IDEA GROUP PUBLISHING

ITJ2304

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

Understanding the Role of Use Cases in UML: A Review and Research Agenda[1]

BRIAN DOBING, University of Lethbridge, Canada
JEFFREY PARSONS, Memorial University of Newfoundland, Canada

A use case is a description of a sequence of actions constituting a complete task or transaction in an application. Use cases were first proposed by Jacobson (1987) and have since been incorporated as one of the key modeling constructs in UML (Booch, Jacobson, & Rumbaugh, 1999) and the Unified Software Development Process (Jacobson, Booch, & Rumbaugh, 1999). This paper traces the development of use cases, and identifies a number of problems with both their application and theoretical underpinnings. From an application perspective, the use case concept is marked by a high degree of variety in the level of abstraction versus implementation detail advocated by various authors. In addition, use cases are promoted as a primary mechanism for identifying objects in an application, even though they focus on processes rather than objects. Moreover, there is an apparent inconsistency between the so-called naturalness of object models and the commonly held view that use cases should be the primary means of communicating and verifying requirements with users. From a theoretical standpoint, the introduction of implementation issues in use cases can be seen as prematurely anchoring the analysis to particular implementation decisions. In addition, the fragmentation of objects across use cases creates conceptual difficulties in developing a comprehensive class model from a set of use cases. Moreover, the role of categorization in human thinking suggests that class models may serve directly as a good mechanism for communicating and verifying application requirements with users. We conclude by outlining a framework for further empirical research to resolve issues raised in our analysis.

The Unified Modeling Language, or UML (Booch, Jacobson, & Rumbaugh, 1999), has rapidly emerged as a standard language and notation for object-oriented modeling in systems development, while the accompanying Unified Software Development Process (Jacobson, Booch, & Rumbaugh, 1999) has recently been developed to provide methodological support for the application of UML in software development. The adoption of UML brings focus to object-oriented developers faced with the task of choosing among dozens of proposed approaches to object-oriented analysis and design. In light of this activity, driven primarily by practitioners, it is important from an academic perspective to independently evaluate the capabilities and limitations of UML and the Unified Process. Such evaluations can contribute to the development of theoretical underpinnings of UML, to an improvement in its modeling power and usability, and to its appropriate application in systems development projects.

This paper focuses on two components of UML: use cases and class models. In particular, we consider the appropriateness of use cases as a component of an object-oriented modeling language by looking at their role as a tool for communicating with users, and the relationship between use cases and the class models that are developed from them. We examine the variability in the amount of detail use cases should contain, according to various proponents, and introduce a theoretical rationale for including fewer task details than many proponents advocate. We discuss the lack of 'object'-orientation in use cases, and present a theoretical argument that use cases may, in fact, not be necessary or valuable in UML. Finally, we develop a framework for empirical research to evaluate the value of use cases and their relationship to class models in UML.

USE CASE FUNDAMENTALS

The term "use case" was introduced by Jacobson (1987) to refer to "a complete course of events in the system, seen from a user's perspective" (Jacobson, Christerson, Jonsson, & Overgaard, 1992, p. 157). The concept resembles others being introduced around the same time. Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen (1991); Wirfs-Brock, Wilkerson, and Wiener (1990); and Rubin and Goldberg (1992) used scenarios or scripts in a similar way. But, despite

Manuscript originally submitted March 1, 2000; Revised May 15, 2000; Accepted June 14, 2000 for publication.

concerns about the awkwardness of the name, the use case has become an important part of most object-oriented analysis and design methodologies. Use cases were incorporated into UML in late 1995, after Ivar Jacobson joined forces with Grady Booch and James Rumbaugh.

The use case differs from typical structured requirements analysis tools that preceded it in two important ways. First, the use case is largely text-based. Structured analysis emphasized the importance of graphical tools, such as Work Flow and Data Flow Diagrams. The rationale for preferring diagrams to text was the oft-cited 'a picture is worth a thousand words.' In addition, before structured methodologies became available, analysts often generated extensive and unstructured text descriptions of existing and proposed systems that were very difficult to use. UML has not abandoned diagrams; Activity, Sequence and Use Case Diagrams all play important roles during analysis. But use cases are the key communication tool, so that "users and customers no longer have to learn complex notation" (Jacobson et al., 1999, p. 38).

