# CASE-BASED DESIGN USING WEAKLY STRUCTURED INFORMATION

SUBMITTED: August 2001
REVISED: June 2002
PUBLISHED: July 2002 at http://www.itcon.org/2002/2/
EDITOR: B-C. Björk

***Peter Johansson, Assistant Professor,***
***Department of Structural Engineering, Chalmers University of Technology, Sweden***
***email: peter.johansson@ste.chalmers.se***

***Mina Popova, Ph.D. Candidate***
***Dept. of Visualization and Modelling, Chalmers University of Technology, Sweden***
***email: mina@arch.chalmers.se***

***SUMMARY****: Over 50% of the work done by the designer on a day-to-day basis is routine design that consists of reusing past design solutions (Moore, 1993). Despite of this fact, there are no tools that rationally support reuse of such solutions. Case-based design (CBD) has been pointed out as a promising aid to help this situation. In order to be of practical use, however, a case-based design system has to be able to use the information that the designer creates during the design process. The design information that the designer creates is today mostly in the form of weakly structured information, e.g. text documents, calculation documents, and 2D-drawings. This paper proposes an approach that enables capturing and representation of weakly structured information for the purpose of case-based structural design. The representation proposed allows us to apply most of the object-oriented abstract principles also on weakly structured information. It is also shown how the conceptual framework, the dependency structure, and the design process can be captured, represented, and used in CBD. The approach is successfully implemented into a prototype for reuse of computerized design calculation documents.*

***KEYWORDS****: case-based reasoning, design, structural design, representation, object orientation, weakly structured information.*

## 1. INTRODUCTION

Empirical knowledge plays a significant role in the human reasoning process as previous experiences help in understanding new situations and in finding solutions to new problems. Experience is used when performing different tasks, both those of a routine character and those that require special skills. This is also the case for structural design where over 50% of the work on a day-to-day basis is routine design that consists of modifying past solutions (Moore, 1993). This means that most of the design problems have been solved before, in many cases over and over again. Despite this, the computer support used by designers still lacks the ability to use experiential knowledge in a rational way.

In recent years, researchers in Artificial Intelligence (AI) have studied if *cases* (information about specific problem-solving experiences) could be used as a representation of experiential knowledge. Cases are valid at a specific situation in contrast to generalized knowledge, e.g. rules. Making use of past experience in the form of cases is commonly known as Case-Based Reasoning (CBR) (Kolodner, 1993). The application of CBR in design, known as Case-Based Design (CBD), is still in its infancy even though several CBD-systems focusing on various domains have been developed (Maher, et al 1997, Rivard and Fenves, 2000). Although many of them are useful in solving the specific problem that they are aimed for, CBD-systems are seldom used in practice. One of the reasons is that the information representation used is system-specific. Creating such representations provides the system developer with an opportunity to investigate new ways to represent design information and much knowledge has in this way been gained. On the other hand, this limits the information available for CBD to information either created by the CBD-system or information translated to the system-specific representation. Because these representations are rather complicated and different from those used by the ordinary designer when documenting design information, it is difficult to achieve an automatic translation. For this reason, most CBD-systems only contain cases that are produced using the respective system or information translated by hand to the system-specific representation.

Case-Based Reasoning (CBR) originates from the cognitive observation that humans often rely on past experience to solve new problems. Using this observation, Schank created the theory of *dynamic memory*, which describes a concept of memory organization that could be used as a guideline for computer representation (Schank, 1982, Schank, 1999). The premise of dynamic memory is that remembering, understanding, experiencing and learning cannot be separated from each other (Kolodner, 1993). We understand by remembering old similar situations and use these to create expectations about the new situation. If these expectations turn out to be right, we feel that we understand; if the expectations fail we try to explain why by remembering old situations with similar failures. These explanations are then used to change the memory (learning) so that also the new situation can be understood. In order to make this possible, the same knowledge structure has to be used for remembering, understanding, experiencing, and learning.

The analogy between dynamic memory and a system facilitating CBD is rather near at hand. The main aim for CBR in such a system is to find, i.e. recall, old experience that can be helpful in the present design situation. This experience is used when designing for understanding the problem and for finding a solution. The design activity creates another experience that can be stored in the design system for the purpose of reuse. As stated in the theory of dynamic memory, this can only be possible if the CBR activity and the design activity share a common representation.

The theory of dynamic memory also implies that understanding is the main aim and remembering supports this activity. Using analogy again, it can be stated that designing is the main aim for a design system while CBR aids this activity. Concerning the choice of representation, this ought to yield that the representation and the information used for designing should also be used for CBD. It should also be pointed out that unless the CBD-process becomes more or less automatic the designers would be reluctant to add potentially useful cases to the case-base (Flemming and Woodbury 1995) or to try to reuse old cases. The only way to avoid extra work for the sake of CBD is by enabling the CBD-system to use the information created by the designer during the design process.

Having this as a base, this paper proposes an approach for capturing and representing this information (Chapter 2), and shortly presents how this information can be retrieved, having this approach (Chapter 3). Chapter 4 describes ARCADE, a prototype implemented to test the proposed approach, and an example showing the benefits and the drawbacks of the approach. Chapter 5 concludes the paper.

# 2. REPRESENTATION

## 2.1 Structured, Weakly Structured Information, Raw Data

Simoff and Maher (1998) divide the computerized design information into three groups:

- Structured information (called structure-valued data by Simoff and Maher 1998), e.g. attribute-value pairs, relational tables and object-oriented data structures;
- Weakly structured information, including, e.g. texts in free or table format, and calculation documents;
- Raw data, including e.g. raster images of photographs, sketches, animated images, and audio and video data.

### 2.1.1 Structured Information

Here, structured information is categorized by the fact that the data structure of the product- and process-specific information is being known to the case-based design system in advance. The data structures known in advance can be divided into two groups:

- Application-related data structures,
- Standardized data structures (product models).

An application-related data structure is used and hard-coded in an application, e.g. data structures used in CAD systems, while standardized structures or product models are independent of applications, e.g. STEP (ISO 10303) (ISO 1995) and Industry Foundation Classes (IFC) (IFC 1999). Today, most of the data structures in both groups use the object-oriented paradigm.

Concerning the representation issue in a CBD-system, it is reasonable to assume that the system has knowledge about standardized product models, but it is doubtful to expect that it also has knowledge about the structures of data used in individual applications.

