USING DESIGN PATTERNS, ANALYSIS PATTERNS AND CASE-BASED REASONING TO IMPROVE INFORMATION MODELING AND METHOD ENGINEERING IN SYSTEMS DEVELOPMENT

Sean Chen, Furman University
Bonnie W. Morris, West Virginia University

Abstract

Information modeling (IM) is the process of identifying information needs and models based on user requirements and systems analysts' perceptions during systems analysis and design. When IM is done correctly, it facilitates communication between the analysts and end-users about the final software product. In addition, successful IM provides a formal basis for both the analysts and the end-users about the tools and techniques that will be used in software development (SD), which, in turn, reduces costly overruns in time and money during systems implementation. *Method engineering* (ME) is the process of designing, constructing, and adapting information modeling methods for information systems development. As Siau (2003) and Kavakli (2005) point out that, while there has been a steady increase in IM and ME research (e.g. Kawalek & Wastell 2003, Kavakli 2005, Matulevicius 2005), most of the models reported in recent literature are still primarily based on common sense approach, and, as a result, lack a slid theoretical foundation.

This paper discusses the feasibility of combining design patterns (DPs), analysis patterns (APs) and case-based reasoning (CBR) to improve information modeling and method engineering. Recent research in DP, AP, and CBR has proven that all those methods are effective in software development. In this paper, we propose a model that combines DP, AP and CBR as a tool to improve IM and ME. We believe that the use of DP and AP, along with CBR will facilitate easier communication among systems analysts, end-users and software engineers thus improve on he efficiency in software development. In the paper, we also provide illustrative examples from accounting systems design to show the effectiveness of our proposed model. Finally, we provide evidence in this paper that the practical application of DPs, APs and CBR to systems development makes it possible to identify and resolve critical issues and risks at earlier stages in IM and ME, and eventually lead to high quality end product.

Keywords

design patterns (DP), analysis patterns (AP), case-based reasoning (CBR), information modeling (IM), method engineering (ME), software development (SD)

INTRODUCTION

Information modeling (IM) is the process of identifying information needs and models based on user requirements and systems analysts' perceptions about the information needs during systems

analysis and design1. It is the most important step in information systems analysis and design (Siau 2003). When IM is done properly, it generates models that bridge the gaps of understanding in systems requirements between the analysts and the end-users (Siau & Rossi 2001). Furthermore, sound IM provides information models that facilitate systems implementation in terms of adequate level of resource allocation and proper use of system tools and techniques. Therefore, a solid IM ensures a successful systems project.

Method engineering (ME) is the process of designing, constructing, and adapting information modeling methods for information systems development (Siau 2003). The development of information systems is a complex task. To help manage the complexity of the system development process, designers, analysts and programmers have developed a vast array of ME tools and methodologies. For example, CASE and Unified Modeling Language (UML) are tools to aid in design and implementation (Booch et. al. 1998 &1999, Siau & Cao 2003, Whittle 2003). These and other tools are intended to lead to well managed, well written applications, developed within budget, and delivered on time.

Ultimately, however, success of an information system is determined by how well it meets the user's needs. No matter how elegant the code, or how quickly and efficiently it was developed, a system that does not meet the users needs is a failure. As Cunningham (Fowler 1997, p. 7) states,

"We already have in our hands the machinery to build great programs.
When we fail, we fail because we lack experience."

Cunningham refers to the analyst's lack of domain experience in the information modeling (IM) and method engineering (ME) processes. Analysts who have been involved in the

1 Kavakli & Loucopoulos (2005) called such processes *Requirements Engineering* (RE), although the content, methods and examples of RE used in their paper are essentially the same as *Information Modeling* defined in Siau (2003). Likewise, Gjersvik et al. (2005) called the process *Enterprise Process Modeling* (EPM). Again, in this paper, we view EPM interchangeable with the process of *Information Modeling*.

development of a similar system can do a better job of eliciting user requirements in IM thus can design a system that better meets the users' needs because they already have developed better information models. During ME, they know about the key components and their interactions, about possible special cases and exceptions that users might fail to mention, and perhaps about design features that have failed in the past.

Knowing the importance of IM and ME in successful systems project, this paper discusses the feasibility and usefulness of incorporating analysis patterns (APs) and design patterns (DPs) into a case-based reasoning (CBR) system to support user-analyst communication in the IM and ME processes. As Siau (2003) and Kavakli (2005) point out that, while there has been a steady increase in IM and ME research (e.g. Kawalek & Wastell 2003, Kavakli 2005, Matulevicius 2005), most of the models reported in recent literature are still primarily based on common sense approach, and, as a result, lack a slid theoretical foundation. Therefore, we are motivated to propose a model that will provide a theoretical foundation that supports IM and ME. We will also provide several examples from accounting-based systems design to illustrate how our proposed model works in soliciting information models to facilitate systems implementation.

The remainder of the paper is organized in the following fashion: The next section discusses the current state of information modeling and method engineering and existing problems. Section 3 introduces our proposed model that applies design patterns (DPs) and analysis patterns (APs) in a case-based reasoning (CBR) setting. Section 4 provides the conceptual background of CBR, DP, and AP. Section 5 illustrates how our proposed model works in several systems design examples. And, finally in Section 6, we provide the conclusion and future direction of the research in patterns and CBR.

Information modeling and method engineering – CURRENT STATUS

One of the first and most important steps in system development is the system requirements analysis task. Effective analysis is required to ensure that designers have correctly defined the business problem and are developing an appropriate solution.

An earlier report on systems development that has been widely quoted is the CHOAS Chronicles research report by the Standish Group (1995). The Standish Group found that only 16% of all large system development projects were delivered on time and on budget, while 31% of all projects were canceled before completion. The average cost overrun was 189% and the average time overrun was 222%. Lack of effective user involvement and incomplete or ineffective systems analysis were cited as the top two factors leading to system failure. Nearly a decade later, the Standish Group (2003) reported similar results about systems development: among all systems projects, 34% of projects were successfully completed, 15% projects were total failures, while the remaining 51% are still "challenged" projects.2

Cushing & Romney (1997) reports the following four basic strategies for improving information modeling:

1. Asking users,
2. Analyzing the existing system,
3. Analyzing usage of the existing system, and
4. Experimentation.

Two factors that contribute to the effectiveness of each method are user experience and analyst experience. Users who lack experience with the current system and with the system

2 Although Standish Group reports on successful as well as failed systems analysis and development, in this paper, we view the process leading to successful systems analysis as a part of information modeling and method engineering.

development process are oftentimes less effective in specifying information models. Lack of specific problem domain experience also limits the effectiveness of the analyst in successfully applying method engineering to solicit user requirements (Cushing & Romney 1997, Siau & Rossi 1998). General system development experience does not seem to compensate for lack of domain experience.

