Reengineering a 2-Tier Database Application with Software Architecture

Hong G. Jung
Advisor: Sam Chung
Quality Engineering Management
College of Engineering
Southern Illinois University, Carbondale, IL
{trampsta, samchung}@siu.edu

Keywords:
Software Reengineering, Software Architecture, MVC, 3-Layered Architecture

Brief Description:
The purpose of this research is to demonstrate how to reengineer a legacy Database Application using to a target system with MVC and 3-layered architecture. A Coffee Inventory Management database application is used for legacy application. The benefits of the reengineering are discussed.

Abstract

One of the major concerns of learning Java is that many examples in Java textbooks do not cover the concept of software architecture and its development practice in industry. Even, most examples focus on 1-tier application using monotonic computing. However, most applications in industry use at least 2 or 3 tier distributed computing. Without using software architecture, the students cannot see how software architecture of a distributed application can reduce future maintenance efforts, especially for businesses. In this research, we demonstrate how a database application using 2-tier distributed computing can be reengineered by using MVC and 3-layered (Presentation, Business logic, and Data access) architecture. For this purpose, we choose a database application named Coffee Inventory Management from a reliable Java reference book. By using a software re-documentation methodology called 5W1H Re-Doc with a Computer-Aided Software Engineering (CASE) tool, we create a visual model in Unified Modeling Language (UML) for the legacy application that consists of 4+1 views - Use Case, Design, Process, Implementation, and Deployment view. Based upon the created legacy model, we create a target model using MVC. The legacy system is reengineered with two architectural patterns – MVC and 3-layered. Based upon the visual model for the expected target system, we first redesign the legacy system with MVC architecture by reducing redundancy and repetition within the code. Second, we then apply the 3-layered architecture to the model part of MVC to separate the model part into business logics and data accesses. Based upon the revised target model, we implement the reengineered Coffee Inventory Management with MVC architecture to the final target system with 3-layered architecture. We conclude that MVC and 3-layered architecture could help to reduce the redundancy and repetition code in the distributed application by emphasizing separation of concerns in terms of MVC and 3-layered architecture.