Representation of the information with the help of standardized product models like IFC and STEP facilitates the use of CBR in design. A product model classifies the information, which is of great importance in the retrieval process. Such a model also solves the correspondence problems by labelling the information with a certain name and using the same name whenever the object is referred to. Despite these benefits, the incapability to represent all the information needed for the purpose of CBD is the reason why the standardized product models have seldom been used in CBD. The product models mainly focus on representing the physical characteristics of the solution (Turk, 1998). For a CBD system to be of use, information about the *function* and the *behavior* of the structure (the purpose of and the response from the structure when used) is needed as well (Maher, et al 1995) and (Qian and Gero, 1996). *Design rationale* or *design intent* is another type of knowledge that would facilitate CBD. Design rationale is the collection of information about the evolution of a design product. Capturing and storing design rationale is synonymous with explicitly expressing the requirements, preferences and reasoning for the final solution (de la Garza and Alcantara, 1997). Or, in simpler terms, it can be defined as the rationale behind the decision-making process during design (de la Garza and Oralkan, 1995). In order to represent this deep knowledge the need for developing a comprehensive conceptual framework, "ontology", for the AEC-industry has been identified (Simoff and Maher, 1998) but the standard product models are not capable of dealing with that issue at present (Ndumu and Tah, 1998). Concepts like Workshop, Building section, Frame, Base plate and so on, are not defined in IFC, but they are vital when dealing with design rationale, which in turn is important for CBD.

It has also been pointed out that the creation of new classes/concepts in a design environment (Fischer, 1994) and the changing of the information structure of a class should be possible. If this is achieved, the product model will become evolvable. Then, while solving a problem, the designer should be able to introduce new concepts and even evolve their description, e.g. by introducing new variables and relations, in the line of ordinary design work. Such concepts and their respective descriptions can then directly be used in the retrieval process.

To conclude the discourse, it can be stated that standard product models, like STEP and IFC, cannot in their present form be used as the only representation in a CBD-system. However, such standards serve well as a foundation for the representation.

### 2.1.2 Weakly Structured Information

In the second group, weakly structured data, the structure of the product and process information is not known in advance but it can never the less be used by a CBD-system. This is often the case when the designer uses general applications like text editors, mathematical editors, spreadsheets, and general CAD-systems. General applications are widely used and the designer sometimes chooses those even when special applications suitable for the current problem do exist. One of the reasons is probably that specific CAD or analysis and design programs cannot represent all information the designer wants to describe. It should also be mentioned that much of the information needed for CBR is documented using these general applications. For instance, the design rationale concerning the choice of a concept is often described mainly in a system description document, which is in most cases created using a text editor. Another example is the fact that many of the concepts used by the structural designer are only documented in the design calculations.

### 2.1.3 Raw Data

The product and process information in raw data is, to a great extend, not reachable to a CBD-system. Examples of such data are: raster images of photographs, sketches, animated images, and audio and video data. This data cannot be used for automatic reasoning using the computer but it is still very useful for a human being.

A CBD-system needs information from all of the three groups: the two first for retrieval and adaptation, the third mostly for the purpose of browsing, and all three for the purpose of reuse.

## 2.2 Weakly Structured Information as Design Calculation Documents

As the title of this paper declares, the study described here focuses on weakly structured information. Let us consider design calculation documents as an example. Designers have recently started to use mathematical

editors, such as Mathcad (MathSoft, 1995), http://www.mathsoft.com/mathcad/, for creating design calculation documents and they are used more and more.

A design calculation document can be regarded as a list of variable definitions where every variable has a name, a physical unit, and a value. The variables can be independent or dependent. The documents can also contain pictures and text. The text can be divided into headings and comments.

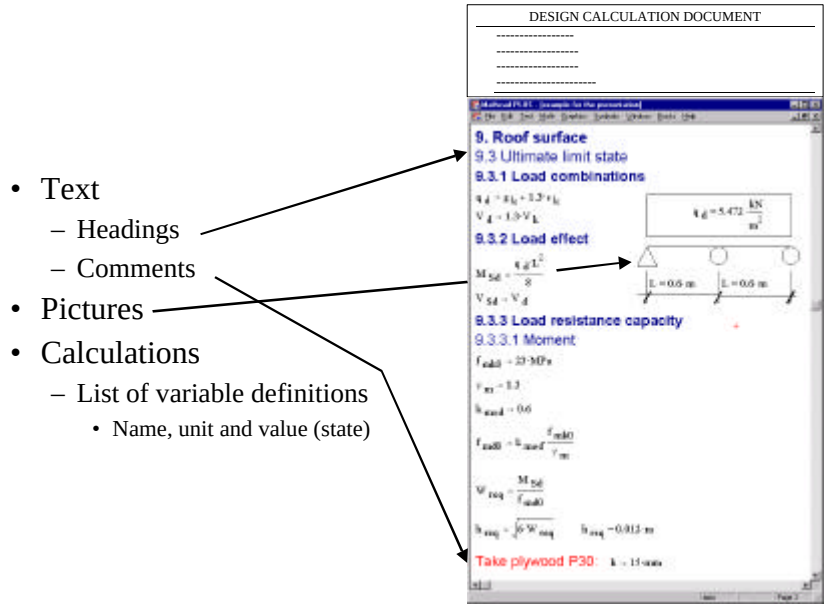


*FIG.1: Example of a design calculation document created using Mathcad.*

## 2.3 Dependency Structures and Design Process Representation

When variable names are used in a definition of a dependent variable, dependencies are created between the variable defined and each named variable, e.g. $f_{md0}$ is dependent of $k_{\mathrm{mod}}$, $f_{mk0}$, and $g_m$. These dependencies create a dependency structure.

