Chapter 12
Legacy Systems towards Aspect-Oriented Systems

Noopur Goel
VBS Purvanchal University, India

ABSTRACT

Evolution and maintainability of legacy systems is all time attention drawing subject for researchers and especially practitioners. Discovering the crosscutting concerns and separating it from core functionalities of a software system may help in evolution of the legacy systems. Aspect-oriented software development (AOSD) tries to achieve the goal. AOSD is new programming paradigm which helps to bring in modularity in the program by writing the crosscutting concerns in the form of 'aspects'. Modularity brings comprehensibility and hence maintainability of the software system. Tools and techniques, which aid in identifying the crosscutting concerns in such systems and refactoring them into aspects, are needed to apply aspect-oriented techniques to legacy systems at use in industry. This chapter aims to identify issues, problems and approaches used in the migration from legacy systems to aspect-oriented software system.

INTRODUCTION

Legacy systems in business industry are very large and complex systems. Evolution of the software systems is inherent due to many causes. A system decomposed into a well modularized system i.e. functions and classes may have some functionality that cut across that modularity. This is often referred to as tyranny of the dominant decomposition (Tarr, Ossher, Harrison, & Sutton Jr, 1999) and such functionalities are called crosscutting concerns because they are spread over many decomposition units. Examples of crosscutting concerns are logging, synchronization, exception handling, persistence, exception handling, and error management. Many crosscutting concerns are spread, either scattered or tangled, all over the code. This leads to the problem of maintenance and understandability of software systems. Identification and modularization of these crosscutting concerns are very difficult. Aspects-oriented techniques can be applied to the legacy systems in the business industry, i.e. there is a need to migrate the legacy codes into the aspect-oriented systems. Aspects represent the non-functional requirements or behaviors of the system. They are the non-functional requirements or –ilities of the system. In order to transform

DOI: 10.4018/978-1-4666-8510-9.ch012

the legacy systems to aspect-oriented systems, there is a need of tools and techniques that can help in identifying the crosscutting concerns in the systems and refactoring them into aspects. Migration of the legacy codes into the aspect-oriented systems is composed of aspect mining and aspect refactoring.

Aspect Mining is a reverse engineering process of identifying the crosscutting concerns in the given source code of the legacy system that can be potentially converted into aspects. Such concerns are referred to as 'aspects candidates'.

Aspect Refactoring is the process of converting the identified aspect candidates into real aspects in the source code.

Due to the large size of the legacy systems, complexity of the code, lack of documentation, and knowledge about the system, need for tools and techniques that can aid software engineers in locating and documenting, discovering and refactoring concerns is realized. Code duplicity in the legacy systems due to the presence of crosscutting concerns scattered and tangled throughout the system can be separated from the base code using the aspect-oriented technology, thus making the system easier to understand, maintain, and evolve.

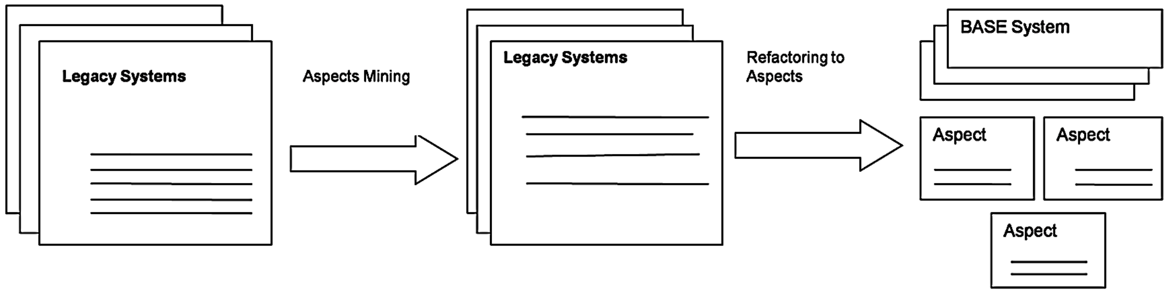
This chapter focuses mainly on issues, drawback, approaches, tools and techniques of aspect mining and does not deal with aspect refactoring part. For over more than a decade, researchers have tried to develop tools and techniques to identify crosscutting concerns in previously developed software system, without using Aspect Oriented Programming (AOP).

In AOP, special classes called "aspects" capture crosscutting concerns. Aspects are defined by aspect declarations, which may include pointcut declarations, advice declarations, as well as declaration of those methods, which are permitted in the class declarations. The aspect is woven to produce the final system, using a special tool called weaver.

The techniques which discover crosscutting concerns, scans for either the symptoms of code duplication, code scattering, or code tangling. Code scattering means the code which implements a crosscutting concern is spread across the system. Code tangling means the code which implements some concern is mixed with code from other crosscutting concerns. The main contribution of this chapter is the presentation of state-of art of the aspect mining techniques.

The chapter is organized as follows: Section 2 presents the background of legacy systems (including concepts, issues, challenges in modernization, and various modernization techniques) and concepts of Aspect-Oriented Software Engineering paradigm. Section 3 discusses various issues involved in the evolution of legacy systems to aspect-oriented systems. In section 4, various problems and causes in aspect mining techniques are identified. Section 5 identifies and discusses various aspect mining approaches and techniques proposed by the researchers.

Figure 1. Migrating a legacy system to an aspect-oriented system


Legacy Systems
Aspects Mining
Legacy Systems
Refactoring to Aspects
BASE System
Aspect
Aspect
Aspect