Another problem system designer's face is that users may have difficulty distinguishing between reality (i.e., the actual business requirements which should be derived from organizational strategy) and the existing model of reality (i.e., the current information system). This leads to a tendency by users to describe the current needs in terms of "how we've always done it" rather than "how it should be." Proper IM should reflect current business needs. The existing system is merely an artifact or model of the business needs at a previous point in time. That artifact may reflect design compromises that resulted from hardware, software or other implementation constraints that are no longer valid information models in the present implementation.

Although development methodologies (e.g. Gjersvik et al. 2005) and textbooks (e.g. Cushing & Romney 1997) stress the importance of user involvement, in many cases, the user plays a very passive role. Users are viewed as repositories of information from which the analysts must extract, elicit or pull the information models through method engineering by conducting interviews, surveys, and observation of the existing system. The focus is on determining how the current system operates and its perceived weaknesses. The responsibility for conceptual design falls largely to the analyst. And, if the analyst lacks the experience in IM and ME, the developed systems tend to fail due to insufficient IM and inadequate ME.

Problems associated with the traditional method in IM and ME can be summarized in the following chart.

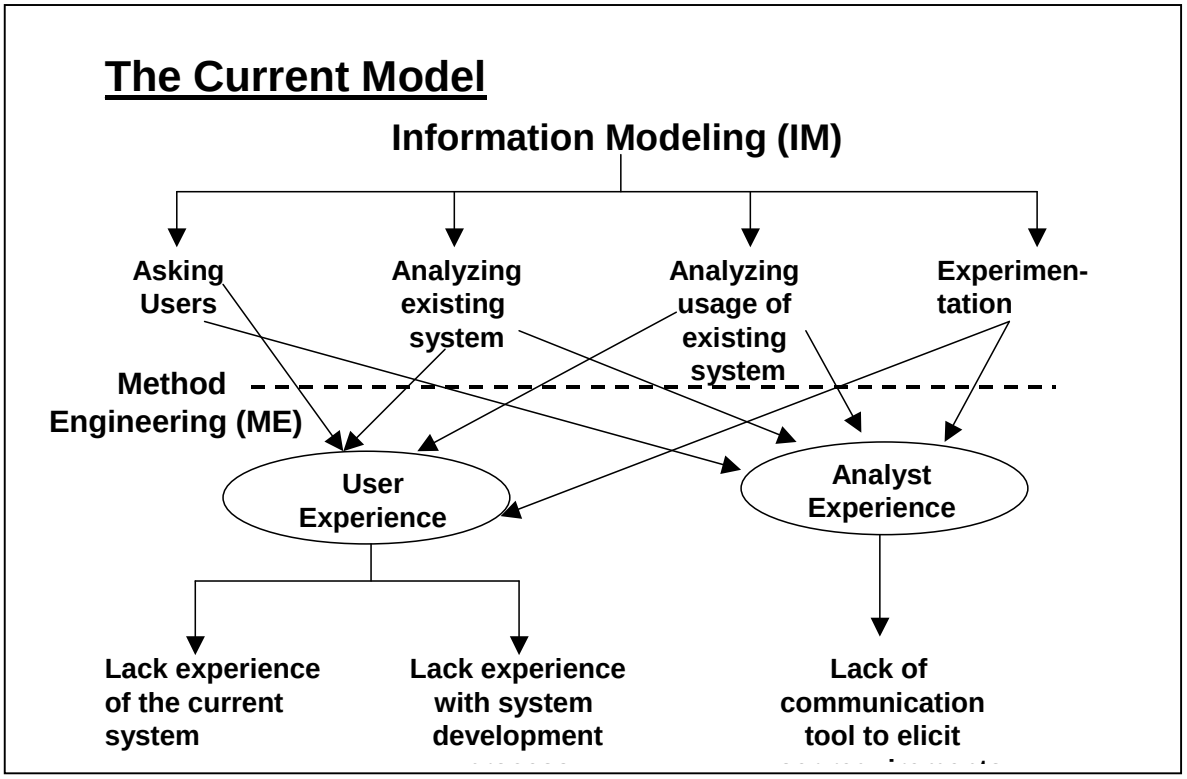


Chart 1: Information Modeling – The Current Model Applying DP and ap in a case-based reasoning setting

This research is motivated by the desire to improve communication between the user and the analyst in IM, and to provide the user with tools that will help compensate for lack of development experience in ME. We believe that the adaptation of design patterns (DPs) and analysis patterns (APs) in a case-based reasoning (CBR) setting will improve communication between the user and the analysts thus generate better information models. Our proposed model will also facilitate the solicitation and construction of information models thus improves the quality of ME. We will elaborate on how our proposed model works in the following:

Case-based reasoning (CBR) is reasoning by analogy to past cases or experiences. Humans utilize CBR in many aspects of their lives. Reasoning from past experience is a strategy that humans use for reducing cognitive load. It is much easier to recall a similar past case and adapt its

solution to solve a current problem than to reason "from scratch" for each new situation (Riesbeck & Schank 1989).