Second, use cases focus on transactions from the user's perspective. In Data Flow Diagrams, transaction sequences were often not explicitly articulated. All the steps needed to, for example, sell goods to a customer would be there, but the connections between taking orders, checking inventory levels, determining payment types and authorizations, printing receipts, and other activities were not always clear. The focus on complete transactions shares some important similarities with the concept of a "process" in Business Process Reengineering, "a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer" (Hammer & Champy, 1993, p. 35). Both emphasize complete transactions viewed from a customer or user perspective, although the terms "user" and "customer" imply a different level of analysis. Jacobson, Ericsson, and Jacobson (1994) deal extensively with using use cases to support reengineering, suggesting the similarity is not coincidental.

Use cases have been all but universally embraced in object-oriented systems analysis and development books written since Jacobson et al. (1992). There are a few exceptions, but their alternatives still share some common features. For example, Coad (1995) refers to "scenarios" that seem more detailed or lower level than use cases (e.g., a sale calculating its total (p. 61)). Nevertheless, Norman (1996, p. 165) suggests that Jacobson's use cases and Coad's scenarios are "similar concepts." Kilov and Ross (1994, pp. 9-10) use the notion of a "contract" that states "what has to be true before and what will be true after the operation." Contracts focus more on pre- and post-conditions rather than the steps in between, but again there are similarities.

USE CASE INTERNAL STRUCTURE

Analysis Versus Design Focus

Despite the strong endorsement of the general use case concept, there are many variations on Jacobson's original theme. Not all use cases are created equal. First, there is a difference in content. Use cases, at least during the analysis phase, are generally viewed as a conceptual tool. The use case should emphasize 'what' and not 'how' (Jacobson et al., 1994, p. 146). This suggests use cases shouldn't mention technology (e.g., Evans, 1999).

A review of use case examples shows that determining when the 'what' ends and the 'how' begins is not always easy. Brown (1997) interprets 'what' to mean what the system will do rather than the internal implementation. Thus, his use cases include references to screen designs. So do those of Satzinger and Orvik (1996, p. 126). Harmon and Watson (1998, p. 121) go further in their example and refer to the salesperson's laptop. And even Jacobson et al. (1992, p. 162) refer to a display "panel," "receipt button" and "printer" in one of their examples. Some use cases also include more detail on business rules. For example, the IBM Object-Oriented Technology Center (1997, p. 489) video store example includes the condition that customers who are not members pay a deposit of $60.

However, as Larman (1998, p. 10) notes, use cases are not tied to object-oriented methodologies and thus are technology-independent in that sense. The same cannot be said for Data Flow Diagrams, which were designed to produce a basic module structure for a COBOL program. Object-oriented systems can be built without use cases and, conversely, use cases could be used in non-OO projects.

A second issue in use case structure is the variety of formats that have been proposed. Some, such as whether use case titles should begin with gerunds (e.g., 'Adding a Customer') or action verbs (e.g., 'Add a Customer'), are not serious. More interesting is the format of the text itself. While the first use cases in Jacobson et al. (1992) were written as a paragraph of text, most others have adopted numbered steps. More recently, Jacobson et al. (1994, p. 109) have done so as well. This may not appear to be a serious issue, but sequenced and numbered steps are an invitation to write about 'how.' While the underlying technology need not be mentioned, use cases have become very process oriented. In most cases, they go much further than simply documenting requirements to providing a suggested solution.

Third, the comprehensiveness of use cases also varies. Some take a minimalist approach. Jacobson et al. (1994, p. 105) suggest that use cases should offer "measurable value to an individual actor." MacMaster (1997) argues that use cases be used only for main system functions. But White (1994, p. 7) states that "the collected use cases specify the complete functionality of the system." While Dewitz (1996) uses 11 use cases in her video store example, the IBM Object-Oriented Technology Center (1997) has 24.

Fourth, the level of detail within each use case also varies. Constantine and Lockwood (2000) distinguish between "essential" use cases, containing few if any references to technology and user interface implementation, and "concrete" use cases that specify the actual interactions. Clearly, use cases could move from essential to concrete as the development process proceeds. But not everyone agrees that concrete use cases should ever be used (e.g., Evans, 1999). There are alternative mechanisms that can be used to document screen design choices and similar decisions.

Jacobson et al. (1999) advocate an iterative development approach in which both the number of uses cases and their level of detail increase as the life cycle progresses. They suggest that only the most critical use cases (less than 10%) be detailed in the first (inception) phase. As analysis progresses and requirements become firmer, additional use cases can be added and each can be expanded to include considerably more detail. The analyst could move toward concrete use cases or simply expand the detail within essential use cases. However, knowing where to start, how far to go at each phase, and when to stop, are clearly critical issues not easily resolved.