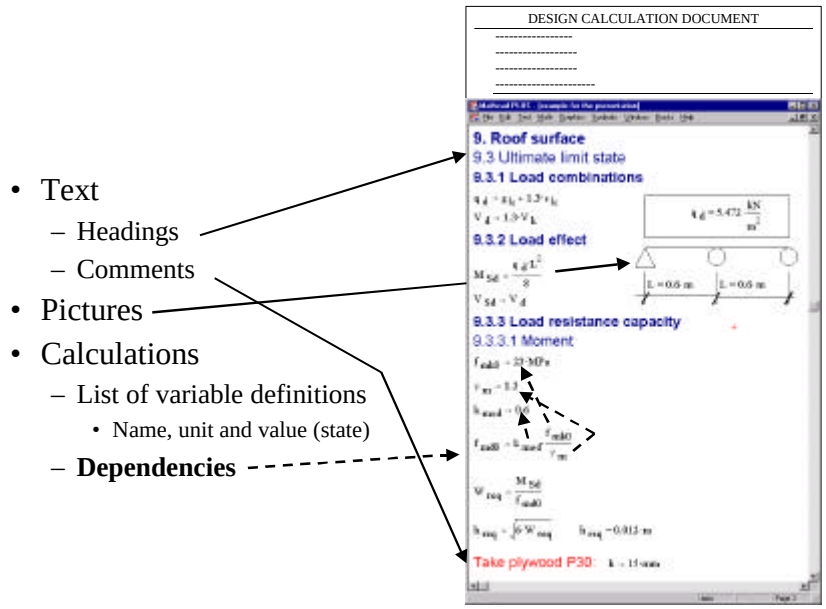


*FIG.2: Example of dependencies in a design calculation document.*

Using this dependency structure, the design calculation documents can be used as a representation of:

- The design rationale;
- The detailed process.

### 2.3.1 Design Rationale

The dependency structure describes some of the *design rationale* (de la Garza and Alcantara, 1997). Looking at the example in Fig.2, for instance, it can be stated that the height of the plywood board was chosen because of the length $L$ and the loads $g_k$ and $s_k$. That is, the dependency structure of the variables allows us to capture automatically the design rationale, which is very important knowledge for a CBD-system. This fact emphasizes the importance of trying to catch the dependency structure of the design information.

### 2.3.2 Process Representation

Documenting design calculations with the help of mathematical editors also makes it possible to represent the detailed process by using the variable definitions. There are three types of information in calculations that describe the process:

Variable definitions (functions), e.g. $f_{md0} := k_{mod} \cdot \frac{f_{mk0}}{\gamma_m}$;

*Dependencies* between the variables e.g. $f_{md0}$ is dependent of [ ], and [ ];

*The order* of variable definitions, i.e. [ ], [ ], $f_{md0}$.

These three types of information help to represent the process of calculations, i.e. the order in which the variables are calculated. We can notice that in most cases this process is rather similar to the design process considering the order in which the designer calculates, evaluates, and decides the variables and their values.

These three categories: the function, the dependency (or the relation between output from one function and input in another) and the order of functions, are fundamental when representing processes. For instance, IDEF0 http://www.idef.com/idef0.html has functions with input, including input, controls and mechanisms, and output, dependencies, and order of the functions. It should be stated, however, that IDEF0 also facilitates the representation of feedback and iteration, which the simplified process model proposed here does not.

This simple process representation can still be very useful for a CBD-system because the information about the process is, in general, more adaptable than the information about the solution, (Mostow, 1989, Carbonell 1986, Raphael et al. 1994). Here, we should notice that capturing the design process implies capturing the dependency structure, which, in its turn, makes it possible to capture the design rationale.

## 2.4 Concept/Component Approach

Realizing that "a design plan can rarely be reused in its entirety" (Mostow, 1989), a fact that also applies for design solutions, is the main reason for striving to subdivide the information acquired into reusable cases.

As stated above, most of the structured information is organized using the object-oriented paradigms. Although the most natural way for that kind of subdivision is according to the objects defined by the classes in the model, there is a problem as many concepts lack a correspondence to a specific class. This makes it hard to separate information about these concepts. In most object-oriented program languages, attributes and operations can only be defined in a class; objects in these systems can only be created using a class. If an object needs more attributes or operations than defined in the class, a new class has to be created. In most cases this is not a limitation but rather reduces the complexity and easies the system maintenance. In the theory of dynamic memory and in human thinking, this is not the case, though. Humans create classifications containing similar classes with objects that can differ in many ways. This kind of a looser classification where objects with different sets of variables and behaviour can still be seen as belonging to the same type is often called conceptualisation and the "types" are called concepts, instead of classes (Mitchell 1997).

Rivard and Fenves (2000) describe the term component in the following way: “A *component* is defined as a group of closely related attributes that are founded together in a repository (access-cohesive), instantiated at the same time (time-cohesive), and that represent the same concept (concept-cohesive)”. This term has been adapted in this study too but with two significant differences:

Time-cohesion is not regarded because the design process is iterative and the information concerning an instance of a concept, e.g. a beam, can be evolved/ changed many times, before the final result occurs. Having time-cohesion would create one case for every iteration. Although some of these cases can be reusable most of them will not.

In this study, a component can also contain other types of information than attributes.

This gives a somewhat different definition:

A *component* is defined as information that is founded together in a repository (access-cohesive), and that represents the same concept (concept-cohesive).

This definition suggests that the relation between concept and component is similar to the relation between class and object. That is, an object is an instance of a class (structured data) and a component is an instance of a concept (weakly structured data).

