# Chapter 36
# Reengineering Structured Legacy System Documentation to UML Object-Oriented Artifacts

**Terrence P. Fries**
*Indiana University of Pennsylvania, USA*

## ABSTRACT

*The need for reengineering of software systems has dramatically increased as legacy systems are migrated to new platforms and rewritten in modern object-oriented languages. Although the de facto standard for describing object-oriented systems is the Unified Modeling Language (UML), many legacy systems have been documented using non-object-oriented structured analysis and design methods. Problems arise in the migration because non-object-oriented documentation is inherently not conducive to the development of object-oriented systems. This chapter presents a set of rules to automate the conversion of systems which were originally modeled using structured techniques to UML. The newly created UML documentation can then be used in developing an object-oriented equivalent system. The UML model may also be used by computer aided software engineering tools to implement a new system. The reengineering rules are tested on an example structured system to demonstrate their viability.*

## INTRODUCTION

Software reengineering was defined by Chikovsky and Cross as "the examination and alteration of a system to reconstitute it in a new form" (1990). The need for software reengineering has drastically increased as legacy systems become obsolete in terms of their architecture, the platforms on which they run, and their maintainability. Software engineering is important for recovering and reusing existing software assets, reducing high software maintenance costs, and establishing a stable, maintainable base for future software evolution.

DOI: 10.4018/978-1-4666-4301-7.ch036

Software change is inevitable as the Y2K problem demonstrated. Changes in currency, government regulations, technology changes, and new and constantly changing business requirements necessitate modifications to the software in legacy systems. As systems evolve over time, they inevitably undergo changes that result in the degeneration of the original architecture. In an effort to expedite system modifications, documentation of the changes is often ignored or incomplete. The degeneration may be so severe that further changes result in an unstable system. When systems reach this state they require a complete resign. In addition, many legacy systems were developed for platforms that have become obsolete (Hochstein & Lindvall, 2005). When a legacy system requires migration to a new platform or redesign, the software development paradigm is almost exclusively that of object-oriented (OO) analysis and design (Schach, 2010; Pressman, 2009; Booch, Maksimchuk, Engel, Young, Conallen, & Houston, 2007). The OO paradigm is applicable for such modern platforms as Web-based applications, distributed systems, component-based systems, model-driven architecture (MDA), and service oriented architecture (SOA).

Due to its widespread use, the unified modeling language has become the de facto standard for modeling the architecture and behavior of software systems (Booch, Jacobson & Rumbaugh, 2005; Rumbaugh, 2004; Fowler, 2003). UML is also the visual modeling tool for the widely accepted object-oriented and design paradigm referred to as the Unified Process (Jacobson, Booch & Rumbaugh, 1999; Arlow & Neustadt, 2007). While UML is usually associated with object-oriented systems, its use for non-object-oriented systems is becoming common. However, many legacy systems have been documented with for non-object-oriented methods, primarily structured analysis and structured design (SASD) (DeMarco, 1978; Gane & Sarson, 1977; Yourdon, 1989; Yourdon & Constantine, 1989). The most common artifacts produced by SASD are data flow diagrams (DFD) and entity relationship diagrams (ERD).

There are many problems in understanding a legacy system. They may be written in an outdated programming language and may contain numerous modifications. Modifications are frequently "kludges" (clumsy or inelegant solutions to a problem) that make the program impossible to comprehend. Software engineers require UML documentation to aid in the maintenance of these systems.

As legacy systems are reengineered to use modern object-oriented languages and techniques, it is necessary to convert the existing SASD models to UML. Documentation is essential in the understanding of legacy systems. UML diagrams have been shown to be an effective aid in program understanding. The new UML model can act as the basis for design and implementation of the new system, as well as documentation for the future evolution of the new system. In addition, computer aided software engineering tools can use the UML format for quality assurance and automatic code generation.

This chapter defines a set of formal rules for reengineering a legacy system document using SASD data flow diagrams and entity relationship diagrams into an object-oriented system defined by UML 2.0 artifacts. The Background section presents the transformation framework. The Reengineering Framework and Case Study sections illustrate and confirm the correctness of the framework by applying it to a test case. The Conclusions section summarizes the research and discusses future work.

## BACKGROUND

Many attempts have been made to reengineer legacy systems to newer platforms and programming languages. These approaches fall into three general categories: language specific, architecture specific, and language/architecture independent.