To further complicate the issue, some of those who favor fewer or less detailed use cases supplement them with "scenarios." Booch (1994, p. 158) defines scenarios as examples of what can happen within a use case. 'Add a customer' is a use case. Adding a specified customer with a particular name, address, etc. is a scenario. A well-chosen set of scenarios provides further detail on exception handling and other special cases (e.g., customers with missing, improbable, or unusual data (Lockheed Martin, 1996)). The same scenarios can later be used in testing. A minimalist approach to use cases combined with extensive scenarios may still result in a large and very detailed set of specifications.

Fifth, and perhaps most important, the role of use cases varies among methodologies. Earlier work on UML focused on the language itself, and was largely agnostic on issues of methodology. But the Unified Process (Jacobson et al., 1999, p. 34) makes clear what was always implicit – use cases "drive the whole development process." In particular, they provide "major input when finding and specifying the classes, subsystems and interfaces." Rosenberg and Scott (1999), however, suggest that "domain modeling" precede use case development. Their domain model is a "glossary of terms" (p. 16), intended to evolve into the objects, attributes, operations and associations. This glossary is based on "available relevant material" (p. 16). From this, a skeletal class diagram is constructed. They warn, "Don't try to write use cases until you know what the users will actually be doing" (p. 45). Thus, use cases will drive design, but not problem solving. Schneider and Winters (1998) begin with a written project description and risk analysis, before defining the system boundary and the actors. Then, use cases are identified. Blaha and Premerlani (1998, p. 49) state that, "Once you have a sound object model, you should specify use cases" and warn that early use cases must be regarded as "tentative and subject to revision" (p. 150).

In summary, review of the literature shows extensive differences in how use cases are defined and used. These differences certainly exceed the basically cosmetic variations in Data Flow Diagram and Entity Relationship Diagram formats found in standard structured analysis books. The existence of different use case formats and roles is not surprising, given UML's relatively short history. Moreover, UML brings together many analysis and design constructs because of its roots. While this is a notable achievement, the end product is loosely defined, complex (perhaps overly so), lacks a strong theoretical foundation, and thus is very difficult to test in a definitive way.

Determining Appropriate Use Case Focus

The use case variations are real. Despite a general consensus that use cases are intended for conceptual modeling of system requirements, many versions of use cases incorporate significant design and implementation details (e.g., at the level of the user interface). One potential way to resolve this apparent inconsistency is to adopt a contingency perspective. Different approaches may be useful under different circumstances, with the best approach in a specific situation depending on the analysts, the task, the users, and other situational variables.

However, we believe a stronger basis can be adopted to predict a most appropriate form for use cases that is applicable across a wide range of circumstances. The key to this proposal is implied by the general idea outlined earlier that use cases are requirements analysis and modeling tools that should describe what a system does (or should do), rather than how the system works (or should work).

Within this context, detailed use cases that specify low-level actor interactions with a system (e.g., down to the point of screen designs) essentially embed certain design choices. Introducing such considerations during analysis may prematurely guide the developers to specific implementation decisions. This is particularly a concern when the development process is intended to support the reengineering of existing processes, an endeavor for which Jacobson et al. (1994) strongly advocate the application of use case-driven methodology.

The potential impact on systems development of use cases that embed design decisions can be understood in the context of a well-known phenomenon in psychology – *anchoring and adjustment* (Tversky & Kahnemann, 1974). Experiments have shown that, when people are given a problem and an initial estimate of its solution, and then asked to find a final solution to a problem, they tend to anchor to the initial estimate (Plous, 1993). That is, they tend to provide solutions close to the initial estimate (anchor), even when those estimates are severely flawed. Anchoring is a useful heuristic that helps humans simplify problem solving in a complex situation. Unfortunately, people tend to rely on anchoring too much, resulting in an adjustment bias, in which people fail to make adequate modifications to an initial solution.

The concepts of anchoring and adjustment, although originally proposed in the context of activities such as subjective probability estimation, have a natural application to use cases. To the extent that use cases include design or implementation details that reflect current ways of doing things, reengineering or process innovation are likely to be inhibited. Consequently, we postulate that the level of innovation that can be achieved through use case-driven process design is inversely related to the level of design or implementation detail embodied in the use cases.

FROM USE CASES TO A CLASS MODEL

Finding Objects in Use Cases

In addition to modeling systems requirements from a user perspective, use cases and use case diagrams specify the behavior of the objects in a system. Some developers use them to identify the object classes required in the implementation, and the behavior of objects. In this way, use cases feed the development of subsequent models in UML: particularly the class model, but also sequence, activity and statechart diagrams and other UML artifacts.