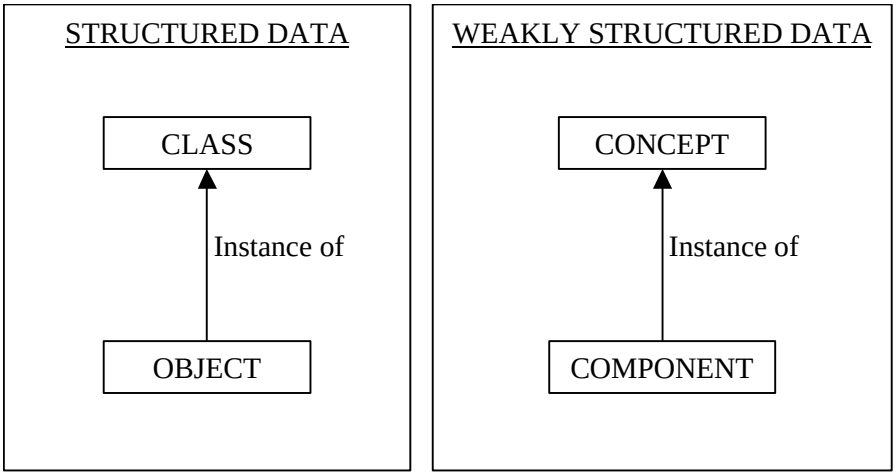


*FIG.3: Analogy between class/object and concept/component.*

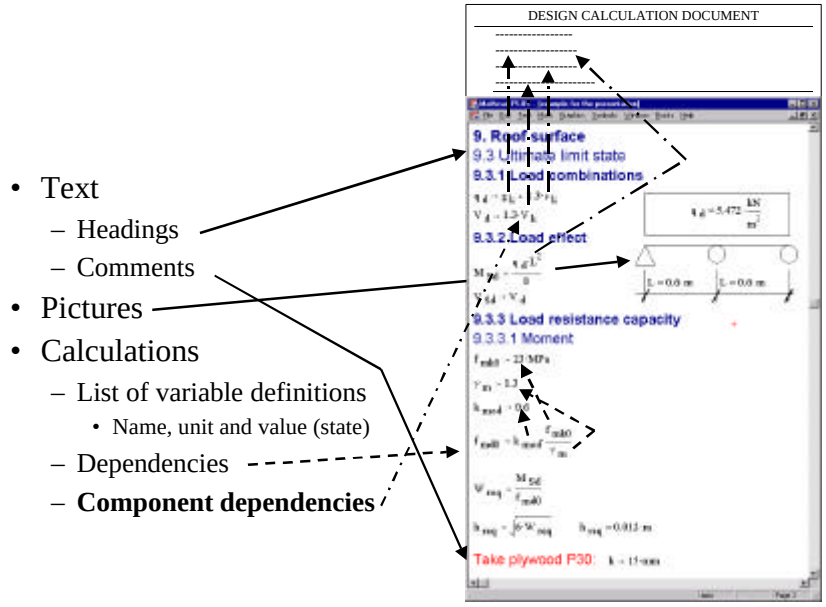


*FIG.4: Example of a component*

As described above, the structure of the weakly structured information is not known in advance but some structure useful to the CBD-system does exist, which assists subdivision. This structure should of course be used for the purpose of subdivision. The best way to ensure a natural subdivision is to use the one created by the designer. In the case of design calculation documents, this is possible to achieve by using headings. The heading, together with the content down to the next heading at the same or higher level, describes an instance of a concept, a component. The text in the heading often reveals the concept, e.g. *Roof surface* in Fig. 4. New concepts are also easily introduced by creating a heading with a text describing the concept and using this text for that concept henceforth. In this way, a conceptual framework can be created with a minimum of extra work. The content describes the variables and other information about the component i.e. the state and the behavior. Two components of the same concept do not necessarily have the same set of variables.

Using the different hierarchy levels of the headings, aggregation relationships between components can be acquired and with the help of the dependency structure, the *dependencies between components* can be calculated. The dependencies of a component are defined by the variables used in a component but defined elsewhere. The component *Roof surface* in Fig. 4 has, for instance, dependency relations with the components that contain the variables $g_k$, $s_k$, and $L$.

Because a dependency is a one-directed association (Larman, 1997) we can conclude that the component/concept approach creates an opportunity for using most of the object-oriented abstraction principles (classification, aggregation, and association) on weakly structured information in the form of design calculation documents.

## 3. RETRIEVAL OF CASES

The retrieval mechanism in Case-Based Reasoning-systems ensures the retrieval of cases useful for the current design situation. The retrieval process can be divided into three sub-steps: Indexing, Search, Match.

### 3.1 Indexing

A CBD-session takes place when the designer identifies a design problem and decides to try to solve it using CBR. When the designer asks the CBR-system to solve a problem, the first task performed by the system is to acquire information about the problem and its environment. It is important that also this process in a CBR-system is as automatic as possible and that the required information is acquired from the information created during the design activities. The information about the design problem should be structured and labelled in the same way as the information in the old cases. This can be accomplished by letting the designer start a CBD-session in the some way as an ordinary design of a component, by writing down what information is known about the component and what is wanted (see section 4.4).

### 3.2 Search

Search aims at dividing the case-base into two groups: cases that can be of interest and cases that cannot. For instance, if a column of concrete is being designed, an old case labelled “column” and “concrete” is of interest while a case labelled “beam” and “steel” is probably not. That is, search is the process of finding components that are instances of the same concept as what is wanted. Deriving analogy from object-orientation we can state that search is using the classification relationship.

It is widely accepted that information about the function of a component is more important for retrieval than information about the physical product. In many studies, it has, for this and other reasons, been suggested that the information should be divided into these two categories: *information about function* and *information about the physical solution that solves the function* (Gero, 1990, Abdalla, et al 1991, Garrett and Maher Hakim, 1992, Maher, et al. 1995). An interesting observation here is the fact that the concept used by the designer describes a function in most cases. The concept *beam*, for instance, is actually describing the function “carry transversal load over a span”. That is, a function which is an essential feature of a design problem. This function is then solved by a physical entity such as a welded steel plate girder. Other examples of such concepts are column, frame, and floor.

### 3.3 Match

When a search is conducted and a group of relevant old cases have been identified, they can be investigated in more detail. This is performed in the match procedure whose main goal is to calculate the usefulness of the old

cases in order to retrieve the most useful ones. The usefulness of a case depends on real world circumstances, which are not completely known at retrieval time (Burkhard, 1998). Thus, the usefulness criterion can only be applied *a posteriori*, i.e. it can only be fully evaluated when the case has been used (Raphael, 1995). However, for case-based reasoning, the usefulness has to be estimated *a priori*. In order to solve this problem, similarity is used as formulated in the hypothesis, which CBR is founded on:

Similar problems have similar solutions.

This statement indicates that similarity is considered as an a priori criterion to approximate the a posteriori criterion of usefulness (Burkhard, 1998).

As stated already, the concept/component approach makes it possible to impose most of the object-oriented abstraction principles. This enables estimation of the similarity between the component describing the new design problem and the old components in the case-base by comparing the variables and the relations of the components.

## 3.4 Process Matching

If the cases in the case-base contain process information, this information should also be used in the match process. Theory about how to do this can be found in the field of derivational analogy. Assume that we have a case-base with cases containing the design process (design plans) and that these plans are general and take all facts of importance into consideration. In this case, two kinds of indices have been defined by Veloso (1992): Goal state, and Footprint.

For the purpose of understanding these concepts, let us look at a part of a design calculation (Fig.5).

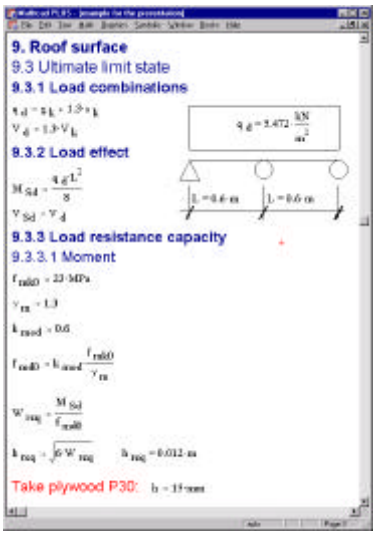


*FIG.5: A part of a design calculation document.*

The goal state is the state that is wanted. When discussing a case from a design calculation document, this can be reformulated as "the variables we wish to calculate". Looking at the design calculation above, it is rather clear that $h$ is the goal state.

