

# Enhancing Software Evolution with Pattern Oriented Software Product Life Cycle

Jayadev Gyani  
University of Hyderabad  
INDIA  
jayadevgyani@yahoo.com

P.R.K. Murti  
University of Hyderabad  
INDIA  
prkmcs@uohyd.ernet.in

## Abstract

*In the current age of software development design patterns have to play a greater role in software development. Design patterns can be defined as the solutions to the frequently recurring problems in design. Pattern based development will improve the software reusability if properly utilized. In this paper, we present pattern oriented software product life cycle (PSPLC) with a focus on Gang-of-Four design patterns. Many developers are resisting the use of design patterns because of unfamiliarity with this area. This approach is very useful to novice developers. Software evolution implies modifying requirements or adding new requirements. PSPLC thrusts the developers in using design patterns, which results in improved software evolution. A case study on real time scheduling is discussed for showing the applicability of PSPLC.*

*Key words: Design patterns, product life cycle, software design reuse, software evolution*

## 1. Introduction

Using the idea of design patterns early in the product life cycle will improve the clarity of the software architecture. If design patterns are suggested before going to the design phase, the new developers can throw light on using design patterns. Even though using a design pattern for a specific set of requirements is a creative task, this approach aids in understanding the use of design patterns. In section 2, we propose Pattern Oriented Product Life Cycle. In section 3, we present few guidelines to suggest design patterns based on the specification.

## 2. Pattern oriented product life cycle

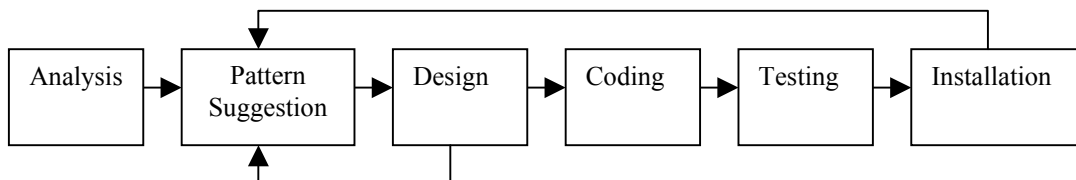
General product life cycle contains the analysis, design, coding, testing and installation phases. We suggest a new phase called **pattern suggestion phase** before design phase. **Figure 1** shows the new pattern-oriented software product life cycle. We call it as PSPLC.

It starts with the object oriented analysis phase encompassing the specification of classes, methods and class relationships. Any of the standard object oriented analysis methods can be used. The specification should include the features for representing association, subclassing, composition and other relationships.

In the next proposed phase, design patterns can be suggested based on the requirement specification. The specification can be analyzed either manually or using a simple parser. The criteria for suggesting patterns will be discussed in section 3. This phase can be called as pattern suggestion phase.

In the design phase, the suggested patterns can be analyzed in the context of the current application. A set of guidelines will motivate the developer to look through these patterns. When the suggested patterns are not suitable to the current application, new design patterns can be designed and added. It is an evolving phase. All the benefits of using design patterns may not be realized for the first time.

When the designs are implemented in one of the languages, the application can be tested. After successful testing and installation, new design patterns can be mined from the current working application. The criteria for suggesting these patterns can be added to the pattern suggestion phase.



**Figure 1. Pattern oriented product life cycle**

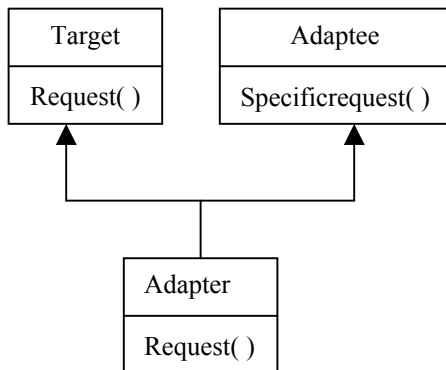
### 3. Criteria for suggesting patterns

Design patterns are solutions to recurring problems in design[1]. Design patterns can be used for creating flexible and maintainable software. Extracting design patterns from design or code is suggested by G. Antoniol et al.[3]. Design patterns are used in developing application frameworks[4]. Pattern oriented software development life cycle model was suggested by M.S.Rajasree et al.[5]. Their approach was based on creating global structure based on communication model. This model focused on transforming requirements to design patterns.

Natural language heuristics can be used as guidelines for suggesting patterns. These guidelines are the minimum evaluation criteria for suggesting a pattern from the specification. If the suggested patterns are not suitable to the current application, the life cycle proceeds in a normal way. Little material is available on suggesting patterns using natural language heuristics. These guidelines are useful when the requirements are specified in natural language. Criteria for suggesting six design patterns will be discussed in this section. All these patterns are taken from Gang-of-Four design patterns.

