Software Reuse and Knowledge Transfer through Analogy and Design Patterns

Chung-Horng Lung[a], Gerald T. Mackulak[b], and Joseph E. Urban[c]

[a]*Department of Systems and Computer Engineering*
Carleton University, Ottawa, Canada
Tel: (613) 520-2600 ext. 2642
Fax: (613) 520-5727
chlung@sce.carleton.ca

[b]*Department of Industrial Engineering*
Arizona State Univeristy, Tempe, AZ
mackulak@asu.edu

[c]Department of Computer Science & Engineering
Arizona State University, Tempe, AZ
joseph.urban@asu.edu

Abstract:

This paper addresses some crucial aspects in analogy and presents applications of those concepts to software reuse and knowledge transfer in the manufacturing domain. Analogical reasoning deals with the transfer of knowledge from a well-known problem to a new problem. Analogical reasoning is closely related to software reuse and knowledge management. Software reuse is the application of existing software artifacts to another problem during the process of software development. Knowledge management talks about how to organize, update, share, and use the knowledge. Design patterns capture the recurring software solution to a problem. The concept and the objective of design patterns are similar to analogy and knowledge transfer. This paper presents some theories and empirical studies reported in analogy and successful experiences in design patterns, which provide insights from which we can learn to support software reuse and knowledge management.

Key words: analogy, software reuse, case-based reasoning, design patterns, OO modelling, knowledge management

1. INTRODUCTION

Analogical reasoning is critical in problem solving and is fundamental in learning and cognitive development, because analogy is a key factor in hypothesis formation, explanation, and the definition of abstract concepts [1,2,6]. Many analogy theories have been proposed to solve the problem A:B::C:X. Given that A is related to B as C is related to some X, find that X. For example, car:engine::gear:tooth.
Software reuse is similar to analogical reasoning in some respect. To reuse an existing solution for a new problem, we are essentially solving a problem like p(X):s(X)::p(Y):s(X'), where problems p(X) and p(Y) are identical or analogous, and solutions s(X) and s(X') are identical or similar. It may or may not be easy to identify the similarities and differences between p(X) and p(Y). Software reuse may occur across applications or even across problem domains. Software reuse can happen within an organization or across organizations in a company. Software reuse can also be at different levels. Examples include higher level software artifacts like software architectures or at the component level such as design patterns. The main challenge is to identify if p(X) or some subset of p(X) is analogous to p(Y) or some subset of p(Y).

Knowledge management deals with the way of how to capture, organize, update, share, and use the knowledge. Knowledge is considered a crucial resource of an organization. Knowledge management is closely coupled with analogy in that both areas discuss the transfer of knowledge from one problem (base) to another problem (target). The challenge of knowledge management is to systematically manage knowledge assets within an organization.

Another related area that is worth mentioning from the perspective of reuse and knowledge transfer is patterns. Software developers have been enthusiastic about patterns, especially design patterns [4]. Design patterns have been successfully applied to transfer design knowledge even across organizations. The success, in many cases, even exceeds the degree of knowledge transfer within an organization.

The objectives of this paper are first to present some important discoveries and theories in analogy. Following that, a brief comparison between analogical reasoning and software engineering is addressed. The comparison includes ideas that are common in both disciplines and areas that are primarily discussed in analogy. Next, based on the analogy methods, we demonstrate an empirical study in the manufacturing problem domain. The study shows reuse of software artifacts from the discrete manufacturing domain to continuous manufacturing problem area. The development time for the target problem artifacts is significantly reduced by transferring existing knowledge from a well-known problem. The result suggests that analogy may be a useful paradigm to support software reuse or knowledge transfer. We also highlight some lessons learned in analogy that software engineering and knowledge management may explore further in the future. We will also discuss the simple but practical factors of success at a higher level view in hope that knowledge management within an organization can benefit from.

2. ANALOGICAL REASONING

Discoveries and theories have been intensively discussed in analogy [1,2,3,5]. Due to the space limit, we only briefly present some critical aspects of analogy. Readers can refer to the references for more discussions.