In this context, it is useful to examine prescriptions in the UML literature for proceeding to the development of a class model from use cases. Booch et al. (1999) advocate applying "use case-based analysis to help find these abstractions" (p. 55), and describe this as an "excellent" way to identify classes. This view has subsequently been echoed in the Unified Process. According to Jacobson et al. (1999, p. 34), "use cases provide major input when finding and specifying classes." They further go on to assert "classes are harvested from the use case descriptions as the developers read them looking for classes that are suitable for realizing the use cases." However, they do not offer specific prescriptions for finding classes of objects in use cases.

Jacobson et al. (1994) provide a more detailed description of the role of use cases in finding classes of domain objects:

> When you have a first proposal for the most obvious entity objects, you continue to work with the use cases. You identify objects by traversing one use-case description at a time to ensure that there is an object responsible for each part of the use case's course of events. … When you work through the use case's course of events in this way, it is probable that you will identify further object entities. (pp. 184-185)

'Noun/verb analysis' is also applied to use cases (e.g., Holland & Lieberherr, 1996). Nouns, particularly things, persons or roles, events, places and interactions, are possible classes. But Jacobson et al. (1994, p. 105) state: "when we say that we identify and describe a use case, we mean that we identify and describe the class." This suggests that whoever is writing use cases should have a reasonable understanding of what classes are and what ones are likely to emerge during analysis. Interestingly, using nouns to identify classes of objects or entities for an application predates UML by a large period, and has been advocated for data modeling for many years. In contrast, some others have suggested the class model (or at least an initial attempt) ought to precede the creation of use cases. Pooley and Stevens (1999), for example, offer a detailed description of methods for identifying classes. They describe a process of identifying nouns in a systems requirement document as a mechanism for identifying candidate classes for an application (p. 58). These nouns may come from use case descriptions or other requirements documents, although Pooley and Stevens are silent on the source and nature of these documents. Rosenberg and Scott (1999, p. 16-17) search for nouns and verbs in "available relevant material," which includes the "problem statement, lower-level requirements, and expert knowledge," along with other sources such as marketing literature. They also identify classes before writing use cases. Booch (1994) similarly advocates the use of noun analysis to identify classes.

Indeed, Pooley and Stevens (1999) indicate a potential problem with use cases as a component of UML:

> Use case modeling should be used with caution, however, since … [t]here is a danger of building a system which is not object-oriented. Focusing on use cases may encourage developers to lose sight of the architecture of the system and of the static object structure. (p. 101)

Moreover, they go on to state "we do not believe that examination of the use cases is *on its own* a good way to find objects and classes" (p. 102, emphasis at source).

Meyer (1997, p. 738) also states that, "use cases are not a good tool for finding classes." One reason is that use cases emphasize procedural sequences and this is at best irrelevant to class modeling and could even be dangerous to the process. Other concerns are that users will either tend to develop use cases around what is happening now, thus failing to consider reengineering of the process, or will simply revert to functional design. However, Meyer believes that use cases can be effectively employed as a validation tool and implementation guide. The final system must be capable of handling the scenarios identified by users, although perhaps not in the same way as they originally envisioned.

Another approach to modeling classes is the use of CRC cards (Beck & Cunningham, 1989; Pooley & Stevens, 1999). While not specifically part of UML, they can be used to model the required functionality responsibilities and association collaborations of classes once the classes that are needed have been identified.

In summary, the process for moving forward from the use case model to identify classes is neither universally accepted, even among use case adherents, nor does it appear to be clearly defined or articulated. Proposed techniques, such as noun identification, are rooted in older techniques from data modeling. The lack of integration between use cases and class models raises questions about the value of use cases in an object-oriented modeling approach.

Objects Versus Processes

A use case is inherently task focused. It describes a sequence of activities, from start to finish, involved in completing a well-defined task or transaction. As in any task, many participants may be involved in the successful completion of a use case. These participants are candidates for objects that will be important to the system. A task or process focus, however, involves participants only to the extent that they contribute to the task. Hence, a use case involves objects only peripherally and only as needed for the task being modeled. Therefore, a complete use case model may not offer a cohesive picture of the structural and behavioral characteristics of the objects in the domain. Instead, these characteristics may be spread over several use cases.