#### 3.1 Adapter

The structure of the adapter pattern is shown in **Figure 2**.

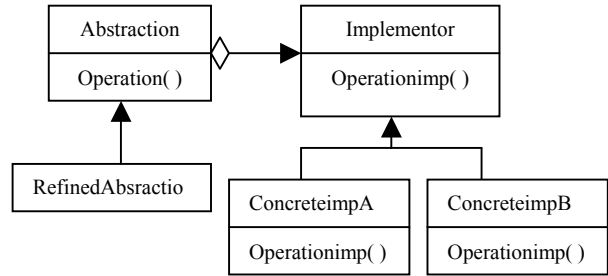


**Figure 2. Adapter**

**Criterion 1:** “Must comply with existing interface” or “Must reuse existing legacy component”.

#### 3.2 Bridge

The structure of the bridge pattern is shown in **Figure 3**.

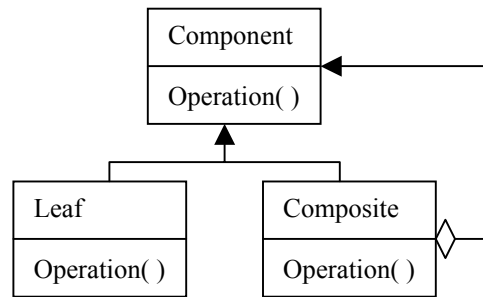


**Figure 3. Bridge**

**Criterion 2:** “Must support future protocols.”

#### 3.3 Composite

The structure of the composite pattern is shown in **Figure 4**.

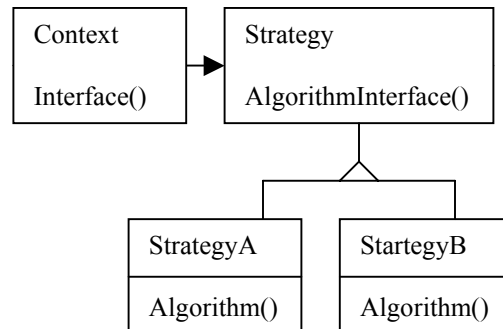


**Figure 4. Composite**

**Criterion 3:** “Must support aggregate structures.”

#### 3.4 Strategy

The structure of the strategy pattern is shown in **Figure 5**.



**Figure 5. Strategy**

**Criterion 4:** "Must implement different algorithms and a common interface to be provided"

### 3.5 Façade

The structure of the facade pattern is shown in Figure 6.

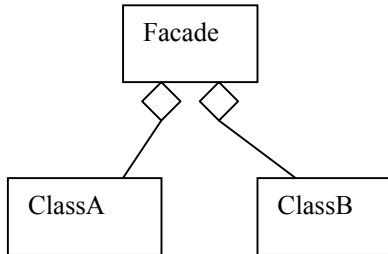


Figure 6. Facade

**Criterion 5:** "Must provide a unified interface for various subsystems."

### 3.6 Proxy

The structure of the proxy pattern is shown in Figure 7.

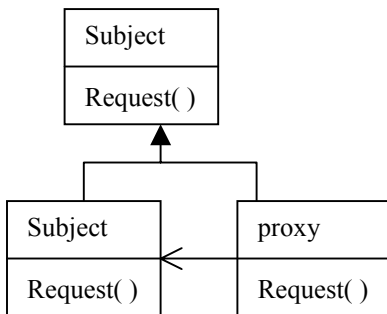


Figure 7. Proxy

**Criterion 6:** "Must support simple object creation when the actual object creation is time consuming."

## 4. Case Study

We explain our concept with Real time scheduling. Requirements Specification for real time scheduling in plain English will be as follows:

1. Real time scheduling algorithms for periodic task requires various parameters such as phase, period, execution time and relative deadline.
2. Application should implement different algorithms with common interface. The user interface takes the parameters specified in the above requirement.
3. Rate-monotonic(RM) and Deadline-monotonic (DM) algorithms have to be implemented.

4. Other scheduling algorithms need to be added later.

5. Output shows the schedule diagram of selected algorithm.

Above Requirements seem to be simple but they are sufficient to explain our concept. According natural language heuristics, *Strategy* pattern can be used. When multiple patterns satisfy a heuristic, then different solutions can be suggested. Identifying a pattern before the design phase will help the developer in framing the solution effectively. This will be done during *pattern suggestion* phase. PSPLC emphasizes the use of design patterns, which results in enhanced software evolution. In the given example if any other algorithm is to be implemented such as Earliest-Deadline-First algorithm, then this can be done adding another concrete strategy class. Strategy pattern for real time scheduling can be shown in Figure 8.