Basically, the analogical reasoning consists of several phases as shown in Figure 1. The process is carried out in an iterative and incremental manner rather than in strict sequential manner. Moreover, these steps may be performed in parallel or may interact with one another in various ways. For example, a partial mapping between the source and the target is necessary for the retrieval of an appropriate source.

Based on the experimental studies reported in analogy, representation plays a key role in identifying analogous problems or solutions. Similarity judgments can be made more efficiently and effectively if there exists suitable underlying representations. Moreover, good representation schemes can support all other phases shown in Figure 1. Rich representation must include syntactic, semantic, and pragmatic components [Gentner 89, Kedar-Cabeli88, Holyoak89].

Another frequently discussed area in various analogical reasoning theories is relations [1,2,3,5]. Relations between concepts, not just the representation of individual concepts, are the key in identifying analogy [1]. The relational modeling emphasizes two aspects: higher-order relations and classification of semantic relations.

Take the idea of higher-order relations first. The central idea in Gentner's structure-mapping theory [3] is the principle of systematicity, which states that analogy is a mapping of systems of relations governed by higher-order relations with inferential import, rather than isolated predicates. Higher-order relations demonstrate the relation of relations. Scientific analogies were presented based on the principle.

Take the analogy between the structure of our solar system and the atom system as an example. Several relations between the sun and a planet include distance, attract, more-massive, revolve-around, and hotter-than. Together, distance, attract, more-massive, and revolve-around form a higher-order relation:

> CAUSE [distance(sun, planet), attract(sun, planet), more-massive(sun, planet), revolve-around(planet, sun)]

This higher-order relation can be mapped to the atom system. Some solutions for the solar system can then be mapped to the atom system. Isolated relations, such as the sun is hotter-than the planet, are discarded in the mapping phase.

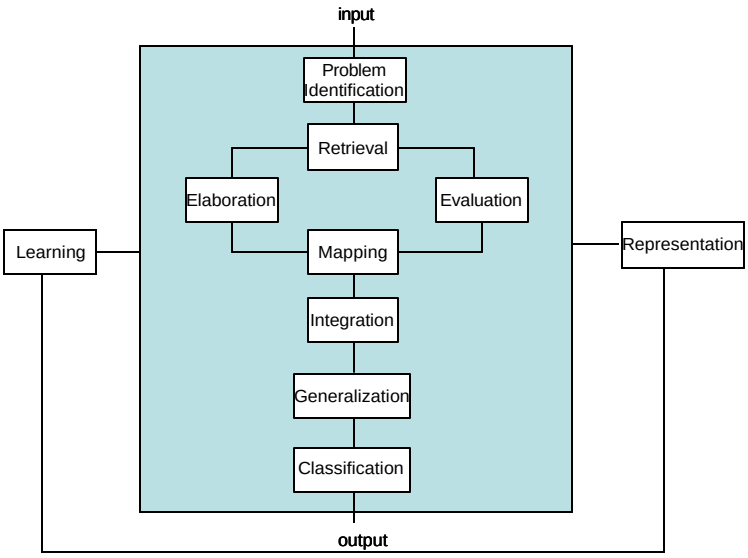

Figure 1. Dimensions of analogy

The ability to readily compare relations means that relations are readily decomposed into more primitive elements [2]. People readily compare relations. This requires that relations can be decomposed into aspects in which they are the same and aspects in which they differ. Bejar et al. [1] then presented taxonomy of the semantic relations. With the taxonomy, relations can be decomposed into common aspects that the systems are sharing and aspects that the systems differ.

Table 1 shows the taxonomy which consists of ten classes. There are two semantic types: intensional and pragmatic. These ten classes are further divided into more specific members. For instance, two members in the part-whole class are

Object : Component – car : engine; and
Mass : Portion – water : drop.

Not all classes or members are directly related to software engineering problems. However, two classes that are closely related to object-oriented methods are class inclusion (Is-A) and part-whole (Has-A). Section 3 will further discuss software engineering practices and the key methods in analogy just described.