The fragmentation across use cases of information needed to construct class definitions conceptually violates the principle of *encapsulation*, widely recognized as one of the cornerstones of object orientation. As a result, it can create a significant amount of work for analysts and developers in 'defragmentation,' or reconstructing classes from a potentially large number of narrowly focused views that might be embedded in many different use cases. Although we are not aware of empirical research, or even anecdotal reports, on the extent of this problem, a case can be made that the task can be daunting. The problem is analogous to the issue of *view integration* in database design (Navathe, Elmasri, & Larson, 1986). There, the issue is one of developing a global conceptual schema from a set of diverse user views of the kinds of entities about which data need to be kept. Since different users have different needs, they generally have a different perspective on which entities are important, and how they are defined in terms of attributes and relationships. Problems to be resolved include identifying synonyms (entities, attributes, and/or relationships with the same meaning that have different names in different views) and homonyms (entities, attributes, and/or relationships with different meanings that have the same name in different views).

Similar problems are possible when identifying object classes, their attributes, and their operations from a series of use cases. Given that different use cases are likely to be relevant to different users of a system, it is reasonable to expect that resolving synonyms and homonyms will impede the comprehensive and consistent identification of objects from use cases. Consequently, we propose that identifying a comprehensive and consistent class model from use cases alone will be very difficult, if not practically impossible.

USE CASES AS A COMMUNICATION MECHANISM

Isolating Users from the Class Model

In view of the apparent lack of 'object' focus in use cases and the potential problems that can arise in deriving a class model from a use case model, it is natural to question the rationale for including use cases in UML. This is particularly interesting since use cases are a relatively recent addition to UML. Much of the rationale for adopting use case modeling in UML focuses on their simplicity and the fact that they are "comparatively easy to understand intuitively, even without knowing the notation. This is an important strength, since the use case model can sensibly be discussed with a customer who need not be familiar with the UML" (Pooley & Stevens, 1999, p. 93). This view suggests that other UML models, in particular the class model, are too technical for end users to understand or be capable of verifying.

Communication with the system's intended users is clearly an important, if not always explicitly articulated, goal of use cases. A use case model provides an inventory of the kinds of interactions that can occur between users and a system, providing "a forum for your domain experts, end users, and developers to communicate to one another" (Booch et al., 1999, p. 229). Use cases are thus oriented towards interaction with end users for the purpose of verifying the developers' understanding of how a system works or will work.

This understanding is essential for effective system development, and also helps create a "shared understanding" among team members that is a critical part of the trust building process (Ring & Van de Ven, 1989). Text may be easier to understand than diagrams, at least to an untrained user. Thus, use cases could contribute both to the accuracy of the requirements specification and also to its apparent openness. The analyst does not appear to be hiding behind diagrams that only IS professionals can understand.

In discussing the value of use cases in reengineering business processes, Jacobson et al. (1994) similarly explain the role of the use case in communicating with users or those responsible for a business process:

> Use cases are best described using simple language to facilitate understanding. … The rightful owner, that is, the defined business process owner for the use case, will thereafter validate each use case's compliance with the established corporate objectives. (p. 178)

Here, use cases are clearly established as a tool for communicating and verifying with users the developers' understanding of how tasks are performed. In contrast, they clearly see the verification of class or object models as the purview of developers:

> The reviewers are normally people in the reengineering team. It is unusual to communicate

> the object models to the employees in general, which means that the only people who are really involved and competent to review these models are in the reengineering team. (p. 190)

Taken together, these statements suggest that use cases are an appropriate mechanism to 'shield' users from the underlying technical UML models that are the basis for systems design and implementation.

The need to exclude users from direct exposure to the class model in particular highlights an interesting contradiction in UML. One of the main arguments offered for developing object-oriented approaches to systems analysis and design is that objects provide a "natural" way of thinking about a problem domain. In this regard, Booch (1996, p. 39) notes that "in a quality object-oriented software system, you will find many classes that speak the language of the domain expert" and "(e)very class in an object-oriented system should map to some tangible or conceptual abstraction in the domain of the end user or the implementer." Jacobson et al. (1992) make the case more directly:

> People regard their environment in terms of objects. Therefore it is simple to think in the same way when designing a model. A model which is designed using an object-oriented technology is often easy to understand, as it can be directly related to reality. Thus, with such a design method, only a small **semantic gap** (emphasis at source) will exist between reality and the model. (p. 42, emphasis at source)

The previous discussion shows that, despite this avowal of the naturalness and ease of understanding of UML models, the developers of the language explicitly introduce use cases as the primary mechanism for communicating with users to verify understanding of system functionality.