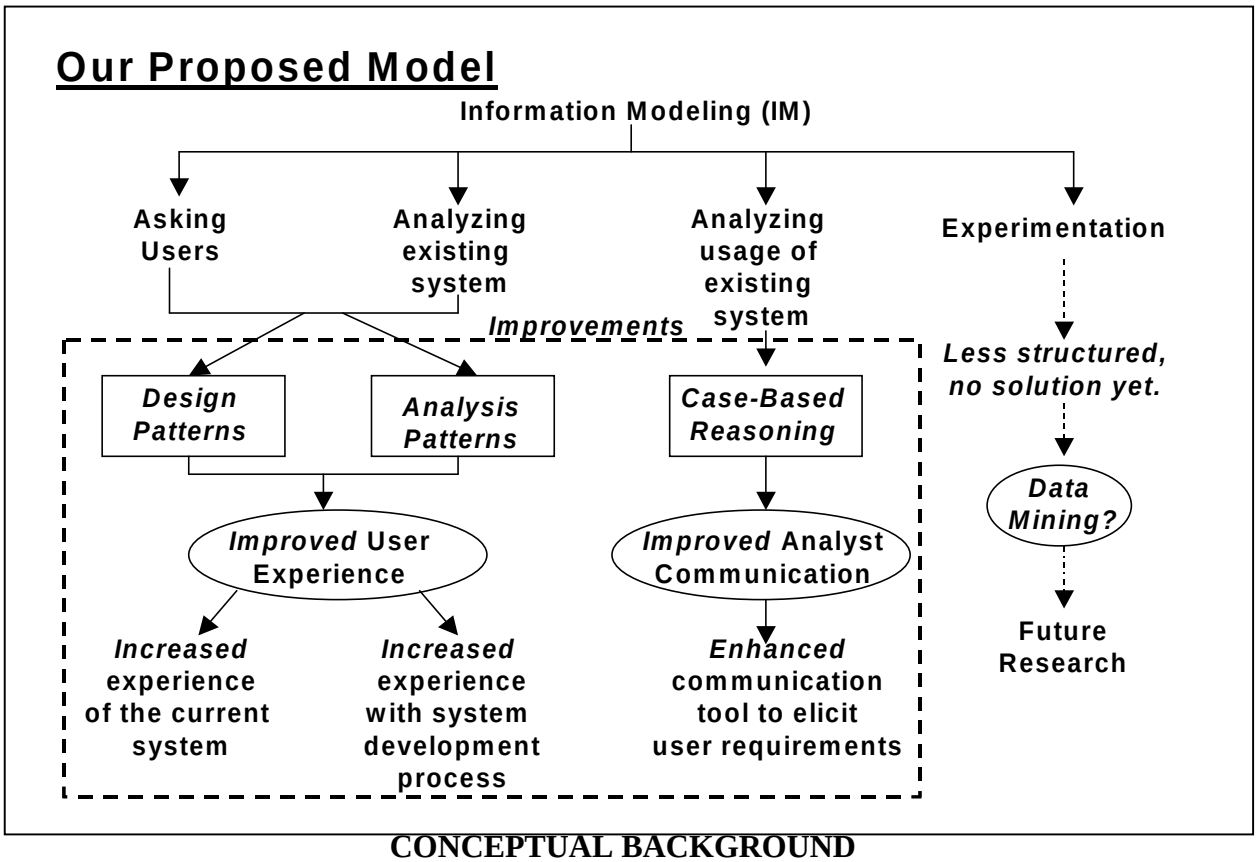
To compensate for the lack of experience, analysts, designers and programmers have begun creating Design Patterns (DP) (Gamma et al. 1995). DP is small, reusable chunks of program code that address recurring design problems. Naming and cataloging patterns provide a way for analysts and designers to communicate with the programmers during information modeling. Therefore, DP can be used to support system implementation.

Another type of pattern is referred to as analysis patterns (APs) (Fowler 1997). APs are developed business cases based on past experiences. They reflect recurring business process patterns, such as the “resource allocation pattern.” Fowler’s work on analysis patterns supports the work of the analyst during IM. AP can be used as a tool in ME that facilitates communication between analysts and designers after information models have been obtained from the user.

IM and ME are important steps in determining system requirement in software development. It is an area that is presently not directly addressed by APs and DPs. Better information models derived from better method engineering improve communication between the user and the analyst, which, in turn, lead to better system development. Communication between the analyst and user is hindered not only by the analyst's lack of domain experience, but also by the users' lack of development experience (Siau 2003). Users may participate in only a few development projects throughout their careers. Often, their involvement is rather passive and provides them with little understanding of the conceptual design process or the potential role they should play. Furthermore, even though occasionally users may have great depth of understanding about their organization's existing systems, they may still lack knowledge about alternative ways that similar applications

have been implemented in other organizations. Knowledge and experience with alternative implementations may aid business process reengineering efforts.

APs and DPs are specific instances of CBR that supplement the experience of analysts, designers, and programmers. The proposed CBR system discussed in this paper will extend the Analysis Pattern-Design Pattern paradigm by providing support for user-analyst communication during IM and ME processes.



CONCEPTUAL BACKGROUND

Case-Based Reasoning (CBR)

CBR is a problem-solving approach using memory of previous problem solving cases. CBR approach relies on analogy. It is often used in task domains that have no strong theoretical model and where the domain rules are incomplete, poorly defined and inconsistent (Ashley and Rissland 1987, 1988; Kolodner 1993). With CBR, problems are solved not by finding and applying the

knowledge of the most appropriate fundamental principles, but by finding and applying the knowledge of the most relevant and similar prior cases (Mukhopadhyay et al.1992).

Auditors regularly use prior cases in their financial judgments (Friedman 1995), particularly when they need to consult their national offices for information (Danos et al. 1989, Salterio, 1994). Salterio (1996) calls these prior cases "precedents", and defines them as "…prior examples of similar situations encountered in the firm's practice, documented in internal memorandum or by other audit firms as seen in the published financial statements disclosures…" It is known that Big Four firms encourage staff to search for relevant precedents before field work (Danos et al. 1989, Salterio 1994 & 1996). In this paper, we will use examples from auditing to show how our proposed model works in IM and ME.

A staff auditor who is assigned to audit a construction company, bank, or real estate holding company can retrieve a related precedent case to learn how business is conducted in that industry. The case may contain information about the industry's critical success factors, difficult accounting issues, areas of potential risk for errors or irregularities, and sample financial disclosures. The precedent cases provide the auditor with a mental model of business practices in the client industry. The precedent can be used as a standard against which the new client's business practices can be compared and contrasted. Providing precedents is one of the major services that the Big Five national offices offer to practicing auditors to enforce a consistent corporate policy (Cushing & Loebbecke 1986; Gibbins & Mason 1988).

Recently accounting, finance and IS research has shown considerable interest in developing CBR systems to support IM (Denna et al. 1992, Jung et al. 1999, Ku et al. 1996, Lee & Han 1998, Morris 1994, Rockwell & McCarthy 1999, Sinha & Richardson 1996). As a computational model, CBR has also been used as a tool for ME when it is applied to various other relevant domains such

as engineering design (Maher & de Silva Garza 1997, Kohno et al. 1997), law (Ashley 1991, Ashley & Rissland 1988), and software control (Mukhopadhyay et al. 1992).

