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 chapter 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
Related Content

On Temporal Summation in Chaotic Neural Network with Incremental Learning

Petri Net Based Deadlock Prevention Approach for Flexible Manufacturing Systems
[www.irma-international.org/chapter/petri-net-based-deadlock-prevention/50437/](http://www.irma-international.org/chapter/petri-net-based-deadlock-prevention/50437/)

A Framework for Intelligent Service Discovery
[www.irma-international.org/chapter/framework-intelligent-service-discovery/28120/](http://www.irma-international.org/chapter/framework-intelligent-service-discovery/28120/)

Timers and Associated Hardware
[www.irma-international.org/chapter/timers-and-associated-hardware/190450/](http://www.irma-international.org/chapter/timers-and-associated-hardware/190450/)

Using Executable Slicing to Improve Rogue Software Detection Algorithms
Jan Durand, Juan Flores, Travis Atkison, Nicholas Kraft and Randy Smith (2011). *International Journal of Secure Software Engineering* (pp. 53-64).
[www.irma-international.org/article/using-executable-slicing-improve-rogue/55269/](http://www.irma-international.org/article/using-executable-slicing-improve-rogue/55269/)