Table 1. Semantic Types of Relations and Examples

Semantic Type	**Semantic Class**	**Example**
Intensional	Class Inclusion	robin:bird
Intensional	Similarity	breeze:gale
Intensional	Attribute	begger:poor
Intensional	Contrast	stutter:speech
Intensional	Nonattribute	harmony:discordant
Pragmatic	Event	tailor:suit
Pragmatic	Cause-purpose	virus:illness
Pragmatic	Space-time	aircraft:airport
Pragmatic	Part-whole	engine:car
Pragmatic	Representation	building:blueprint

3. APPLICATION OF ANALOGY TO SOFTWARE REUSE AND KNOWLEDGE TRANSFER

This section presents an application of analogy to software engineering and knowledge transfer. Specifically, those areas are the dimensions of analogy and reuse, modeling of higher-order relations and analysis of semantic relations. The approach is not totally new. Analogy has been applied to software engineering [7-9]. An empirical study in reusing software artifacts from the discrete manufacturing domain to continuous manufacturing area is presented.

Representation phase, as shown in Figure 1, has been widely discussed in analogy. Based on the experimental studies reported in analogy, representation plays a key role in identifying analogous problems or solutions.
Similar to Figure 1, dimensions of software reuse has also been discussed in the software community. The paper describes the following phases: abstraction, identification, selection, specialization, and integration Krueger [5]. Here, we focus on the representation issue.

As described previously, a rich representation should encompass syntactic, semantic, and pragmatic components. The software community also echoes this point. Generally, in modeling software, various viewpoints are advocated. These viewpoints include these three types of components; some explicitly, some implicitly.

Lung and Urban [7] presented an approach that consists of object modeling, functional modeling, relational modeling, and dynamic modeling. Object, functional, and dynamic modeling are similar to object-oriented methods. Specifically, the main artifacts produced from those modeling phases include entity relationship diagram, data flow diagram, functional descriptions, and a rule-based Petri net representation. Relational modeling is mainly concerned with the identification of high-order causal relations, and reasoning and classification of semantic relations. We highlight some important points and present an application on the transferring of knowledge from a well understood domain to a relatively new problem.

We start with a comparison of the main components between the discrete and continuous manufacturing, as shown in Table 2. Both domains have similar material handling systems (MHS), whose primary operation is to move and store materials, parts, and products. In a discrete problem, there are usually a group of machine stations each performing some simple operations such as machining, assembling, or inspecting materials or products. While in a continuous problem, the number of machines is fewer, but generally each machine is costly and performs more complex operations. Disassembling and chemical processing are two typical examples in the continuous domain. Because the stations are more sophisticated in the continuous domain, planning for maintenance and equipment failure is more important than in the discrete domain.

Another significant difference is the scheduling process due to the difference in product type and machine stations. As a result, simulation models needed for these two domains are also different. In the discrete domain, finite state machines and discrete event models are commonly used. In the continuous domain, difference equations or differential equations are needed [12].

Table 2. Comparisons of Main Components between Discrete and Continuous Manufacturing

	Discrete	**Continuous**
Material Handling	Transport and store material	Very similar to discrete domain
Stations	Many, one operation / station, queue before	Few, many operations / station, no queue
Product Type	Usually mixed, many could also be 1 product	Product type disassembly (one main product, several sub-products)
Queue	Have queues	Normally no queues
Equipment Failure	Same process, minor disturbance, fast recovery, possible alternative	Scarp in process, major disturbance, high reliability required
Planned Maintenance	Should do, but often skipped	Very important, hard to stop, more regular
Scheduling	Math difficult, little optimization, predicable, a priori, finite state machines	Sequencing, continuous monitoring (real time), response time, high variance, differential equations
Sensors	Fixed position sensors, flag when an object moves into a range	Normally no sensors

Based on above comparison and observations, and the object model for discrete manufacturing, an entity relationship diagram (ERD) for the continuous domain is derived. A simpler ERD is illustrated in Figure 2. As a result, entities queue and sensor and associated relationships are removed for the target domain.