Past cases can be used to provide a pattern for a solution to a similar problem, to remind one of past failures, to set expectations about missing or tacit information, and to help explain or justify one's decisions (Morris 1994). For complex design problems, it is easier to adapt an existing successful design than to create a new design "from scratch." The past design has been tested and its strengths and weaknesses identified. The difficult task of combining and integrating multiple components has been solved. The user and designer can spend their time more productively adapting a useable design to generate information models that will fit the current needs better.

Cases, patterns, and examples are all tools that can be used to improve communication in Method Engineering. Sometimes individuals who have a shared experience will name the experience and then use that name as a shorthand way to communicate about the events that occurred. For example, when the speaker mentions an event such as "the New Year's Eve party at Stan's", the listener will be reminded of the who was there, what foods were served, and the fact that the host became ill and went to bed at 10 p.m.

To further illustrate the effectiveness of using past cases or patterns for communication, consider the situation where a family decides to build a new home. The family has the following three alternatives ways of communicating their requirements to the architect/builder:

1. To enumerate all the features they want,
2. To describe their current home and explain how they want the new home design to differ from the existing one, or
3. To search pattern books to find a floor plan that is similar to what they want to build.

We will discuss the aforementioned three alternatives in more details in below:

Alternative 1:

Giving the architect/builder a list of requirements (i.e. information models) shifts the responsibility for the design from the owner to architect. This is a common approach for commercial buildings, but is not widely used for individual homes because it is a very expensive method. If the family wants something that is unique or designed by a famous architect, they may be willing to pay a premium. When the Kaufman family engaged Frank Lloyd Wright to build *Fallingwater*, they were buying a Frank Lloyd Wright design. Under this approach, the homebuyers give the architect a general description of the features they want. Through subsequent interviews and an iterative process of design reviews (i.e. the process of information modeling), the architect learns how the family intends to use the home and refines the initial requirements list. This method requires a significant amount of buyer-designer interaction to refine the user requirements via an iterative process – i.e. the method engineering (ME) process. In system development, as well as home building, this is a costly approach.

Alternative 2:

As an alternative, prospective homeowners could begin by describing their current home as information models and explaining how the home they want to build differs. This is the approach that is often taken in software requirements analysis. Users are asked to describe the current system, identifying its good features and its weaknesses or deficiencies.

In software development as well as home building, the desired changes from the existing design are likely to be significant - otherwise, why build a new one? The existing home (or system) reflects budgetary and time constraints that were present when it was built or purchased. Those are

likely to have changed over time. Also, the homebuyers' needs are likely to have changed as well. They may have more children, (or fewer children), an elderly parent, or some change in physical condition that requires a special design. As the extent of differences in the constraints and requirements increases, the usefulness of the existing plan (information model) as a communication mechanism decreases.

In fact, starting with the existing software may be somewhat dysfunctional due to the anchoring and adjustment bias. By anchoring on the existing system, it will be more difficult to conceive of and make major changes, thus hindering reengineering efforts.

Alternative 3:

The third alternative - searching plan books - is an intuitively appealing approach, and it the most common approach used. Plan books containing floor plans and front elevations (information models) are arranged by type of dwelling (e.g., ranch, multi-level) and by size and style (e.g., southwest, colonial) to make the search easier. The homebuyers can browse through the plans to get new ideas. They can compare and contrast designs and weigh the pros and cons of design alternatives (method engineering). When they find a plan that has most of the features they want (better information models), they take it to a designer to make modifications (method engineering). The plan selected by the homebuyers is rich details that would be difficult to communicate otherwise.

More importantly, detailed architectural renderings including mechanical drawings for heating, wiring and plumbing can be purchased for the plan selected. This means that the designer can begin with a tested design that combines and integrates multiple components. The homebuyer and designer can spend their time more productively adapting a tested design than trying to generate one from scratch.

Architectural, engineering and systems design are tasks that lack strong domain models and require effective communication between the buyers/users and the designers. CBR has been used to improve communication and support the design process for architectural and engineering systems and is likely to be useful for the systems development task as well.

Design Patterns

The concept of design patterns (DPs) in software engineering is often said to originate from the architectural works of Alexander et al. (1977) and Alexander (1979), in which the authors suggest that many beautiful commercial structures are not simply the result of creative architects. Rather, the designs evolved as the architects in the region copied and modified the successful designs of others to address local design problems such as climate and terrain. Alexander et al. (1979) began to recognize recurring patterns in architecture around the 1960s. With those recognized patterns, they were able to provide solutions to different architectural design problems in the form of fixed patterns. Even novice architects who desire to design buildings by themselves could use these patterns.

Alexander (1979) and Alexander et al. (1977) describe patterns and features that are derived from well-regarded architectural designs in the form of design problems and their solutions. Each solution provides relationships needed to solve a designated problem in both general and abstract fashions. By adapting a particular design that had been used to solve previous problems, lay architects were able to sketch novel designs that fulfill their preferences and meet local conditions in the place where the new structure is located. With 253 patterns included in Alexander (1979), a novice architect could create nearly infinite combinations of architectural designs. Furthermore, Alexander argues that patterns are well-researched solutions to recurring design problems. He also asserts that designs that violate the derived patterns were noticeably less successful than those that followed them.

Over the last decade, as object-oriented design has gained acceptance as a good technology for software development, people from the software community discovered Alexander's concept of pattern language. Although software design is seemingly a different domain from architectural design, researchers in software engineering (e.g. Eden & Hirsheld 1999, Eden & Yehudai 1999, Gamma et al. 1995, Keller et al. 1999) have found it helpful to have an abstract method, similar to that of Alexander's pattern language, to express the core solutions and the rationale behind software system designs.

Gamma et al. (1995, 3) define design patterns as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context." They name and identify key aspects of a common design structure such as "factory method pattern" or "singleton pattern," and provide a good way for creating a reusable object-oriented design. In addition, they also assist a software engineer in the maintenance of software product, modification of existing programs, removal of programming bugs, and addition of new features.

DPs are usually not invented; instead experienced software engineers and programmers derive them empirically from observations. Therefore, the use of DP ensures the successful application by other programmers in different situations. DPs are core solutions to recurring problems in software implementation. They capture both static and dynamic structures and collaborations of components in successful solutions to problems, and give software engineers an easier way to communicate designs on a higher level of abstraction. Otherwise, for each new design, the designers have to think in terms of individual classes and their behaviors.