Footprint for a case is the set of the weakest preconditions necessary to achieve the goal state. For a case from a design calculation document, this can be reformulated as “the variables that are used in the case, but not defined in it”, e.g. the footprint of the case above is $g_k$, $s_k$, $V_k$, and $L$. It can be noticed that the footprint is also the dependencies of the component.

Using the concept of goal state and footprint, two questions that should be answered in the match process can be stated:

Does the old case contain the goal state of the design problem? That is, does it contain the information that is wanted?

Does the current state for the design problem contain the footprint of the old case? That is, is the information needed by the old case available?

## 4. EXAMPLE

This example was created using the CBD-system ARCADE in order to illustrate how the weakly structured information could be used for case-based reasoning and to evaluate the approach described in this paper.

### 4.1 ARCADE

ARCADE is a prototype implemented to study how weakly structured information in the form of design calculation documents can be used for case-based reasoning. ARCADE used the format of Mathcad 6.0 (Mathsoft, 1995) and it was implemented under Windows NT using the Visual C++ development environment http://msdn.microsoft.com/visualc/default.asp.

### The Case-Base

For this example, the case-base contains design calculations from six different projects.

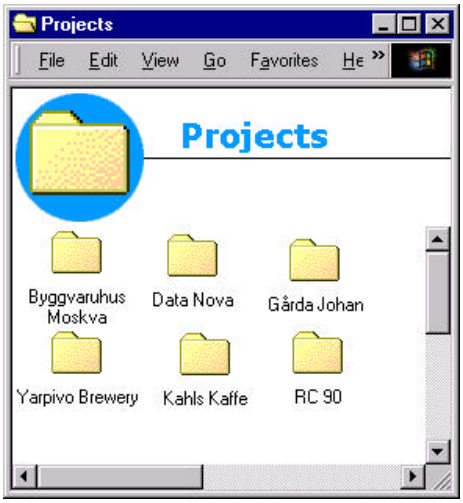


*FIG.6: The projects used in this example.*

The design calculation documents used were gathered from several Swedish designers hence the documents are in Swedish. Only in one of these projects, however, they were documented using Mathcad. In the other five, they were documented in the traditional way using pencil and paper. The calculations documented traditionally were keyed into Mathcad in a rather straightforward manner. The main difference between the pencil-and-paper version and the Mathcad version of a document is that the equations in the former mostly contain numerical values. For this reason, the variable values used in the equations have to be translated to the variable names.

The calculations documented using Mathcad show that it is natural to use variable names in an equation instead of entering the values of the variables when using Mathcad for documentation (or any other programs of similar type, e.g. Matlab or Mathematica).

The other main effort was to create correspondence between the calculations. These calculations were created by four different designers who did not always use the same notations. Both concepts used in headings and variable names differed somewhat between the designers, which required a change of both headings and variable names to some extent. Different calculations made by one designer did not differ at all so much as the calculations from different designers, though.

When having a Mathcad-file containing design calculations, ARCADE can acquire the information, divide it into components, and store them in the case-base using the Acquisition File-command.


*FIG.7: The Acquisition File command.*

## 4.3 Current Project

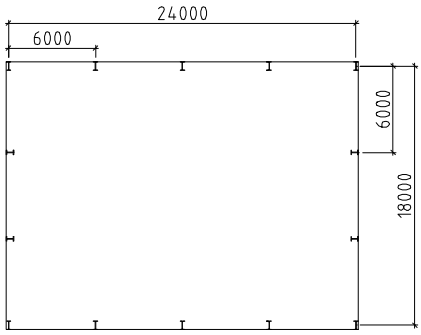


*FIG.8: Column layout.*

This design example concerns the foundation of a small warehouse. As most of the conceptual design has been carried out, the designer has decided the location of the columns and chosen a piled foundation.

The complete design calculation documents of the present design containing information about the main geometry of the building and the load calculations can be found in (Johansson, 2000).

## 4.4 Indexing of the Design Problem and Retrieval

The structural engineer is now about to start designing the foundation of the building in the project called Current. This procedure resembles an ordinary calculation: a heading is created and information known about the design problem is keyed in (member variables). On top of that, the designer also keys in the information wanted (goal state variables).

As stated above, the concept of a component can be found in the heading of the component. The current version of ARCADE takes, for simplicity, the first word in the heading as being an identifier of the concept. That is, a search is performed simply by comparing the first word in the heading of the component describing the design problem with the first word in the heading of the old components in the case-base.

In this example, the concept of the problem is described in the heading as "*Bottenplatta*" (Foundation slab). By placing a question mark after the concept, the designer indicates for ARCADE that this is a design problem. The goal state variables are given by creating variables without values, e.g. $Spårlinje_L$ (centre distance between piles in the long direction). When the problem description is ready, a retrieval session can be performed using the Retrieve Cases-command.

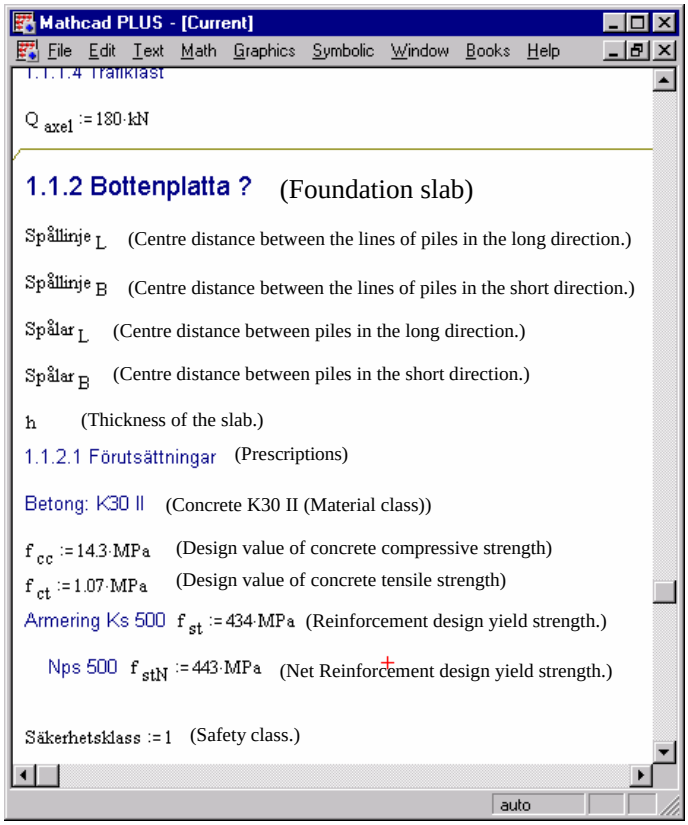


*FIG.9: Adaptation of an old component to a design problem. (Between the parentheses is an English translation of the heading. The translation is not included in the original calculation document.)*