The main artifact for the functional model is data flow diagrams. For brevity, only a data flow diagram (DFD) for the continuous manufacturing is illustrated, as shown in Figure 3. The seven processes are derived from the base, discrete manufacturing domain. Processes 1, 4, 5, 6, and 7 are repeated use without modifications. Process 2 is a modification from "monitor the position of all products and identify collisions". Process 3 is changed from monitoring the product position to modifying the product quality. A queue process and sensors entity, and all related entities, data stores, and data flows are removed, since normally there are no queues or sensors in the continuous problem. Both the ERD and DFD for the continuous manufacturing were developed in hours as opposed to days for the discrete manufacturing.

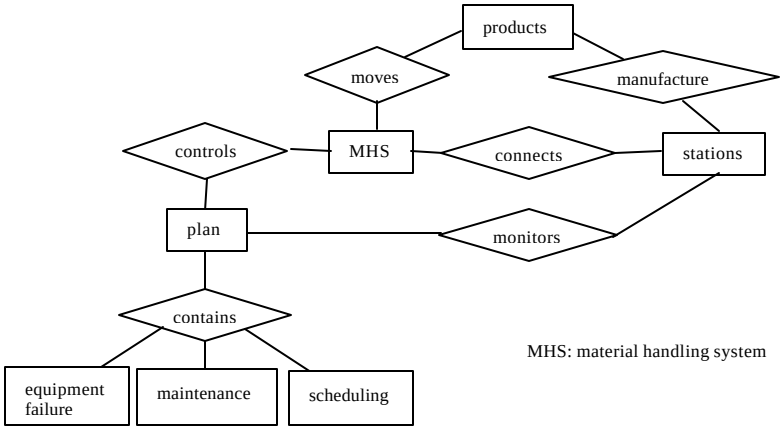

Figure 2. Entity relationship diagram for the continuous manufacturing

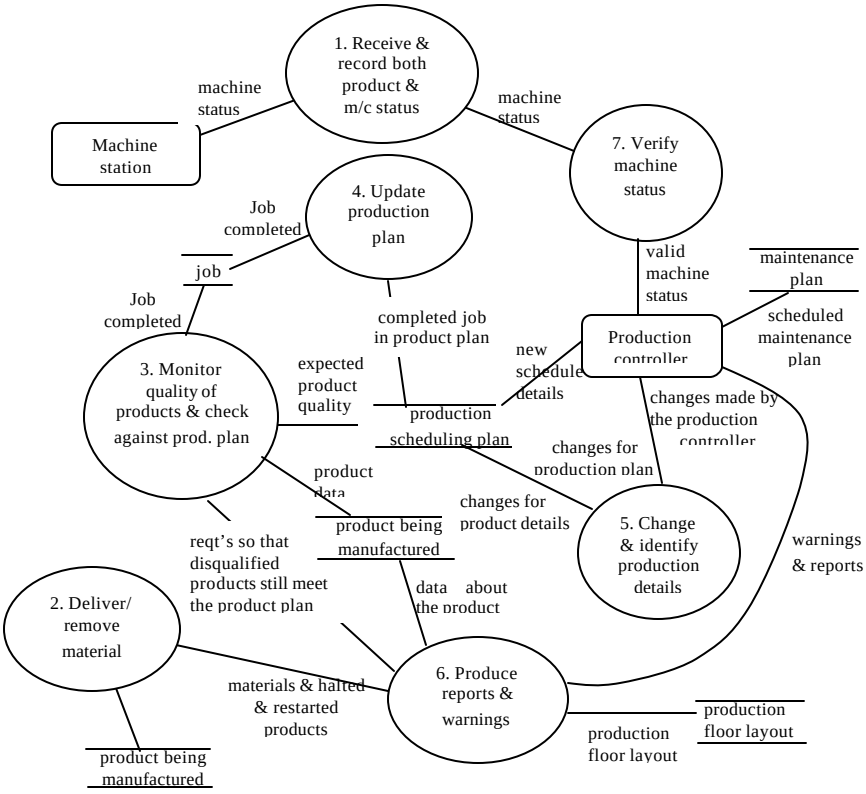

Figure 3. Data flow diagram for the continuous manufacturing

Next, we discuss the modeling the high-order relations and dynamic modeling. High-order relations are related to dynamic modeling. Dynamic models capture more detailed and more specific information. Modeling of higher-order relations, on the other hand, conveys high-level cause-effect information. Currently, there is no appropriate technique or formal mechanism for modeling higher-order relations, i.e., relations of relations, as proposed in analogical reasoning. In traditional OO modeling, we usually capture relations between classes or objects. Those types of relations represent low-order relations. Figure 4 and 5 show the causal structure for both problems. The arrow is used to show causation. The item at the tail causes a change in the item at the head [13]. The structure for the continuous problem is developed based on that of the discrete domain.