According to Agerbo (1998), when compared with traditional software designs, the application of DPs in software development has the following advantages:

1. They encapsulate experience.

2. They provide a common way for software developers to communicate with each other

3. They enhance the documentation of software designs.

Gamma et al. (1995) describe a DP as including the following elements: *pattern name, intent, motivation, applicability, structure, participant, collaboration, consequence, implementation, sample code, known uses, and related patterns*. These elements provide information such as where to apply the pattern, how to use it, and what the results will be, etc. Table 1 below explains each element in more detail:

Elements	Function
Intent or pu	A short statement that explains: what does the design pattern do? What is its rational and intent? What particular design issue or problem does it address?
Motivation	A scenario that illustrates a design problem and how the structure of class and object in the pattern solve the problem.
Applicabilit	A list of situations that includes when the design pattern can be applied, examples of poor designs that the pattern can address, and how the user can recognize these situations.
Structure	A graphical presentation of the classes in the pattern by a notation on OMT (Object Modeling Technique).
Participants	A list of classes and/or objects that participate in the design pattern and their responsibilities.
Collaboratio	A description that explains how the participants collaborate in order to carry out their responsibilities.
Consequenc	An explanation of how the pattern supports its objectives, what are the trade-offs and results of using the pattern, and what aspects of the system structure that may change independently.
Implementa	A list of descriptions about pitfalls, hints, or techniques that could cause problems when implementing the pattern, as well as other language-specific issues.
Sample Cod	An illustration of how to implement the pattern in C++, Java, or Sma
Known Uses	An introduction of where to successfully use the pattern in specific s
Related Patt	A list of related design patterns and their differences.

Table 1: **Elements in a Design Pattern (Gamma, et al. 1995)**

Design patterns represent a form of case-based reasoning in IM that supports the communication between designers and programmers.

Analysis Patterns

Analysis patterns (APs) differ from DPs. APs represent the conceptual structures of *business processes* rather than actual *software implementations* (Fowler 1997, p. xv). Analysis patterns are intended to reflect the way people think about business activities rather than the way a computer system is designed. As with DPs, APs are derived from experience, not from an academic exercise.

Fowler provides a catalog of APs divided into the following categories: *trading, measurement, accounting,* and *organizational relationships*. APs are groups of concepts that represent a common construction in business modeling (Fowler 1997, p. 8). Examples of accounting APs are: *account* and *entry, transactions, summary account, posting rules, booking entries to multiple accounts*. Each pattern is discussed in terms of one or more actual contexts or business scenarios in which it was originally used.

For many patterns, Fowler includes alternative designs together with a discussion of the strengths and weaknesses of each. For example, under the topic of transactions, he discusses two-legged vs. multi-legged transactions. Two-legged transactions are ones in which there are exactly two accounts involved (equal amounts are always debited and credited). Multi-legged transactions can involve more than two accounts with the constraint that the sum of the debits equals the sum of the credits3. As Fowler notes, although there may be business situations where all transactions are only two-legged, such as tracking the movement of goods from the receiving dock to the warehouse, using a multi-legged construction is more general, and provides greater flexibility to

3 Fowler avoids the words “debit” and “credit.” His concepts are intended to be more abstract and not related specifically to actual accounting and inventory contexts. For example, such transactions could also be used to track the movement of evidence in legal proceedings. Furthermore, his audience is software designers, not accountants.

accommodate future changes. Inclusion of alternatives and an assessment of the pros and cons of each within the context of a specific problem enhances the usefulness of the pattern approach.

Analysis Patterns are not mere extensions of the DP. DP represent small components of code. AP represent small chunks of business processes. AP sometimes make use of established DP. For example, Fowler's (1997) accounting patterns utilize Gamma, et al., (1995) *iterator*, *singleton*, and *strategy* patterns.

Fowler's patterns are based on an object-oriented model. Hay (1995) also discusses similar conceptual business process patterns using a relational data modeling style.

Analysis patterns are a form of case based reasoning that supports the conceptual design process and communication among design team members.

THE PROPOSED MODEL – Illustrations of How the Proposed model works

Case based reasoning approaches are available for the conversion of requirements into a conceptual design (APs) and for conversion of the design into code (DPs), but not for determining system requirements. This section examines the feasibility and usefulness of such a system. It should be noted that the proposed system need not be a computational model. Fowler (1997) presented his catalog of patterns as a hard copy book.

This section is divided into two parts. Section 5.1. describes a design problem. Section 5.2. describes how a case could be structured to support the analysis and design process and illustrates its use with the design problem from section 5.1.

Design Problem

Consider the design problem described in Table 2. The objective is to design a system to support work of energy brokers.

Case scenario:

As a result of energy deregulation in the USA, companies are forced to shop for electricity and natural gas. Shopping for electricity can be quite difficult. Different suppliers have different pricing schemes. Prices may vary with the peak demand, as measured over fifteen minute intervals—a single spike in a single fifteen minute period may increase the rate multiplier under many pricing plans. The best prices are for those customers who have smooth demand over long blocks of time. The contracts offered by the power providers vary in length. The process is further complicated by "shopping credits" and "price-to-compare" numbers provided by power providers under mandates from their Public Utility Commissions to facilitate the transition from a regulated to a deregulated environment.

Because of the complexity of the calculations involved in comparing utility contracts, many organizations turn to energy brokers or consultants. Some work for a flat fee. Some charge by the number of kilowatt hours used by the client. Others are aggregators who accept customers who can contribute to a smooth demand and thus qualify for lower kilowatt hour charges and they split the savings with the customer. This client currently charges by the kilowatt hour used.

The job of the broker/agent is to match the customer with the electric service provider that provides the "best" deal. To do this, the broker/agent obtains a billing

history of electricity usage from the customer or from the customer's current provider. Billing histories obtained from the utilities are preferable because they can be obtained in an electronic format and therefore do not need to be re-keyed. Unfortunately, the data format varies from one utility to another. The broker/agents calculate the cost of providing electricity to the customer under the various rate options available from the electricity service providers, taking into account shopping credits, price-to-compare and available contract length.
System Objective: To develop a system to support the work of the energy brokers

Table 2 - Energy Broker Case Narrative

This is a good case to illustrate the usefulness of a CBR approach. Energy Broker is a new business model for the electric industry. There was no need for broker/agents in a regulated world. If employees and management of the broker/agent firms were previously employees of a regulated utility, they may not be familiar with alternative broker/agent models. Because this is a new business model, they also may not have a good understanding of how the business is likely to evolve. If the system analysts/designers also lack prior experience with broker/agent systems, the SRA task will be even more difficult.

