51780 Pir Khurram Rashdi

34. ASPECT-ORIENTED APPROACH FOR DISCOVERING DESIGN PATTERNS FROM JAVA-BASED CODE

ABSTRACT

The problems of lack of design knowledge and unavailability of up-to-date design knowledge need to be addressed for software development when existing software architecture and design need to be modified. In past various approaches along with tools and techniques have been developed and proposed. However, the complexity, flexibility and accuracy were problem areas in the domain of design knowledge discovery in terms of design patterns instances from object-oriented source code especially Java-based code. The research-undertaken deals with the problem of design knowledge discovery from the Java-based source code using aspect-oriented approach. The aspect-oriented approach proposed has been experimented using tools such as Eclipse SDK and AJDT plug-in for AspectJ to prove that aspect-oriented programming is suitable approach than the previous approaches to avoid complexity, and providing with the required flexibility and accuracy. Initially, three patterns i.e. Singleton, Prototype and Iterator have been discovered from Java-based code.

In the software development lifecycle, when an existing software system is modified or replaced it is often the case that design knowledge is not available or outdated in case of old legacy systems or rapidly evolving systems making it harder for developers to take design decisions. To overcome this bottleneck, a variety of approaches has been proposed in the past for extraction of design knowledge from source codes. Since the design patterns are the abstract concepts representing the design knowledge, the discovery of design patterns from source code can help us in providing with the up-to-date design knowledge. The design extraction or reverse engineering approaches targets properties, which are structural, behavioural or mixed of both for the detection of design patterns. Such attributes are most of the time tightly coupled in source code and multiple design patterns may reside in a single class or may co-exist anywhere, a single

design pattern may have variety of implementations, and patterns are not well implemented; so the accuracy of the approaches is still unsatisfactory due to the limitations of tools and technique.

In the research, we have targeted an aspect-oriented paradigm, which has built-in capability when it comes to tracing the point of interests (joinpoints) in the code. The implementations of design patterns have been chosen as Joinpoints in our case. The structures provided by aspect-oriented programming provide us with all the capability necessary for discovering the design patterns implementation. We define Pointcuts (much like wildcard patterns) to define the joinpoint (i.e. design patterns) to be captured. We use Aspect structure (similar to Class in OOP) to contain Pointcut or Advice like structures. We use before() advise for executing logging functionality before the pointcut we define to capture joinpoints (design patterns).

The aspect-oriented approach presented discovers and counts the design patterns or joinpoints from targeted source code. Firstly, each advice presents a full list of class methods in object-oriented code using cross-references view, which contains potential design patterns. Secondly, the before() advise itself also counts the design patterns implemented in the targeted code. Thirdly, the visualiser view in Eclipse SDK provides with graphical representations with specific colour for each design patterns and allows a user to click the coloured stripe to take him to the exact location where instance of design pattern is discovered. These valuable achievements were possible after aspect-oriented programming.

The open-source JHotDraw 7.1 drawing framework has been chosen as a target object-oriented system. JHotDraw is a Java based framework for structured drawing editors and for document-oriented applications so that the two frameworks can run independently of each other. The drawing editor framework consists of applications, which can also run as an applet. The applications are namely Draw sample application, Net sample application, PERT sample application, SVG sample application and Teddy sample application.

The Eclipse SDK with the support of AspectJ as plug-in is experimented for verification of aspect-oriented approach presented. The source code of JHotDraw was imported along with libraries in Jar format and compiled for removing the dependencies. The Java-based project was converted to an AspectJ project with the support provided by AJDT plug-in for AspectJ in Eclipse SDK. AspectJ runtime library was attached as soon as JHotDraw framework was available as AspectJ project.

The aspects were defined after creating a separate package in converted AspectJ project which comprise pointcut descriptor to capture joinpoints (e.g. design pattern instances), before() advices and attributes such as variables to count the instances of joinpoints. These aspects were weaved with java classes to give combined effect as soon as any of JHotDraw applications was executed in Eclipse SDK.

Findings or results gathered in our approach fall in three broad categories. Firstly, in the Eclipse SDK, the cross-references view displays all class methods signatures, which are captured by pointcut descriptor being advised using before() advise. Hence, the joinpoints so captured are assumed to be potential design patterns instances, which are targeted by corresponding aspect definition.

Secondly, the AspectJ code is written in before() advise for tracing or logging the design patterns instances found while executing the actual source code. A good example of this is when an application or applet in JHotDraw framework is executed using Eclipse SDK, the code in before() advise of say Singleton aspect gets executed whenever a joinpoint (singleton design pattern) fulfils the criteria defined in pointcut descriptor for Singleton pattern.

Thirdly, the Eclipse SDK presents a graphical view named as Visualiser view in which three design patterns chosen in our research namely Singleton, Iterator and Prototype are represented with different colours. Visualiser view constitute of graphical bars, which represent individual object-oriented class. A separate view called Visualiser Menu represents colours for each aspect such as AspectIteratorDP, AspectPrototypeDP and AspectSingletonDP. The vertical bars in Visualiser view comprises of various stripes with the different colours representing the three design patterns that are discovered. The clicking an individual stripe in graphical bar takes a user to corresponding line of code where the design patterns instance is found.