An Aspect-Oriented Implementation Method
with support to Progressive Implementation

Sérgio Soares¹*  Paulo Borba¹
¹Software Productivity Group
Centro de Informática - Universidade Federal de Pernambuco
email: [scbs,phmb]@cin.ufpe.br

Abstract

Usually, researchers and software engineers do not give much attention to implementation methods because implementation mistakes have less impact in project schedule and development costs than mistakes during requirements and design. However, the effort given to requirements and design can be wasted if there is not a commitment with the implementation activities. The main goal of this research is to define an implementation method using aspect-oriented programming, helping to achieve better software with higher productivity levels. The implementation method defines patterns to structure the system architecture, aspects frameworks to support implementing the crosscutting concerns, and implementation activities integrated with Use Case Driven Development.

1. Introduction

Usually, researchers and software engineers do not give much attention to implementation methods [1], because implementation mistakes have less impact in project schedule and development costs than requirements and design mistakes. However, the effort given to requirements and design can be wasted if there is not a commitment with the implementation activity. This is necessary in order to increase productivity, reliability, reuse, and extensibility levels. For example, the maintenance activity usually has the highest cost, which is inversely proportional to reuse and extensibility. This motivates the continuous search to increase those levels.

The need for quality software motivated the use of object-oriented programming [2] towards higher reuse and maintainability levels, increasing development productivity and support for requirements change. However, the object-oriented paradigm presents a number of limitations [6], such as tangled and spread code across different concerns. Examples are, business code tangled with presentation code or data access code, and distribution, concurrency control, and exception handling code spread over several classes. To solve these limitations, techniques, like aspect-oriented programming, aim to increase software modularity in practical situations where object-oriented programming does not offer an adequate support.

We believe that the aspect-oriented paradigm [3], is very promising [5, 9]. The paradigm tries to solve the inefficiency in capturing some of the important design decisions that a system must implement. This difficulty leads the implementation of these design decisions spread through the functional code, resulting in tangled code with different concerns. This tangling and scattering code hinders development and maintenance of these systems. Therefore, an aspect-oriented software development increases modularity by separating code that implements specific functions and affects different parts of the system. These are called crosscutting concerns.

*Supported by CAPES
2. Aspect-Oriented Implementation Method

No matter how good the programming language, an implementation method is important to define activities to be executed and the relations between them, including their execution order. We defined an implementation [7] method using aspect-oriented programming, helping to develop better software with better productivity levels. Our implementation method guides the implementation of persistence, distribution, and concurrency control concerns that conforms to specific software architecture. Despite being specific, the software architecture can be used to implement several kinds of systems. The method has an aspect framework, implemented in AspectJ, and tool support to implement those non-functional requirements.

In addition, the implementation method allows adopting an alternative implementation approach, so called progressive implementation [8, 7], which increases implementation productivity and decreases implementation complexity. This alternative approach anticipates requirements change by early validating a functional prototype, without any effort to implement persistence, distribution and concurrency control. The productivity increases because if any functional requirements change, these non-functional requirements have not to be changed, since they were not implemented.

We also provide how the implementation method can be combined to Use-case Driven Development (UCDD) [4], a wide used development approach, in order to incorporate the idea of progressive implementation [7].

References