The dynamic model is represented in rule-based Perti-net format [7]. The rules are detailed descriptions of those cause-effect relations depicted in the causal structure. The idea is similar to the dynamic modeling in OO modeling languages like UML [10]. Features in UML to model behavioral diagrams, e.g., use case diagram, sequence diagram, and collaboration diagram, represent collection of relations, which is conceptually similar to the structure of relations proposed in analogical reasoning. However, many OO applications focus on the solution space. To be effective in reuse or knowledge transfer across applications or even across domains, the problem space should be explicitly and clearly modelled. Moreover, we also need to represent the problem

space beyond objects and lower-order relations. Improvements in the modeling and representation will facilitate the identification of similarities and differences between p(X) and p(Y) as stated in the introduction.

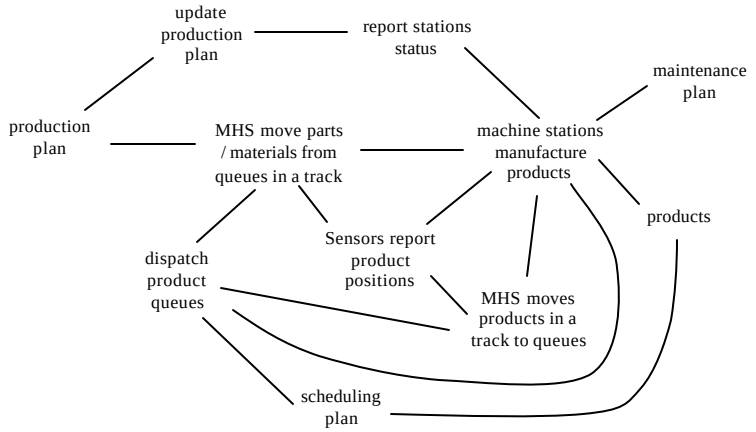

Figure 4. Causal structure for discrete manufacturing systems

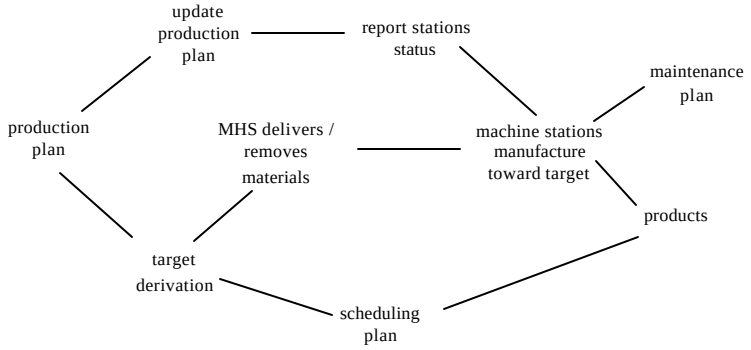

Figure 5. Causal structure for continuous manufacturing systems

Lung and Urban [7] presented an example that is relevant to the classification of semantic relations for the main components. The example deals with the library problem and the car rental problem. In the example, the different types of relations between the (borrower:book – Agent:Object relationship) and (borrower:car – Agent:Instrument relationship) stimulate more careful examinations for the target car rental problem. As a result, an additional requirement, corporate user, was identified for the car rental problem, which did not exist in the existing library problem (base) in the study.
Among the semantic relations, class inclusion (Is-A) and part-whole (Has-A) which are widely used in OO modeling. Bejar et al. [1] listed five members for the Is-A class and ten members for the Has-A class. Readers can refer to [1] for detailed discussion and examples.