*FIG.10: The Retrieve Cases command.*

This starts a retrieval session with the following result:

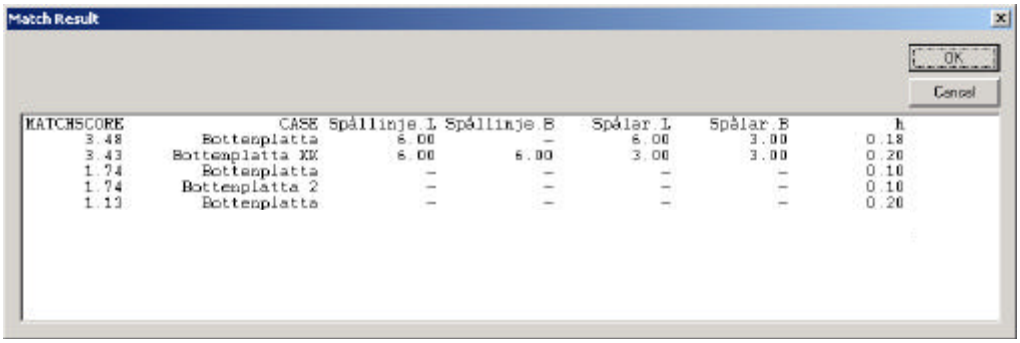


*FIG.11: The match result.*

Five different old components of the concept Bottenplatta (foundation slab) have been retrieved from the case-base. We can notice that two of them, *Bottenplatta* and *Bottenplatta KK* are more similar than the rest.

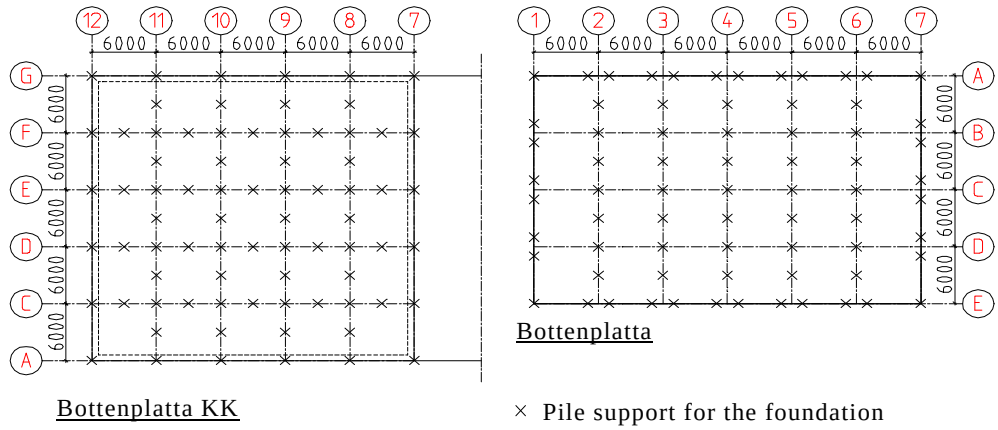


*FIG.12: Placement of piles in the two old components with the highest match score.*

ARCADE prints more detailed match result into a text file. Let us study this information focusing on the two components with the highest match score.

*Table 1. Detailed match result.*

| | | Old Case: Bottenplatta | BottenplattaKK |
|---|---|---|---|
| Variable Type | Variable Name | | |
| Goal state | Spållinje.L | 1 | 1 |
| Goal state | Spållinje.B | | 1 |
| Goal state | Spålar.L | 1 | 1 |
| Goal state | Spålar.B | 1 | 1 |
| Goal state | h | 1 | 1 |
| **Goal state Match score:** | | **0.8** | **1** |
| | | | |
| Member | f.cc | 1 | 1 |
| Member | f.ct | 1 | 1 |
| Member | f.st | 0.70 | 0.65 |
| Member | f.stN | 1 | 1 |
| Member | Säkerhetsklass | 1 | 1 |
| **Member variable Match score:** | | **0.94** | **0.93** |
| | | | |
| Footprint | q.NLk | 0.98 | 0.22 |
| Footprint | S.B | - | 1 |
| Footprint | S.L | 1 | 1 |
| **Footprint Match score:** | | **0.99** | **0.74** |
| | | | |
| **Children Match score:** | | **0** | **0** |
| **Parent Match score:** | | **0.75** | **0.75** |
| **Commonality Match score:** | | **0** | **0.01** |
| | | | |
| **Total Match score:** | | **3.48** | **3.43** |

In the *goal state match*, the design problem asks each old component of the same concept if it contains the goal state variables. The only difference between these two old cases is that *Bottenplatta KK* contains *Spållinje*$_B$, while *Bottenplatta* does not which gave the former a higher *goal state match score*. In the case *Bottenplatta*, the slab is calculated as carrying the load in only one direction and does only have lines of piles along the length (see Fig 12), thus rendering the variable *Spållinje*$_B$ unnecessary while designing *Bottenplatta.*

In the *member variable match*, the design problem asks each old component if it contains the member variables of the design problem and if so, how close the value of each corresponding variable in the old component is to the respective value in the design problem component. As seen, the only difference between the two old components is that *Bottenplatta* has a closer value of the member variable $f_{st}$.

The *foot print match* investigates for every old component if the design calculation document of the current project contains the variables that the old component is dependent of and if so, how close the value of each old corresponding variable is to the value of the matching variable in the current project. The two components had the following footprint variables:

*Bottenplatta:* $q_{NLk}$(Live load) and $S_L$ (Centre distance of the columns along the length).

*Bottenplatta KK:* $q_{NLk}$, $S_L$ and $S_B$ (Centre distance of the columns along the width).

Because the value of $q_{NLk}$ for *Bottenplatta* is more similar to that of the corresponding variable in the current component than the value for *Bottenplatta KK*, it gets a higher footprint match score. We can notice that the footprint variables in this example are of great importance and not taking them into consideration would dramatically weaken the performance of the retrieval process. The footprint variables are also acquired automatically, in other words, without extra work by the designer, which makes them even more valuable. The whole calculation document containing also the footprint variables can be found in (Johansson, 2000).