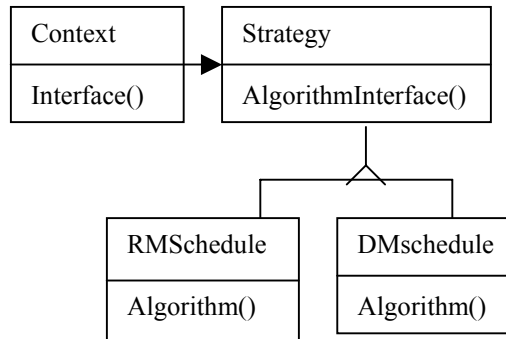


Figure 8. Strategy for Real Time Scheduling

After developing the application the user would like to implement one more algorithm EDF that can be simply added to the Strategy without affecting other modules. The resulting figure is shown in Figure 9.

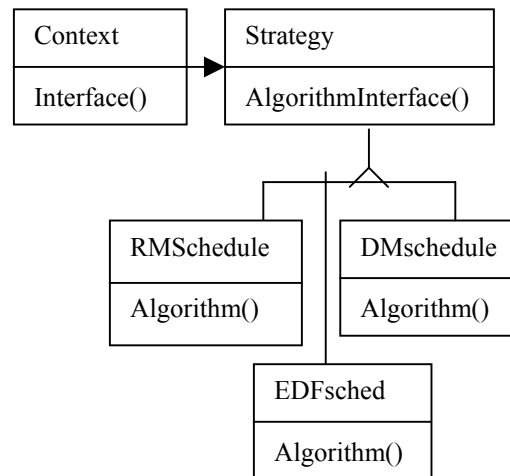
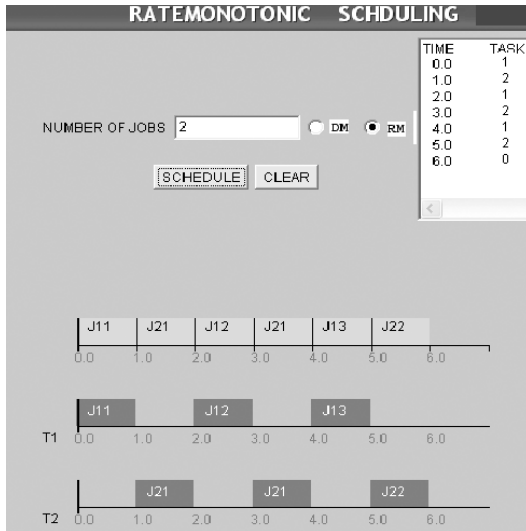


Figure 9. Strategy after adding EDF

After installing the application, new patterns can be mined such as user interaction pattern. We suggest one pattern for user interaction for real time scheduling. We name it as “RealSchedule”. We have implemented this user interface in java. Whenever real time scheduling is required, this pattern can be used. A snapshot of this pattern is shown in **Figure 10**. Same user interface can be used for DM schedule also.

1997, Communication of the ACM, Volume 40, Number 10, pp. 32-38

[5] M. S. Rajasree, P. Jithendra Kumar Reddy, D. Janaki Ram, “Pattern Oriented Software Development: Moving Seamlessly from Requirements to Architecture”, International Workshop on Software Requirements to Architectures (STRAW 03) in association with International Conference on Software Engineering (ICSE 03) MAY 2003 Portland, Oregon, USA



**Figure 10. RM schedule**

## 5. Conclusions and future work

Pattern-oriented product life cycle focuses on design reuse. While suggesting patterns, we focussed on the few patterns of GOF pattern catalogue because of simplicity in understanding the behavior of these patterns. However, same approach can be extended to include other patterns of GOF pattern catalogue and POSA patterns.

## 6. References

- [1]. Erich Gamma and others, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison Wesley, 1995
- [2] Tiffany Winn, Paul Calder, “Is This a Pattern?”, IEEE Software, January 2002, pp. 59-66
- [3] G. Antoniol, R. Fiutem, L. Cristoforetti, “Using Metrics to Identify Design Patterns in Object-Oriented Software”, 5th. International Symposium on Software Metrics, March 1998, pp. 23
- [4] Mohamed Fayad and Douglas C. Schmidt, “Object-Oriented Application Frameworks”,