Case Structure and Examples

A case based approach requires a set of past cases, an indexing scheme for storage and retrieval, and a mechanism for determining similarity of cases. In domains where there are a large number of cases, case indices can be derived computationally using rule induction or other data mining or classification techniques. In problem domains such as systems analysis, the indexing scheme will evolve as cases are added. When a new case is added, a subcategory can be created if

the new case solution is sufficiently different from the previous cases. Similarly, case categories can be collapsed if the solutions in two categories are very similar.

Indexing should be broadly defined and should be related to the attributes that have the greatest impact on the system design. Manufacturers, wholesalers, retailers, and service organizations conduct business in very different ways. Consequently their information systems needs are quite different. Therefore, organization type is an important index. "Manufacturer" is probably too broad a category. More specific indexes based on the nature of the product (commodity or differentiated), the production processes (continuous or batch), or other factors that affect the information system design may be needed. Similarly, the "service" category can be further indexed into organizations such as accounting firms, engineering firms and attorneys who charge by the hour and broker/agents, such as real estate agents and investment managers who charge based on completed transactions. Refinement of the indexes can take place as additional cases are added to the database.

In most CBR systems, indexing and similarity metrics are devised to retrieve a single case that is most similar to the current case. For the SRA task, it is not as important to select a single case as an analog. In fact, it is more desirable to retrieve several past cases in order to generate many alternative designs. In medical diagnosis, research has found that "early closure," i.e., the failure to consider sufficient alternative diagnoses, leads to poor outcomes. In a similar way, failure to consider alternative designs may adversely affect the development process.

For the purposes of illustration, assume that we have a CBR system with two broker/agent cases. The first is an Investment Management system and the second is a Web-based Freight broker.

First, the Investment Management system will be used to illustrate what comprises a case and how the CBR system could be used to aid in system development. The case is from Carmichael (1998). It is based on an actual experience rather than a textbook model. One part of the case record includes a narrative description as shown in Table 3 (although, in greater detail).

Case description:

The client is an Investment Management firm with several regional offices. The regional offices have Investment Managers who manage client portfolios. Actual contact with the client is done by separate Relationship Managers. The Investment Managers learn of investment opportunities from their internal Investment Department. For some clients, the Investment Manager is authorized to make trades directly. For other clients, the Investment Manager makes recommendations and the firm trades for the account only when authorization is received from the client. A third type of client makes his/her own investment decisions and the Investment Manager simply executes the trades they request. When making trades for managed accounts or referral accounts, the Investment Manager must consider the client's current holdings (cash account and securities) as well as transaction in progress (authorized, but not yet executed and settled).

The firm has a Dealing Division that carries out all interactions with the market based on instructions received from the Investment Managers. They make bulk trades for the total shares authorized for the various clients.

When the trade is completed, funds are transferred from the client bank account and securities are transferred to the portfolio account.

System Objective:

To improve the services that the Investment Managers give.

Table 3. Investment Management Case Narrative (Carmichael, 1998)

The next component of the case record is an overview of the object model. See Figure 1 for Carmichael's overview of the class model which shows the following classes: *ClientGroup, Client, Opportunity, BankAccount, Holding, InitialHolding, Deal, BulkDeal,* and *Security*. More detailed documentation of the model would be included. For example, the *Client* package can be modeled

with three subclasses for *ManagedClient, ReferClient*, or *IndependentClient* or as a *General client* class with subclasses for *ManagedClient, ReferClient.* Alternative models should also be stored in the case record together with the author's opinion of the pros and cons of each.

BankAccount and *Holding* use Fowler's *account-entry* and *transaction* analysis patterns. Finally, the case will hold a design model (see Figure 2) that, in this case, is for a Java implementation. Other AP and DP are included in the detail design. The design model provides the final link from the business model to the implementation model and from the user to the programmer.

To summarize, a case record includes a narrative description, an overview of the class model, details of the class model, model alternatives and the justification for the selected alternative, and a design model. The narrative and overview of the class model are used to facilitate the communication between the user and the analyst during SRA. The other case components aid the analyst in developing the conceptual design and the programmers in implementing the design.

The next section illustrates how such a case might be used to support communication between the user and the analyst.

Even with a limited case description and a single sketchy case in our case database we can use CBR to support communication between the user and the analyst. Both the design problem and the investment manager cases would be indexed (at least minimally) as broker/agents. The user can understand the narrative and most likely the analysis class model (Figure 1), as well. By retrieving the narrative and analysis class overview from the Investment Management case, the analyst and user can begin a dialogue that will quickly lead to design details. For example, the following questions can be generated from the class overview:

- Is the broker going to do *BulkDeals* (i.e., act as an aggregator)?

- Is there an equivalent of *BankAccount*? (Will the broker bill the customer for the electricity used or only for the broker's fee?)
- How does the contract acquired by the customer differ from the investment manager's *Security*?
- Will customers have more than one contract (i.e., the equivalent of a *portfolio*)?
- What details about *Holdings* (contracts) does the system need to keep track of in order to help the brokers manage *Opportunities*?

Many of the questions could have been generated and answered during a traditional user interview. The difference is that with CBR the questions generated from the overview are linked to a conceptual design and to an analysis of the pros and cons of various design alternatives.

Other questions related to the detail of the class model would be generated as well, such as the following questions about the client package:

- Will there be three classes of customers (*Refer, Independent* and *Managed*)?
- If so, are there case features that would suggest a preference for either of the alternative models (one general class with two subclasses or three subclasses)?

The analysts are not only interested in the way business is conducted today, but also how the business might change in the future. If the energy broker does not have *Managed* accounts now, how likely is it that they will add them in the future? Is the likelihood sufficiently high to justify the additional cost of building the capability for *Managed* accounts into the system now?

As the questions are answered, the differences flow down through the system and change the final design model, which in turn changes the implementation. The recalled case suggests new ways of doing similar tasks, both at the strategic level and at the operations level. It suggests potential problems with alternative modeling approaches. It helps explain modeling choices.

To further illustrate the benefit of considering multiple past cases, a second case, the Freight Broker, is discussed. The narrative for this case is shown in Table 4.

