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

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
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 WorkFlow 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