Use Cases Versus Class Models for Communication

The contradiction highlighted above can be dealt with in at least two ways. First, there is significant literature in cognitive psychology to support the contention that people think about the world in terms of things that are classified in particular categories (e.g., Medin & Smith, 1984). Lakoff (1987) views such category structures as vital for human survival, arguing that "(w)ithout the ability to categorize, we could not function at all" (p. 1). Parsons and Wand (1997) apply categorization research to analyze those aspects of object orientation that are meaningful from a systems analysis perspective, and conclude that classification is a vital element for object-oriented analysis.

From a cognitive perspective, one would expect that users should be able to handle class models as a mechanism for communicating with developers in verifying the conceptual structure of the domain being modeled. Of course, issues such as the difficulty of learning the notation associated with a particular class modeling technique can negatively influence communication. Nevertheless, the fundamental idea that a domain can be described in terms of the kinds of objects in it, the attributes of the objects, the behavior the objects can exhibit, and the associations among kinds of objects, is highly consistent with research on the nature of categories that people use to structure their knowledge about things in the world. Consequently, we hypothesize that end users will be able to interact directly with class models in verifying the structure of a domain.

Cognitive psychology also provides a second basis for understanding the contradiction inherent in advocating use cases as the primary mechanism for communicating and verifying system requirements with users. Advocates of use cases point to the ease with which they can be understood, as they describe a process from start to finish. Not surprisingly, a significant body of research in cognitive science deals with how people think procedurally. For example, Schank and Abelson's (1977) work on scripts deals with the sequencing of ordinary and exceptional events involved in a goal-oriented activity. Scripts provide a mechanism by which people can understand the temporal relationship in a list of events, including inferences about events that are not explicitly stated in a description (Bower, Black, & Turner, 1979).

Since people can think in either process-oriented or object-oriented modes, we postulate that both process-oriented and object-oriented models can be understood by users and are appropriate for verifying different aspects of application requirements. This suggests that advocating use cases for work with users, while isolating users from the class models that are the primary basis for the design of an object-oriented architecture, is not necessary. Moreover, the peripheral and diffuse role of objects in use cases is a potential source of difficulty in developing class models from use cases and verifying whether they are a good model of the domain's category structure as understood by users. It may be more appropriate to use class models directly as a mechanism for communicating and verifying the structure of the application domain with users.

CALL FOR RESEARCH

The analysis presented above is purely theoretical. As far as we are aware, advocates of use cases do not offer empirical evidence that they are a 'good' mechanism for communicating with users. 'Goodness' of use cases could be ascertained by developing a standard of effective communication against which use cases can be evaluated. Alternatively, 'goodness' could be established in a relative sense by comparing them to other mechanisms for communicating the same information with users. At present, the value of use cases has not been established empirically in either of these senses.

Similarly, although we have presented an argument that use cases may be inadequate for developing class models, such inadequacy has not been demonstrated empirically. In addition, although research on classification points to the naturalness of category structures in organizing information about things in the world, there are few empirical studies addressing the ability of users to understand class models. The few studies that have addressed this were conducted prior to the development of the particular class modeling technique that is part of UML. For example, Vessey and Conger (1994) found that novice analysts were better able to specify requirements using process- and data-oriented methodologies than using object-oriented methodologies.

In addition, we have identified the growing tendency for use cases to include design or implementation decisions that could be a possible impediment to effective process design in systems development. Despite the attention paid by some to the role of use cases in process reengineering, there is reason to believe that popular use case structures may anchor developers to particular solution approaches and thereby narrow the scope of possible solutions considered. However, there is no empirical evidence that such adjustment biases occur in practice.

In view of the movement toward UML as a standard modeling language in practice, the paucity of empirical research on the effectiveness of various modeling techniques and prescriptions in UML is troubling. We have offered a theoretical framework for studying three issues: premature inclusion of design decisions, the adequacy of use cases for extracting class models, and the justification for choosing use cases as the primary mechanism for developer interaction with users. From these perspectives, we think it is important to conduct a range of empirical studies to evaluate the various modeling components of UML.

First, research is needed to examine whether including design and implementation details in use cases leads to anchoring and adjustment problems with respect to effective process redesign. This question can be addressed directly through lab experiments in which developers design a system starting from either abstract use cases or use cases in which design or implementation decisions are stated. In each group, the 'innovativeness' of the resulting designs relative to existing processes can be measured. To measure the external validity of such results, correlational field studies of object-oriented development using UML can also be undertaken to measure the relationship between the structure of use cases used and the extent to which implementations achieve effective redesign.