Case description:

When a company wants to ship goods to their customers or to other plant locations, they contact freight carriers (trucking companies) to schedule the shipment. This is a labor-intensive task. Dispatchers may have to telephone several carriers before they can match their outgoing shipment with a truck that can make the delivery within the customer's schedule requirements.

Some Internet based freight broker exchanges focus on the "spot" market. Their systems function as an auction where a Shipper posts a shipment and Freight Carriers bid on it. The lowest cost Carrier who can meet the schedule constraints is awarded the shipment.

The spot market accounts for less than 20% of the total market. Most Shippers have a set of trusted Freight Carriers who have proven to be reliable. To schedule a shipment, their dispatchers begin by contacting these Freight Carriers first. They use the spot market only when none of their preferred carriers are available.

The system requirements are: a system that allows Shippers to post shipments on the Web, allows the Shipper to designate preferred Carriers, notifies Carriers when a Shipper has designated them as a preferred Carrier, allows the preferred Carriers to bid or decline the shipment or to make a counter offer (different time), allows the Shipper's dispatcher the opportunity to confirm or reject the offer or counter offer, rolls the shipment over to the next Carrier on the list of preferred Carriers if the first choice declines the shipment, and finally rolls the shipment over to the spot market if none of the preferred Carriers is able to make

the schedule or propose an acceptable alternative.
System Objective: To develop an online system that supports collaboration and communication between Shippersand Freight Carriers

Table 4 - Freight Broker System Narrative

The Freight Broker case raises a number of issues that were not considered in the Investment Manager case. First, the Investment Manager adds customer value by making recommendations to customers and by posting customer transactions on the stock exchange for the customer. The Freight Broker system is an exchange. It adds value by facilitating communication and matching shipments with carriers.

In some ways the Energy Broker system is like the Investment Management system—both make recommendations to customers. In other ways, the Energy Broker is more like the Freight Broker. How likely is it that the Energy Broker system will evolve into a web-based B2B exchange like the Freight Broker? This is an important question because it affects the way the current system should be designed.

There are other questions that the Freight Broker system raises:

- Does the Energy Broker need to have something similar to the *Preferred Carrier?*
- Will the Energy Broker act as an agent who is able to bind the customer to the contract or does the system require the customer to confirm and approve a contract?

The Freight Broker system was intended to save Shippers money by reducing the labor costs associated with scheduling shipments. The initial design did not accomplish that because

dispatchers were not eliminated. Their task changed from interacting with Carriers by telephone to re-entering shipment data from their company's production/order fulfillment system into the Freight Broker exchange. To address the problem, they developed custom interfaces for their large customers that allows them to dump shipment schedules directly into the exchange. This is an expensive solution because each customer has its own data format.

The Energy Brokers also have a problem with receiving system input in several different formats. Even though the Freight Brokers solution is inadequate, it is important to raise the issue and have the designers of the Energy Broker system consider how to address. In fact, additional indexing based on design problems can be added to the CBR system.

CONCLUSION AND SUMMARY

This paper discussed the usefulness of developing a CBR model to support user-analyst communication in the development of accounting based software. CBR has been used in several problem domains, including other engineering design tasks. Design patterns and analysis patterns are methods of case-based reasoning that are currently being used to support programmers, analyst, and designers. Developing a CBR system to support users during the requirements analysis phase seems like a logical extension. CBR contributes to the development process in two important ways. First, the indexing scheme is used to help the analyst generate questions about the system requirements. Second, the retrieved cases can be used to generate alternative designs and to weigh the pros and cons of various design choices. The examples in this paper illustrate the potential usefulness of such an approach.

Advantages of the case-based approach are:

- it will better support business process reengineering efforts

- it will make users more proactive in the development process
- it will encapsulate experience
- it will aid communication between user and designer
- it may improve system documentation

For CBR to be useful as a tool to aid analysts and users, cases like the Investment Management case presented by Carmichael (1998) must be compiled, indexed and used. Future research in this area should focus on identifying and indexing additional cases and exploring the development of computational tools to integrate CBR, design patterns, and analysis patterns.

DP and AP are represented graphically using UML or other documentation methods. Graphical depictions of the information in the case narratives should be developed and tested to determine whether the graphical or textual representation leads to better understanding by the users.

REFERENCES

Agerbo, E. & Cornils, A. (1998): "How to Preserve the Benefits of Design Patterns," *Proceedings of the 1998 Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) Conference*:134-143.

Alexander, C. (1979): *The Timeless Way of Building*, New York: The Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. & Angel, S. (1977): *A Pattern Language*. New York: The Oxford University Press.

Ashley, K. (1991): "Reasoning with Cases and Hypothetical in HYPO," *International Journal of Man-Machine Studies*, 34:753-796.

Ashley, K. & Rissland, E. (1987): "Compare and Contrast: A Test of Expertise," *Proceedings of AAAI-87*:273-278.

Ashley, K. & Rissland, E. (1988): "Case-Based Approach to Modeling Legal Expertise," *IEEE Expert*, 3(3): 70-77.

Booch, G., Jacobson, I. & Rumbaugh, J. (1998) *The Unified Modeling Language User Guide*, Addison-Wesley.

Booch, G., Jacobson, I. & Rumbaugh, J. (1999) *The Unified Software Development Process*, Addison-Wesley.

Carmichael, A.(1998): "Applying Analysis Designs in a Component Architecture," http://www.togethersoft.co.uk

Carmichael, A. & Swainston-Rainford, M. (2000): "The Features Game", http://www.togethersoft.co.uk

Cushing, B.E. & Loebbecke, J.K. (1986): *Comparison of Audit Methodologies of Large Accounting Firms.* American Accounting Association: Studies in Accounting Research No. 26.

Cushing, B.E. & Romney, M. (1997): *Accounting Information Systems*. New York: Wiley & Sons. Co.

Danos, P.D., Eichenseher, J.W. & Holt, D.L. (1989): "Specialized Knowledge and Its Communication in Auditing," *Contemporary Accounting Research*, 6(1):91-109.

Denna, E.L., Hansen, J.V., Merservy, R.D. & Wood, L.E. (1992): "Case-Based Reasoning and Risk Assessment in Audit Judgment," *International Journal of Intelligent Systems in Accounting, Finance & Management*, 1(3):163-171.

Dykman, C.A. & Robbins, R. (1991): "Organizational success through effective systems analysis," *Journal of Systems Management* (3):6-8.