Using the different hierarchic levels of the headings makes it also possible to compare the aggregation relationship of the current problem component and the old components. This is performed by comparing the similarity of the parents and the children of the two components. For information on how this is performed, see (Johansson, 2000).

In most cases, when reusing a solution, i.e. information describing the solution, it is better to chose one used many times before because it is thus also, to some degree, quality controlled. If a solution has been used many times, each time there has been an opportunity to find (to correct) problems and defects on that solution. This is even more relevant for process information. Integrating this aspect as well into the match process is a task for the *commonality match*. In the example above, it can be seen that the component Bottenplatta KK has been used more times (twice) than the component Bottenplatta (once). For a more detailed description, see (Johansson, 2000).

In conclusion, we can state that using the concept/component approach together with the simple process description makes it possible to match automatically the similarity of members, relation (classification, aggregation, and association - dependency), the design process, and the commonality of the components.

### 4.4.1 Adaptation and Evolution of the Design Process

After the retrieval, the designer can choose to reuse the values of the goal state variables from the retrieved old cases by copying the values from the Match Result-window into the current calculation. Since information about the process is available in the form of calculations, the designer may also reuse the calculation process of an old component. This is done by selecting a suitable old component in the Match Result-window and clicking on the OK-button. Having the detailed match information above, the designer in this example has chosen to reuse the component with the best match score - *Bottenplatta.*

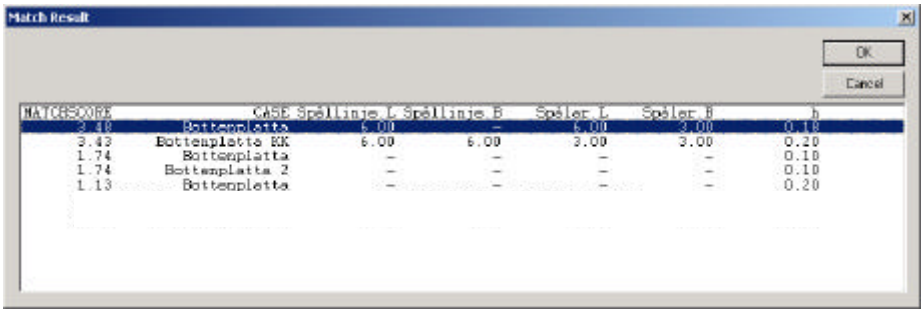


*FIG.13: Selection of old component to reuse.*

Then, ARCADE will create a Mathcad-file containing the calculations of the old component chosen and this file will be placed in the same directory as the current design calculation document. The designer can now include this component into the design calculation document of the current design.

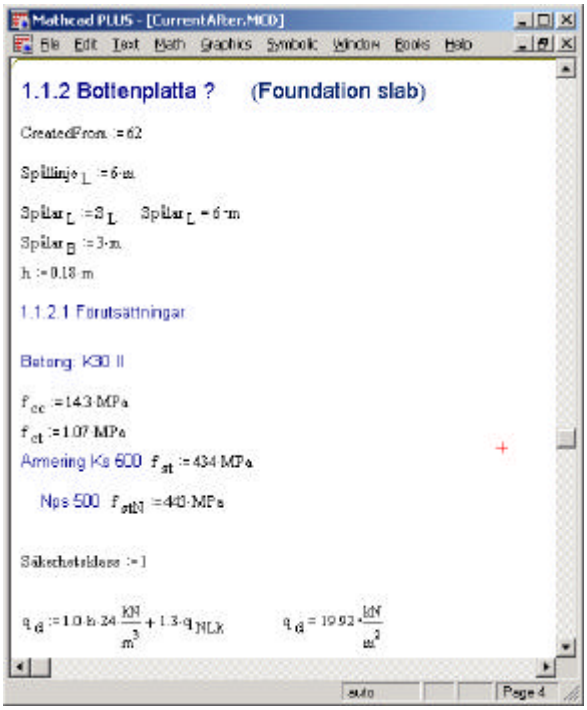


*FIG.14: Reuse of calculations from the old component.*

The calculations of the old component are now automatically adapted using derivational replay, that is, the design process described in the design calculations is recalculated using the values on the member variables and the footprint variable from the current design.

In this case, the calculations could be used without changes. In most cases, it is however required that the designer evaluate and, if necessary, revise them before they can be reused. This also provides an opportunity to evolve the calculations and make them even more reusable. For an example on this, see (Johansson, 2000).

## 4.5 Evaluation and Discussion

The implementation of the prototype shows that the approach described in this paper is feasible. The example, in turn, shows how the approach makes possible the reuse of weakly structured information in the form of design calculation documents. The concept/component approach together with the simple process description supports the automatical acquision and representation of weakly structured information and matching of the similarity of members, relations (classification, aggregation, and association/dependency), the design process, and the commonality of the components. It also allows us to adapt old components to the new situation using derivational replay. It can, in this way, help designers find solutions to design problems and reuse these solutions, and if available also the design process leading to the solution, with a minimum of extra work.

It is stated, in this paper, that a conceptual framework is defined using the text in the headings and the names of the variables. It has also been stated that different designers not always use the same notation i.e. the conceptual frameworks are designer-specific to some extent. This makes it sometimes hard to find relevant old components from different designers. This weakness is helped, to some degree, by the fact that reuse in itself uniforms the conceptual frameworks in such a way that the notation used in the reused components will be also used by the reuser, see (Johansson 2000). But this is probably not enough and in order to use the full strength of CBD, a standardized conceptual framework has to be introduced.

In the line of the evaluation, the prototype with examples has been shown to practicing structural engineers. Their view was that the search of old design solutions/processes is not a minor part of the design work, implying that the prototype would be of great use and lead to time-saving. The designers at first pointed out the need for quality control but after a discussion did conclude that reuse also creates an opportunity to find problems and defects, which in turn could result in calculations with fewer errors. Although old design calculations are reused

today to some degree, it is emphasised that the search of old solutions is mainly focused on drawings. It is also concluded that details have been the main goal for this search.

In order to be totally certain of the benefits of CBD, it has to be tested in practice where time-saving and errors of the artefacts produced could be measured and compared before and after the use of CBD. Such a study was not included in this research effort, however.

