Retrieval</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Semantic</td>
<td>Semantic and Structural</td>
</tr>
<tr>
<td>2</td>
<td>Semantic</td>
<td>Structural (Independence)</td>
</tr>
<tr>
<td>3</td>
<td>Semantic</td>
<td>Structural</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>Semantic and Structural</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Structural (Independence)</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Structural</td>
</tr>
</tbody>
</table>

**Strategy 5** This strategy is similar to strategy 4 where no retrieval algorithm is used. The ranking is then performed by the independence measure.

**Strategy 6** This strategy is similar to the previous one, in the sense that it does not use a retrieval algorithm. It only ranks all the cases in the case library using the second similarity metric. This strategy is also implemented to be compared with the other ones.

### 4.2 Mapping Process

The second step of analogy is the mapping of each candidate to the query diagram, yielding an object list correspondence for each candidate. This phase relies on two alternative algorithms: one based on relation mapping, and the other on object mapping, but both return a list of mappings between objects.

**Relation-Based Mapping**

The relation-based algorithm (see figure 5) uses the UML relations to establish the object mappings. It starts the mapping selecting a query relation based on an UML heuristic (independence measure), which selects the relation that connects the two most important diagram objects. The independence measure is an heuristic used to assign to each diagram object an independence value based on UML knowledge that reflects an object’s independence in relation to all the other diagram objects. Then it tries to find a matching relation on the candidate diagram. After it finds a match, it starts the mapping by the neighbor relations, spreading the mapping using the diagram relations. This algorithm maps objects in pairs corresponding to the relation’s objects.

**Object-Based Mapping**

The object-based algorithm (see figure 6) starts the mapping selecting the most independent query object, based on the UML independence heuristic. After finding the corresponding candidate object, it tries to map the neighbor objects of the query object, taking the object’s relations as constraints in the mapping.

Both algorithms satisfy the structural constraints defined by the UML diagram relations. Most of the resulting mappings do not map all the problem objects, so the mappings are ranked by number of objects mapped (see [Gomes et al., 2002b]). An important issue in the mapping stage is: which objects to map? Most of the time, there are several candidate objects for mapping with the problem object. In order to solve this issue, we have developed a metric that is used to choose

---

```
Relations ← Get the best relation from the query diagram based on the independence measure of the relation
MappingList ← Ø
WHILE Relations ≠ Ø DO
    PRelation ← Get best relation from Relations based on the independence measure of the relation
    CRelation ← Get best matching relation from the base case relations that are matching candidates (structural constraints must be met)
    Mapping ← Get the mapping between objects, based on the mapping between PRelation and CRelation
    Remove PRelation from Relations
    Add Mapping to MappingList
    Add to Relations all the same type relations adjacent to PRelations, which are not already mapped
    (if PRelation connects A and B then the adjacent relations of PRelations are the relations in which A or B are part of, excluding PRelation)
Return MappingList
```

Figure 5: The relation-based mapping algorithm.

```
Objects ← Get object from the query diagram which has the highest independence measure
MappingList ← Ø
WHILE Objects ≠ Ø DO
    PObject ← Get best object from Objects, based on the object’s independence measure
    CObject ← Get best matching object from the base case objects that are matching candidates (structural constraints must be met)
    Mapping ← Get the mapping between PObject and CObject
    Remove PObject from Objects
    Add Mapping to MappingList
    Add to Objects all the objects adjacent to PObject which are not already mapped (and adjacent object B to an object A, are all the objects that have a relation with A).
Return MappingList
```

Figure 6: The object-based mapping algorithm.
the mapping candidate. Because we have two mapping algorithms, one based on relations and another on objects, there are two metrics: one for objects, and another for relations. These metrics are based on the WordNet distance between the object’s synsets, and the relative position of these synsets in relation to the most specific common abstraction concept (for details on these metrics see [Gomes et al., 2002b]).

4.3 Knowledge Transfer
The last step is the generation of new diagrams using the established mappings. For each mapping the analogy module creates a new diagram, which is a copy of the query diagram. Then, using the mappings between the query objects and the candidate objects, the algorithm transfers knowledge from the candidate diagram to the new diagram. This transfer has two steps: first there is an internal object transfer, and then an external object transfer. In the internal object transfer, the mapped query object gets all the attributes and methods from the candidate object that were not in the query object. This way, the query object is completed by the internal knowledge of the candidate object. The second step transfers neighbor objects and relations from the mapped candidate objects to the query objects, from the new diagram. This transfers new objects and relations to the new diagram, expanding it.