Eden, A.H. & Hirsheld, Y. (1999): "On the Understanding of Software Patterns," http://www.math.tau.ac.il/~eden

Eden, A.H., Gil, J. & Yehudai, A. (1999): "Automating the Application of Design Patterns," http://www.math.tau.ac.il/~eden

Fowler, M. (1997): *Analysis Patterns: Reusable Object Models*. Menlo Park CA: Addison Wesley Longman, Inc.

Friedman, S. (1995): "Case Study: Pay Attention to Warning Signals," *Journal of Accountancy*, 180(4):88-90.

Gamma, E., Richard, H., Johnson, R. & Vlissides, J. (1995): *Design Patterns: Elements of Reusable Object-Oriented Software.* New York: Addison-Wesley.

Gibbins, M. & Mason, A (1988): *Professional Judgment in Financial Reporting.* Toronto, Canada: Canadian Institute of Chartered Accountants.

Gjersvik, R, Krogstie, J. & Folstad, A. (2005): "Paticipatory Development of Enterprise Process Models," in Krogstie, J., Halpin. T.A. and Siau, K. (eds.) *Information Modeling Methods and Methodologies*, Hershey, PA: Idea Group Publishing, 195-215.

Hay, D.C. (1995): *Data Model Patterns: Conventional Thoughts.* Dorset House Publisher.

Jung, C., Han, I. & Shu, B. (1999): "Risk Analysis for Electronic Commerce Using Case-Based Reasoning," *International Journal of Intelligent Systems in Accounting, Finance & Management*, 8(1):61-73.

Kavakli, E & Loucopoulos, P. (2005): "Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods," in Krogstie, J., Halpin. T.A. and Siau, K. (eds.) *Information Modeling Methods and Methodologies*, Hershey, PA: Idea Group Publishing, 102-124.

Kawalek, P. & Wastell, D. (2003): "A Case Study of the Use of the Viable System Model in the Organization of Software Development," in Siau, K. (eds.) *Advanced Topics in Database Research*, Vol. 1, Hershey, PA: Idea Group Publishing, 120-134.

Keller, R.K., Schauer, R., Robitaille, S. & Page, P. (1999): "Pattern-Based Reverse Engineering of Design Components," *Proceedings of 1999 International Conference of Software Engineering*:226-235.

Kohno, T., Hamada, S., Araki, D., Kojima, S. & Tanaka, T. (1997): "Error Repair and Knowledge Acquisition via Case-Based Reasoning," *Artificial Intelligence*, 91:85-101.

Kolondner, J.L. (1993): *Case-based Reasoning*. San Francisco, CA: Morgan Kaufmann.

Ku, S., Suh, Y. & Tecuci, G. (1996): "Building an Intelligent Business Process Reengineering System: A Case-Based Approach," *International Journal of Intelligent Systems in Accounting, Finance & Management*, 5(1):25-39.

Lee, S. & Han, I. (1998): "The Design of EDI Controls Using Case-Based Reasoning: EDICBR," *International Journal of Intelligent Systems in Accounting, Finance & Management*, 7(3):135-152.

Mahar, M.L. & De Silva Garza, A.Z. (1997): "Case-Based Reasoning in Design," *IEEE Expert Intelligent Systems and Their Applications*, 12(2):34-41.

Matulevicius, R. (2005): "Validating an Evaluation Framework for Requirements Engineering Tools," ," in Krogstie, J., Halpin. T.A. and Siau, K. (eds.) *Information Modeling Methods and Methodologies*, Hershey, PA: Idea Group Publishing, 148-174.

Morris, B.W. (1994): "SCAN: A Case-Based Reasoning Model for Generating Information System Control Recommendations," *International Journal of Intelligent Systems in Accounting, Finance & Management*, 3(1):47-63.

Mukhopadhyay, T., Vicinanza, S.S. & Prietula, M.J. (1992): "Examining the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation," *MIS Quarterly*, 16(2): 155-171.

Munch, J. & Heidrich, J. (2004): "Software Project Control Centers: Concepts and Approaches," *Journal of Systems and Software*, 70(1-2):3-19.

Riesbeck, C. & Shank, R. (1989): *Inside Case-Based Reasoning*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Rockwell, S.R. & McCarthy, W.E. (1999): "REACH: Automated Database Design Integrating First-Order Theories, Reconstructive Expertise, and Implementation Heuristics for Accounting Information Systems," *International Journal of Intelligent Systems in Accounting, Finance & Management*, 8(3):181-197.

Salterio, S. (1994): "Researching for Accounting Precedents: Learning, Efficiency, and Effectiveness," *Contemporary Accounting Research*, 11(1):515-542.

Salterio, S. (1996): "The Effects of Precedents and Client Position on Auditors' Financial Accounting Policy Judgment," *Accounting, Organizations and Society*, 21(5):467-486.

Siau, K. & Rossi, M. (1998): "Evaluation of Information Modeling Methods – A Review," in *Proceedings of the Thirty-First Annual Hawaii International Conferences on Systems Sciences (HICSS)*, Vol. 5, 314.

Siau, K. & Rossi, M. (2001): "Information Modeling in the Internet Age – Challenges, Issues, and Research Directions," in Rossi, M. & Siau, K. (eds.), *Information Modeling in the New Millennium*, 1-8.

Siau, K. (2003): "The Psychology of Information Modeling," in Siau, K. (eds.) *Advanced Topics in Database Research*, Vol. 1, Hershey, PA: Idea Group Publishing, 106-118.

Siau, K. & Cao, Q. (2003): "How Complex Is the Unified Modeling Language?" in in Siau, K. (eds.) *Advanced Topics in Database Research*, Vol. 1, Hershey, PA: Idea Group Publishing, 294-306.

Sinha, A.P. and Richardson, M.A. (1996): "A Case-Based Reasoning System for Indirect Bank Lending," *International Journal of Intelligent Systems in Accounting, Finance & Management*, 5(4):229-240.

Standish Group (1995): *CHAOS Chronicles* Company Research Report, http://www.standishgroup.com.

Standish Group (2003): *CHAOS Chronicles* Company Research Report, http://www.standishgroup.com.

Whittle, J. (2003): "Formal Approaches to Systems Analysis Using UML: An Overview," in in Siau, K. (eds.) *Advanced Topics in Database Research*, Vol. 1, Hershey, PA: Idea Group Publishing, 324-341.