Second, research is needed to test the assertion that, since use cases do not focus on objects, it will be difficult to extract a class model from a set of use cases. Although it may be possible to test this in a controlled laboratory experiment, it would be difficult to avoid biases in the development of use cases that might influence the ability to extract class models. Consequently, an appropriate method for examining the degree to which use cases support the development of class models (and, more generally, how class models are developed and verified) would be surveys and/or case studies of the progression from use case models to class models in projects that use UML. Among the variables to measure are: the extent to which use cases are the exclusive mechanism for communication and verification of requirements with users; the extent to which use cases drive the development of the class model; problems encountered in using use cases to develop the class model; perceptions about the causes of such problems; and approaches that are used to deal with these problems.

Third, research is needed to examine whether users are capable of directly reading and understanding class models, as well as other UML models. In addition, there is a need to study whether use cases add value (e.g., in ease of understanding or ability to capture additional information relative to other models in UML). For this type of study, laboratory experiments offer the ability to enforce necessary control to permit useful comparisons across groups. Several issues need to be resolved in conducting this kind of study. For example, use cases include process or task information, while class diagrams do not. Hence, comparisons between use cases and class models must be restricted to object/attribute/relation-

Table 1: A Framework for Empirical Research on Use Cases

Research Question	**Primary Independent Variable**	**Primary Dependent Variable**	**Methodology**
Do design/implementation details in use cases impede process redesign efforts?	Use case structure	Process innovation	Experiment; Case study
Can class models be effectively extracted from use cases?	Use cases	Class model completeness	Case study; Developer surveys
Do use cases facilitate communication between developers and users?	Communication medium (use cases or class models)	User understanding Domain coverage	Experiments; User surveys

ship identification, or class models must be used in conjunction with other UML models to conduct comprehensive comparisons with use cases.

Table 1 summarizes a research framework for studying the need for, and effectiveness of, use cases in UML.

CONCLUSIONS

UML is a modeling language for object-oriented development that grew out of the combination of three distinct approaches developed in the early 1990s. Much of the conceptual foundation of the language comes out of issues in object-oriented programming (Booch, 1994), and there is little evidence about the extent to which it is appropriate as a language for modeling an application domain or system requirements. In short, we feel there is a strong need for academic research to evaluate the usefulness of UML and determine its limitations for modeling requirements. Here, we have offered a framework for evaluating the roles of, and relationships between, use cases and class models in the UML. Similar research is needed to understand the capabilities and limitations of the other models in the language.

REFERENCES

Beck, K., & Cunningham, W. (1989). A Laboratory for Teaching Object-Oriented Thinking. *ACM SIGPLAN Notices, 24*(10), 1-6.

Blaha, M., & Premerlani, W. (1998). *Object-Oriented Modeling and Design for Database Applications.* Upper Saddle River, NJ: Prentice Hall.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications* (2nd ed.). Redwood City, CA: Benjamin/Cummings.

Booch, G., (1996). *Object Solutions: Managing the Object-Oriented Project.* Reading, MA: Addison-Wesley.

Booch, G., Jacobson, I., & Rumbaugh, J. (1999). *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley.

Bower, G., Black, J., & Turner, T. (1979). Scripts in Memory for Text. *Cognitive Psychology, 11,* 177-220.

Brown, D. (1997). *An Introduction to Object-Oriented Analysis: Objects in Plain English.* New York: John Wiley & Sons.

Coad, P. (1995). *Object Models: Strategies, Patterns, and Applications*. Englewood Cliffs, NJ: Yourdon Press.

Constantine, L. L., & Lockwood, L. A. D. (2000). Structure and Style in Use Cases for User Interface Design. In M. Van Harmelen & S. Wilson (Eds.), *Object Modeling User Interface Design*. Reading, MA: Addison-Wesley. (In press) Available: http://www.foruse.com. (June 12, 2000).

Dewitz, S. (1996). *Systems Analysis and Design and the Transition to Objects.* New York: McGraw-Hill.

Evans, G. (1999). Why Are Use Cases So Painful? *Thinking Objects,* [on line serial], *1*(2). Available: http://evanetics.com/TONewsletters/thinking-v1n2.htm. (June 12, 2000).

Hammer, M., & Champy, J. (1993). *Reengineering the Corporation: A Manifesto for Business Revolution.* New York: HarperCollins.

Harmon, P., & Watson, M. (1998). *Understanding UML: The Developer's Guide.* San Francisco: Morgan Kaufmann.

Holland, I., & Lieberherr, K. (1996). Object-Oriented Design. *ACM Computing Surveys*, *28*, 273-275.

IBM Object-Oriented Technology Center (1997). *Developing Object-Oriented Software.* Upper Saddle River, NJ: Prentice Hall.