For this case study, part-whole relations are examined in more detail, since the relations are common in the discrete manufacturing. For instance, an engine is part of a car.

Specifically, car:engine relation is in the Object:Component category [1]. However, there are differences beyond the part-whole level. For the continuous manufacturing, there also exist part-whole relations, but they fall into the Mass:Portion category as in the milk:skim milk example. The distinction between the two problems in terms of the semantic relation for two components direct us further investigation. As a result, two different types of machine stations and control processes are needed for these two domains, albeit these two domains share a similar higher-order relation.

Part-whole relationships are widely used in OO modeling. There is confusion about the relationship [11]. In UML, composition is advocated as a special form of aggregation within which the parts are inseparable from the whole. The idea of separating composition from aggregation is similar to the classification of semantic relations. However, there exist more differences than just aggregation and composition. For example, car:engine and department:company are different even though they both share part-whole relationship. In the first case, car has one engine and only one (in most cases). Other parts, e.g., transmission, are very different from engine. However, departments within a company share many similarities. In [1], both examples share part-whole relationship, but car:engine is classified as Object:Component and company:department belongs to Collection:Member.

Empirical evidences support that comparison and evaluation of relations is a commonly used and effective approach to judge similarities and differences. The exercise helps the mapping diagrammed in Figure 1, so that we can map relevant relations and make necessary adaptations more effectively.

4. DESIGN PATTERNS AND KNOWLEDGE TRANSFER

The case study presented above demonstrates the potential of learning by analogous examples. The case study is successful in that it facilitates the development of the artifacts for the target problem much effectively. However, this study is a bit opportunistic. Some phases shown in Figure 1 that may hinder the analogy mapping are not addressed here. For instance, identification and retrieval of a similar problem and the capture of the learning process of the practices. Those phases are implicit in this study. A related area that has drawn a lot of attentions and practices is patterns. This section highlights some relevant lessons that we learn from patterns community to support knowledge transfer and management.

Patterns, at various levels of abstractions, have been successfully applied to many applications; some are in the same problem domain, while others fall into different problem areas. A design pattern [4] usually is described in a "standard" template, which consists of problem, context, structure, system dynamics, variants, and example sections. The format is useful in problem identification and other phases depicted in Figure 1. In fact, the format also includes critical areas that analogical reasoning advocates.

The concept of design patterns is similar to analogical reasoning, except that design patterns usually deal with more focused problems than that of analogical reasoning. A

design pattern is a solution, s(X), to a set of problems, p(X1), p(X2), …., p(Xn). Those problems may or may not in the same problem domain, but they share common structures, problem context, and system dynamics.

Design patterns are relative new, but its success actually exceeds the outcome of knowledge transfer in many companies, especially large companies. Patterns published and relevant discussions in the community further support the identification and learning of the patterns. Some of the lessons that we can learn from patterns to facilitate knowledge management or transfer include:

- *Independent organization.* A virtual organization, i.e., patterns community, continuously publishes and evolves empirical studies and examples, and serves as a reusable library.
- *Open environment.* Knowledge in patterns is reviewed by peers and organized in an open environment. The reusable library integrates formal and informal, certain and uncertain (context dependent) knowledge.
- *Infrastructure.* Web and internet technologies provide an excellent infrastructure for discussions, sharing, and retrieval of knowledge assets.
- *Learning.* The community provides a continuous learning organization. Various ways for discussions, e.g., conferences and discussion groups across organizations, have been formed. On the contrary, it is rare in many companies.
- *Concrete examples.* Lots of examples are demonstrated in patterns. Examples are valuable stories, which also are a key factor to analogy.
- *Explicitness.* Implicit knowledge is explicitly captured and documented.
- *Format.* The format used in patterns is simple and yet it captures three basic ingredients proposed in analogy: syntactic, semantic, and pragmatic information to some extent. The format also supports higher-level of abstraction. The corporate knowledge also needs to be stored in a more abstract form rather than project-specific representations.
- *Specifics.* Some patterns could be used across problem domains, but patterns are more successful in specific problem areas. This is an advantage to organizations, since they are dealing with specific areas. This feature can greatly reduce the time needed to identify an analogous problem, which actually is a barrier in general for analogy.
- *Usefulness.* The knowledge is useful for the designers. Perhaps, more appropriately, the actual peer designers think that the knowledge is useful.
- *Recognitions.* Recognitions are realized and reputations are established through publications. Company may not provide incentives to publish knowledge assets or to encourage employees to share knowledge.
- *Cooperation.* People cooperate actively. Usually, questions posted in community are answered quickly. On the contrary, departments within a company may not cooperate as well for various reasons.