5 Experiments
In this section we present the results obtained from our experiments. The aim of these experiments is to study the correlation between the analogical retrieval strategies and the creative properties of the generated diagrams in the software design domain. The creative properties in which we are interested in are usefulness and novelty.

5.1 Setup
The results presented in 5.3 were obtained using a case library compromising 60 cases, each case has 1 package with 5 to 20 objects (there are a total of 586 objects). These cases are from four different domains: banking information systems, health information systems, educational institution information systems, and store information systems (grocery stores, video stores, and others). A set of 25 UML class diagrams were used as problems during these experiments, each problem compromises, on average, 4 objects. The problems used in these experiments also belong to the four domains formerly presented.

5.2 Method
During these experiments we used the 6 candidate selection strategies presented in table 1 for choosing the most promising candidates for reuse through analogy. When semantic retrieval is used as part of one of these strategies, only 10 (the first ten cases found by our retrieval algorithm) cases are used for ranking, otherwise the whole case library is used. After the ranking phase, the most promising candidate is submitted to the analogy module. Afterwards the output of the analogy module (a UML class diagram) is evaluated for both usefulness and novelty. For these experiments we use the object-based mapping algorithm, since this algorithm has shown better results in previous work (omitted reference).

<table>
<thead>
<tr>
<th>Strategy</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.277</td>
<td>0.693</td>
<td>0.693</td>
</tr>
<tr>
<td>2</td>
<td>0.275</td>
<td>0.654</td>
<td>0.544</td>
</tr>
<tr>
<td>3</td>
<td>0.275</td>
<td>0.655</td>
<td>0.505</td>
</tr>
<tr>
<td>4</td>
<td>0.277</td>
<td>0.690</td>
<td>0.689</td>
</tr>
<tr>
<td>5</td>
<td>0.276</td>
<td>0.646</td>
<td>0.432</td>
</tr>
<tr>
<td>6</td>
<td>0.274</td>
<td>0.649</td>
<td>0.304</td>
</tr>
</tbody>
</table>

Table 2: Values computed for assessing novelty of solutions generated through analogy.

Novelty is assessed using the similarity metric used in case retrieval. We assess the novelty of a solution (the output of the analogy module) using the similarity with the cases in the case base. Three different similarity values are computed using the generated solution:

1. The average similarity between the generated solution and the cases in the case library (N1).
2. The similarity value between the solution and the most similar case in the case library (N2).
3. The similarity value between the solution and the source case chosen for establishing analogies with the target case (N3).

Since the similarity metric, returns values between 0 and 1 where 1 is an identical match, we are interested in situations where this value is low. The results of this evaluation is presented in table 2.

Usefulness is evaluated through human judgement of the generated diagrams. Two human judges evaluated these diagrams, identifying:

1. The percentage of target objects that were mapped to a source case in which the knowledge transferred (methods and attributes from the source case to the target case) was incorrect or useless (U1).
2. The percentage of target objects that were mapped to a source case in which the knowledge transferred (methods and attributes from the source case to the target case) was partially incorrect (U2).
3. The number of objects that were transferred into the target diagram and that were considered incorrect or useless (U3).

These values are presented in table 3 of the next subsection. We are aware of the subjectivity involved in the result of these experiments, especially in the judgement of the usefulness, and feel that the only faithful evaluation is through a case study in a real software production environment where the tool is judged by its users. Despite this fact we will present our evaluation in the preceding section.

5.3 Results
In tables 2 and 3 we present some of the values computed for our analysis of novelty and usefulness by analogy retrieval strategy (we omit standard deviations and confidence intervals due to size restrictions of the present paper).
Table 3: Values computed for assessing usefulness of solutions generated through analogy.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>U1</th>
<th>U2</th>
<th>U3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14%</td>
<td>1%</td>
<td>0.600</td>
</tr>
<tr>
<td>2</td>
<td>14%</td>
<td>3%</td>
<td>1.640</td>
</tr>
<tr>
<td>3</td>
<td>15%</td>
<td>40%</td>
<td>1.400</td>
</tr>
<tr>
<td>4</td>
<td>9%</td>
<td>2%</td>
<td>0.600</td>
</tr>
<tr>
<td>5</td>
<td>20%</td>
<td>20%</td>
<td>3.560</td>
</tr>
<tr>
<td>6</td>
<td>21%</td>
<td>36%</td>
<td>2.720</td>
</tr>
</tbody>
</table>

As can be seen from Table 2, the average distance of the solution in relation to the rest of the case library is practically the same. This observation can be due to the fact that the cases in the case library are very sparse, in the sense that different domains are modelled by the cases present in the case library. So whatever the solution, on average it will tend to be dissimilar to the case library. Observing the value of similarity between the solution and the most similar case (N2) we can see that strategy 1 and 4 present the highest values, which makes sense since this metric contains a semantic component. In respect to N3, we can see that strategy 5 and 6 obtain solutions that are dissimilar to the source case used for establishing analogies. These values are expected, taken in consideration that neither of these strategies use semantics during the retrieval or ranking phases, and that the similarity metric does. Thus, these strategies are able to generate more novel solutions than the other strategies, which are by some way using semantics to retrieve and rank source cases.