## 5. CONCLUSIONS

It has, in this paper, been argued that case-based design (CBD) systems should use the information created by the designer during the design process. The design information is today mostly in the form of weakly structured information. This paper has suggested an approach called the concept/component approach and a simple process description in order to enable capturing and representation of weakly structured information in case-based structural design. It has been shown that this allows us to automatically acquire and represent weakly structured information, and match the similarity of members, relations (classification, aggregation, and association/dependency), the design process, and the commonality of the components. It also makes it possible to adapt old components to the new situation using derivational replay. It is stated, in this paper, that a conceptual framework is defined using the text in the headings and it is shown how this can be captured. Although promising, issues regarding the building and maintaining of such a framework are indicated as the main weakness of the approach.

## 6. ACKNOWLEDGEMENTS

The work described here was financed by the national R&D programme “IT Construction and Real Estate” (ITBoF2002) together with the consulting engineering firms: J&W Byggprojektering, Skanska Teknik, NCC Teknik, Scandiaconsult, and FB Engineering. The design information used in this work has been gathered from: J&W, NCC Teknik, Bengt Johansson & Co, Göteborgs Byggprojektering, and BLOCO. All this support is hereby gratefully acknowledged.

## 7. REFERENCES

Abdalla J.A., Phan D.H.D., Howard H.C. (1991) Form, Function and Behavior in Structural Engineering Knowledge Representation. Artificial Intelligence and Structural Engineering, Topping, B.H.V. (Ed), Civil-Comp Press, Edinburgh, pp 1-9.

Björk B-C. (1996) Lecture notes from the lecture Product Models.

Burkhard H-D. (1998) Extending some Concepts of CBR-Foundations of Case Retrieval Nets, in Lecture notes in Artificial Intelligence, Lenz M., Bartsch-Spörl B., Burkhard H., Wess S. (Eds), vol. 1400, Springer, Berlin, pp.17-50.

Carbonell J.G. (1986) Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In Machine Learning, An Artificial Intelligence Approach, vol. II, Mikalski, Carbonell and Mitchell (Eds) Morgan Kaufmann, Boston.

Flemming U. and Woodbury R. (1995) Software environment to support early phases in building design (SEED). Journal of Architectural Engineering vol 1. no. 4, pp. 147-152.

Fischer G. (1994) Domain-Oriented Design Environments. In Automated Software Engineering, Kluwer Academic Publishers, Boston, MA, pp. 177-203.

Garrett J.H. Jr., Maher Hakim M. (1992) Object-Oriented Model of Engineering Design Standards. Journal of Computing in Civil Engineering, vol. 6, no. 3, pp. 323-347.

de la Garza J.M., Alcantara P.T. (1997) Using Parameter Dependency Network to Represent Design Rational. Journal of Computing in Civil Engineering, vol. 11, no. 2 , pp. 102-112.

de la Garza J.M., Oralkan G.A. (1995) Using Design Intent for Interpreting Brand-Name-or Equal Specification. Journal of Computing in Civil Engineering, vol. 9, no. 1, pp. 43-56.

Gero J.S., (1990) Design Prototypes: A Knowledge Representation Schema for Design. AI Magazine, vol. 11 no. 4, pp. 26-36.

Johansson P. (2000) Case-Based Structural Design –using weakly structured product and process information. (PhD thesis), Chalmers University of Technology, Division of Steel and Timber Structures, Publ. S 00:7, Göteborg, Sweden.

Kolodner J.L. (1993). Case-Based Reasoning. Morgan Kaufmann, San Mateo, CA.

Larman C. (1997). Applying UML and Patterns. Prentice Hall, Upper Saddle River, NJ.

Maher M.L., Balachandran M.B., Zhang D.M. (1995) Case-Based Reasoning in Design. Lawrence Erlbaum.

Maher M.L., Pu P. (1997) Introduction to the Issues and Applications of Case-Based Reasoning in Design. In Issues and Applications of Case-Based Reasoning in Design, Maher M.L., Pu P. (Eds), Lawrence Erlbaum Associates, Mahwah, NJ, pp. 1-10.

Maher, M.L. (1997) SAM: A multimedia case library of structural designs. In Y-T Liu, J-Y Tsou, and J-H Hou (eds) CAADRIA '97, pp 5-14.

MathSoft (1995) Mathcad User's Guide Mathcad 6.0. Cambridge, MA, MathSoft, Inc.

Mitchell T.M. (1997) Machine Learning. McGraw-Hill.

Moore, C.J. (1993) Complementary Innovative Computer Systems for Bridge Design. EG-SEA-AI Workshop: Application of Artificial Intelligence in Structural Engineering, Lausanne, Switzerland, pp. 2-14.

Mostow J. (1989) Design by Derivational Analogy: Issues in the Automatic Replay of Design Plans. Artificial Intelligence, vol. 40, no. 1-3, pp. 119-184.

Ndumu D.T., Tah J.M.H. (1998) Agents in Computer-Assisted Collaborative Design. In Artificial Intelligence in Structural Engineering, Smith I. (Ed) Springer, Berlin, pp. 249-270.

Qian, L., Gero, J.S. (1996) Function-Behaviour-Structure Paths and Their Role in Analogy-Based Design. AIEDAM, vol. 10, no. 4, pp. 289-312.

Raphael B. (1995). Reconstructive Memory in Design Problem Solving. (PhD thesis), University of Strathclyde, Glasgow.

Raphael B., Kumar B., McLeod I.A.M. 1994: Case Based Design Based on Methods. EG-SEA-AI Workshop: Application of Artificial Intelligence in Structural Engineering, Lausanne, Switzerland, pp. 305-317.

Rivard H., Fenves S.J. (2000) A Representation for Conceptual Design of Buildings. Journal of Computing in Civil Engineering, vol. 14, no. 3, pp. 151-159.

Schank R. (1982) Dynamic Memory: A Theory of Learning in Computers and People, Cambridge University Press, Cambridge.

Schank R.C. (1999) Dynamic Memory Revisited. Cambridge University Press, Cambridge.

Simoff, S., Maher, M.L. (1998) Ontology-Based Multimedia Data Mining for Design Information Retrieval, Proceedings of ACSE Computing Congress, Cambridge.

Turk Z. (1998) On Theoretical Background of CAD. Lecture Notes in Artificial Intelligence, 1454, Springer, pp. 490-496.

Veloso M.M.M. (1992) Analogical Reasoning in General Problem Solving. (PhD thesis), School of Computer Science, Carnegie-Mellon, University, Pittsburg, PA.