Chapter 14
A Software Cost Model to Assess Productivity Impact of a Model-Driven Technique in Developing Domain-Specific Design Tools

Achilleas Achilleos
University of Cyprus, Cyprus

Nektarios Georgalas
British Telecom (BT) Innovate, UK

Kun Yang
University of Essex, UK

George A. Papadopoulos
University of Cyprus, Cyprus

ABSTRACT

Programming languages have evolved through the course of research from machine dependent to high-level "platform-independent" languages. This shift towards abstraction aims to reduce the effort and time required by developers to create software services. It is also a strong indicator of reduced development costs and a direct measure of a positive impact on software productivity. Current trends in software engineering attempt to raise further the abstraction level by introducing modelling languages as the key components of the development process. In particular, modelling languages support the design of software services in the form of domain models. These models become the main development artefacts, which are then transformed using code generators to the required implementation. The major predicament with model-driven techniques is the complexity imposed when manually developing the domain-specific design tools used to define models. Another issue is the difficulty faced in integrating these design tools with

DOI: 10.4018/978-1-60960-215-4.ch014

model validation tools and code generators. In this chapter a model-driven technique and its supporting model-driven environment are presented, both of which are imperative in automating the development of design tools and achieving tools integration to improve software productivity. A formal parametric model is also proposed that allows evaluating the productivity impact in generating and rapidly integrating design tools. The evaluation is performed on the basis of a prototype domain-specific design tool.

INTRODUCTION

The escalating and rapidly changing user requirements contribute towards increased complexity in the software development process. Furthermore, the advancements and diversity in technologies currently present escalate further the complexity introduced to the process. Consequently, the software engineering community seeks innovative and abstract techniques that provide the capability to scale down the complexity problem, in order to simplify and expedite the development of domain-specific software services. The objective is to provide "platform-independent" techniques that support the creation of software services at an abstract level steering the developer away from platform-specific implementation complexities.

During the early years of *Software Engineering* the difficulties and pitfalls of designing complex software services were identified and a quest for improved software development methodologies and tools began (Wirth, 2008). The first steps towards this goal introduced formal notations, known as programming languages, used mainly for performing mathematical analysis computing tasks. Examples of such numerical programming languages are *FORTRAN, Algol and COBOL*. Since then demand for more powerful software applications that perform complex computational tasks, rather than simple mathematical tasks, has largely grown. Therefore, it was acknowledged that more competent programming languages, software tools and automation capabilities were required to successfully implement these complex computing tasks (Wirth, 2008).

The software engineering discipline concentrated on the development of high-level programming languages, which simplify the development of software applications. A minor setback in the inclination towards programming abstraction was the machine dependent *C language*. As Wirth (2008, p. 33) clearly states:

"From the point of view of software engineering, the rapid spread of C therefore represented a great leap backward..... It revealed that the community at large had hardly grasped the true meaning of the term "high-level language", which became a poorly understood buzzword. What, if anything, was to be "high level" now?"

Although the *C language* provides efficiency in creating simple hardware-dependent software services, it proved scarce and complex in developing, testing and maintaining large and versatile software applications (Wirth, 2008). The lessons learned from using the *C language* guided though software engineers to devise abstract and disciplined software techniques, like the predominant *Object-Oriented (OO)* programming model (Chonacky, 2009). On the basis of this model different 3GLs were developed such as Smalltalk, C++, Java and C#. These languages aimed to raise the level abstraction in software engineering and facilitate the definition of disciplined, systematic and object- oriented techniques for software development. 3GLs allow building advanced software