Table 3 presents the values obtained during the evaluation of the usefulness of the generated solution. We remind the reader that these values represent situations that were considered wrong or useless, this means that lower values correspond to solutions that are more useful since these require less corrections from the software designer. As we can observe strategies 5 and 6 seem to generate the diagrams which require the most correction. These high values may be due to the fact that the knowledge transference scheme implemented simply transfers objects from the source domain into the target domain without transforming the source object into an analog in the new target domain. If these source objects are transferred into completely different domains they will usually be considered incorrect.

These results suggest that semantics should be considered during the retrieval phase in order to obtain the most useful diagrams, that is, diagrams that require less correction. On the other hand the most novel analogies tend to occur when the ranking phase assesses the best candidates through the use of structural properties. Using a retrieval strategy that combines both aspects can be a good retrieval strategy for analogy retrieval. Such an example are strategies 2 and 5, which are able to maintain a compromise between strategies 1/4, which are more accurate but less novel, and strategies 3/6 that are capable of generating more novel solutions but with more errors in the class diagrams.

6 Related Work

There are several research works that use analogy to generate designs. From these we have selected the ones that are similar to our work. Most of these are from other domains of application, such as mechanical design, but the last two works presented address the software design domain.

Qian and Gero [Qian and Gero, 1996] show how analogy-based design can be implemented using the Function-Behavior-Structure (FBS) knowledge representation. They claim that the FBS path connecting function with behavior, and behavior with structure is the key to analogy in design. They consider that function and behavior are analogical retrieval cues, due to the one to many mapping between function and behavior, and between behavior and structure.

KDSA [Wolverton and Hayes-Roth, 1994] is an approach to analogy-based design, using a controlled search algorithm in a semantic network. The base idea of this approach is that good analogies stem from artifacts with similar functions, but different behaviors and structures.

Goel and Bhatta [Bhatta and Goel, 1997] developed model-based analogy, a computational theory of analogy-based design that uses domain models about artifacts. These models represent the structure, behavior and function of a design. It uses design patterns as generic design abstractions that are learned during system operation. A design pattern can then be applied to a concrete situation, generating a new analog design.

CADET is a system developed by Sycara and Navinchan-dra [Sycara and Navinchandra, 1991] that uses behavior as a thematic abstraction for case retrieval. They use influences as an appropriate thematic abstraction for representing behavior of physical devices.

There are some research works that address the area of analogy applied to software reuse. From these we point out the work of Maiden and Sutcliffe in Ira [Maiden and Sutcliffe, 1992]. Ira is a CASE tool system that enables the reuse of software specifications based on analogy. It provides user support for the task of system specification. Specification reuse involves three processes: categorization of a new problem, selection of candidate specifications, and adaptation of the selected specification to the new domain. Ira addresses these three issues by: obtaining the description of the new target problem from the software engineer; controlling the interaction with the user during selection and adaptation of an analogous specification; and reasoning with critical problem features to match new problems.

Spanoudakis [Spanoudakis and Constantopoulos, 1994] developed a computational model of similarity for analogical software reuse based on conceptual descriptions of software artifacts. Their approach is based on semantic similarity of software objects.

7 Conclusions and Future Work

This paper describes the analogy module os REBUILDER, and presents a study about the influence of the retrieval strategy used in the novelty and utility of generated solutions. From the experimental results it can be inferred that semantic retrieval generates more useful class diagrams, but they
are less novel than diagrams generated using structural strategies. So, a combination of both retrieval aspects indicates that it is a good retrieval strategy for generation of designs using analogy. Future work will address further variations of the retrieval strategies and different generation mechanisms, like design composition. An empirical study on a real development environment is also a goal of our research.

Acknowledgments

This work was partially supported by POSI - Programa Operacional Sociedade de Informação de Fundação Portuguesa para a Ciência e Tecnologia and European Union FEDER, under contract POSI/33399/SRI/2000, by program PRAXIS XXI. REBUILDER homepage is http://rebuilder.dei.uc.pt.

References