5. CONCLUSION AND FUTURE DIRECTIONS

This article presented an analogy-based approach to support software ruse and knowledge transfer. A study on manufacturing was demonstrated. The development time of the software artifacts was significantly reduced for the target problem as a result of the knowledge transfer. We also highlighted lessons that we can learn from analogical reasoning and design. Analogical reasoning presents discoveries that we could adopt or enhance OO modeling. Examples include modeling of higher-order relations,

comparison and classification of semantic relations, and other supports to the phases shown in Figure 1.

Software engineering is a people- and knowledge-intensive business that would benefit from the reuse of past experience. Other relevant methods reported in analogical reasoning may also be applied to software engineering. Finally, evidently successful experience in design patterns provides encouraging results from software reuse and knowledge transfer perspectives. The concept of patterns is similar to that of analogy. One main advantage of patterns in general is their specifics. Being specific reduces the time for problem identification stage, which usually is difficult and requires intensive knowledge. Identification of an analogous problem is given or ignored in some analogy methods. In reality, problem identification plays a vital role to facilitate knowledge transfer and management. Those success factors collaboratively learned in patterns community are valuable and could be adopted to support reuse and knowledge transfer within an organization.

References:

1. I.I. Bejar, R. Chaffin, S. Embretson, *Cognitive and Psychometric analysis of analogical problem solving*, Springer-Verlag, 1991.
2. R. Chaffin and D. Herrmann, "Relation Element Theory: a New Account of the Representation and Processing of Semantic Relations", *Memory and Learning: The Ebbinghaus Centennial Conf.*, 1987, pp. 221-245.
3. D. Gentner, "Structure-Mapping: a Theoretical Framework for Analogy", *Cognitive Science*, vol. 7, no. 2, 1983, pp. 155-170.
4. E.Gamma et al., *Design Patterns Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing, 1995.
5. C.W. Krueger, "Software Reuse", *ACM Computing Surveys*, vol. 24, no. 2, June 1992, pp. 131-183.
6. D. Helman, ed., *Analogical Reasoning*, Kluwer Academic Publishers, 1988.
7. C.-H. Lung and J.E. Urban, "An Expanded View of Domain Modeling for Software Analogy", *Proc. of Int'l Computer Software & Applications Conf*, 1995, pp. 77-82.
8. N.A.M. Maiden and A.G. Sutcliffe, "Exploiting Reusable Specifications through Analogy", *Commu. of the ACM*, vol. 35, no. 4, 1992, pp. 55-64.
9. N.A.M. Maiden and A.G. Sutcliffe, "Requirements Engineering by Example: an Empirical Study", *Proc. of IEEE Int'l Symp. on Requirements Eng*., 1993, pp. 104-111.
10. Object Management Group, OMG Unified Modeling Language Specification.
11. A.L. Opdahl, et al. "Ontological Analysis of Whole-Part Relationships in OO-Models", *Info and Software Technology*, 43, 2001, pp. 387-399.
12. F. Pichler and H. Schwartzel, eds., *CAST Methods in Modelling: Computer Aided Systems Theory for the Design of Intelligent Machines*, Springer-Verlag, 1992.
13. N. Roberts, D.F. Andersen, R.M. Deal, M.S. Garet, and W.A. Shafeer, *Introduction to Computer Simulation*: *The System Dynamics Approach*, Addison-Wesley Publishing Company, 1983.