Chapter XIII
Software Modeling Processes: UML–xUML Review

Roy Gelbard
Bar-Ilan University, Israel

ABSTRACT

Applications require short development cycles and constant interaction with customers. Requirement gathering has become an ongoing process, reflecting continuous changes in technology and market demands. System analysis and modeling that are made at the initial project stages are quickly abandoned and become outmoded. Model driven architecture (MDA), rapid application development (RAD), adaptive development, extreme programming (XP), and others have resulted in a shift from the traditional waterfall model. These methodologies attempt to respond to the needs, but do they really fulfill their objectives, which are essential to the success of software development? Unified modeling language (UML) was created by the convergence of several well-known modeling methodologies. Despite its popularity and the investments that have been made in UML tools, UML is not yet translatable into running code. Some of the problems that have been discovered have to do with the absence of action semantics language and its size. This chapter reviews and evaluates the UML evolution (UML2, xUML), providing criteria and requirements to evaluate UML and the xUML potential to raise levels of abstraction, flexibility, and productivity enhancement. At the same time, it pinpoints its liabilities that keep it from completely fulfilling the vision of software development through a continuous exactable modeling process, considered to be the future direction for modeling and implementation.

INTRODUCTION

In his book, Evitts describes the beginnings of UML tools (Evitts, 2000). The context prompting the development of UML was the increasing complexity of software which began in the 90s, when technologies (tools) that could deal with a network and information-driven world did not yet exist. In 1991, Malone and Rockart described expectations that would soon emerge from all quarters. They noted that whenever people work together, there is a need to communicate so as to make decisions, allocate resources, and provide and receive products and services at the right time and place. However,

in the early 90s, methodologies were rarely supported, either by common modeling tools, traditional methodologies (based upon process charts, ERD, and DFD), or object oriented methodologies. The semi-standard development process, the "waterfall," was convenient, albeit unperfected, whereas object-oriented provided none of these comforts, and the general opinion was that very few of its efforts had any real advantages over mainstream approaches.

In early 90s, the rise of Java, the standardization of C++, the birth and rebirth of CORBA, and the emergence of pattern languages for software design attracted a great deal of attention and popularity to UML. In June 1996, Rational released the 0.9 revision of UML, and then later on January 1997, Rational's 1.0 spec reached the market. In September 1997, Rational's UML 1.1 was combined with the OMG's UML proposal to create the final product that was called UML 1.0.

The current chapter evaluates the extent to which the UML can be used to support the modeling process, providing not only better communication among system analysts and developers. Primarily, it examines productivity enhancement through generating capabilities of wider range of software elements based upon modeling definitions.

BACKGROUND REVIEW

A. From UML 1 to UML 2.0

The scope of the UML has recently broadened. It is no only longer used to describe software systems, but now also business processes. With the service-oriented architect (SOA) and model driven architecture (MDA) initiatives, it has evolved to describe and automate business processes (activity diagram is a UML variation of the traditional process diagram), as well as become a language for developing platform-independent systems.

Earlier versions of the UML standard did not describe what it meant to support the standard. As a result, UML tool vendors were free to support incomplete UML features, and converting models from one tool to another was often extremely difficult, if not impossible.

UML 2.0 defines 38 compliance points (Ambler, 2004; Bjorkander & Kobryn, 2003). A compliance point is an area of UML, such as use cases. All implementations are required to implement a single compliance point, the kernel. The other 37 compliance points are currently optional. Evaluating modeling tools in light of these compliance points helps clarify which model elements are supported, and to what extent. For each compliance point, there are four compliance options. A compliance option determines how compliant a given implementation is. The four options are as follows:

- **No compliance**—the implementation does not comply with the syntax, rules, and notation for a given compliance point.
- **Partial compliance**—the implementation partially complies with the syntax, rules, and notation for a given compliance point.
- **Compliant compliance**—the implementation fully complies with the syntax, rules, and notation for a given compliance point.
- **Interchange compliance**—the implementation fully complies with the syntax, rules, notation, and XMI schema for a given compliance point.

However, UML 2.0 does not address any of UML 1.x's significant deficiencies, namely the lack of **business rule** Modeling, **workflow** modelling, and **user interface** modeling, although there is a business rule working group within the OMG. Several methodologists have suggested approaches to user interface flow modeling and design modeling using UML, but no official effort to develop a common profile exists.

B. Executable UML (xUML)

xUML is a subset of the UML, incorporating action language that allows system developers to build ex-