Jacobson, I. (1987). Object-Oriented Development in an Industrial Environment. *OOPSLA'87 Conference Proceedings, SIGPLAN Notices, 22*(12), 183-191.

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

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach.* Reading, MA: Addison-Wesley.

Jacobson, I., Ericsson, M., & Jacobson, A. (1994). *The Object Advantage: Business Process Reengineering with Object Technology.* Reading, MA: Addison-Wesley.

Kilov, H., & Ross, J. (1994). *Information Modeling: An Object-Oriented Approach.* Englewood Cliffs, NJ: Prentice Hall.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* Chicago: University of Chicago Press.

Larman, C. (1998). *Applying UML And Patterns: An Introduction to Object-Oriented Analysis and Design.* Upper Saddle River, NJ: Prentice Hall.

Lockheed Martin Advanced Concepts Center and Rational Software Group. (1996). *Succeeding With The Booch And OMT Methods: A Practical Approach.* Menlo Park, CA: Addison-Wesley.

MacMaster, B. (1997). Saving time with 'use cases.' *Computing Canada, 23*(21), 52.

Medin, D., & Smith, E. (1984). Concepts and Concept Formation. *Annual Review of Psychology, 35*, 113-138.

Meyer, B. (1997). *Object-Oriented Software Construction.* Upper Saddle River, NJ: Prentice Hall.

Navathe, S., Elmasri, E., & Larson, J. (1986). Integrating User Views in Database Design. *IEEE Computer*, 19, 50-62.

Norman, R. (1996). *Object-Oriented Systems Analysis And Design.* Upper Saddle River, NJ: Prentice Hall.

Parsons, J., & Wand, Y. (1997). Using Objects in Systems Analysis. *Communications of the ACM, 40*(12), 104-110.

Plous, S. (1993). *The Psychology of Judgment and Decision Making.* New York: McGraw-Hill.

Pooley, R., & Stevens, P. (1999). *Using UML: Software Engineering with Objects and Components.* Reading, MA: Addison-Wesley.

Ring, P. S., & Van de Ven, A. H. (1989). Formal And Informal Dimensions Of Transactions. In A. H. Van de Ven, H. L. Angle, & M. S. Poole (Eds.), *Research on the Management of Innovation: The Minnesota Studies* (pp. 171-192). New York: Harper & Row.

Rosenberg, D., & Scott, K. (1999). *Use Case Driven Object Modeling with UML.* Reading, MA: Addison-Wesley.

Rubin, K., & Goldberg, A. (1992). Object Behavior Analysis. *Communications of the ACM, 35*(9), 48.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & W. Lorensen (1991). *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall.

Satzinger, J., & Orvik, T. (1996). *Object-Oriented Approach: Concepts, Modeling, and System Development.* Danvers, MA: boyd & fraser.

Schank, R., & Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding.* Hillsdale, NJ: Erlbaum.

Schneider, G., & Winters, J. P. (1998). *Applying Use Cases: A Practical Guide.* Reading, MA: Addison-Wesley.

Tversky. A., & Kahneman, D. (1974). Judgment Under Uncertainty: Heuristics and Biases. *Science, 185*, 1124-1131.

Vessey, I., & Conger, S. (1994). Requirements Specification: Learning Object, Process, and Data Methodologies. *Communications of the ACM, 37*(5), 102-113.

White, I. (1994). *Rational Rose Essentials: Using the Booch Method.* Redwood City, CA: Benjamin/Cummings.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). *Designing Object-Oriented Software*. Englewood Cliffs, NJ: Prentice Hall.

Endnote

1 This research was supported in part by a research grant from the Social Sciences and Humanities Research Council of Canada to Jeffrey Parsons.

Brian Dobing *is an assistant professor, Faculty of Management, University of Lethbridge, Lethbridge, AB, Canada. He received his B.Sc. in mathematics from the University of Victoria, his M.B.A. and M.Sc. in Computational Science from the University of Saskatchewan, and his Ph.D. from the University of Minnesota. His research focuses on issues in user-analyst relationships and object-oriented analysis.*

Jeffrey Parsons *is an associate professor, Faculty of Business Administration, Memorial University of Newfoundland, St. John's, NF, Canada. He received his B.Com.(Hon.) from Memorial University of Newfoundland and his Ph.D. in Information Systems from the University of British Columbia. His research has been published in journals such as ACM Transactions on Database Systems, Communications of the ACM, and Management Science. His current research interests include data modeling, object-oriented analysis and design, and electronic commerce.*