Chapter 2
Aspect Adapted Sculpting

Rachit Mohan Garg
IT Engineer, India

ABSTRACT

This chapter enables the learners to envisage the prevailing glitches of concern in programming and enables them to identify the concerns that may be incurred in the development of the software. How the designing phase of the product is molded separates these concerns from the core business logic, thereby providing a product with an ability to incorporate any change without actually touching the core business logic unless necessary. Moreover, creating a good design leads to less rework in the latter stages of the SDLC (Software Development Life Cycle).

INTRODUCTION

Success at developing simpler systems leads to aspirations for greater complexity. Managing complex software system is one of the most important problems for software engineering to handle. Increase in the complexity of the software requirements and the initiation of innovative technology makes it indispensable for software packages to integrate and accommodate the countless assortments of special concerns which primarily include concurrency, distribution, real-time constraints, location control, persistence, and failure recovery. Hiding among these special purpose concerns is the core concern responsible for the fundamental business logic. These non-functional concerns help in either fulfilling special requests of the application (real-time, persistence, distribution), or to manage and to optimize the core business logic (location control, concurrency).

Even though the concerns may be separated conceptually and incorporated correctly, interpersing them in the code brings about a number of problems:

- Interpreting and maintaining a tangled code is rigid and multifaceted as all the concerns have to be maintained at the same time and at the same level due to very strong coupling among them.

DOI: 10.4018/978-1-4666-4494-6.ch002
Tangled code is hard to understand because of the lack of abstraction as depicted in the above point.

Also in systems designed using OOPs concepts, tangled code gives rise to inheritance incongruities as it becomes impossible to re-factorize a method or a concern without re-factorizing both, which again is the byproduct of strong coupling of different concerns.

Modeling is considered to be one of the most successful elements in the progression of computer systems expansion. It is the process of representing real-world concepts in the computer domains as a blueprint for the purpose of software development. Recent trends in software development have also revealed the value of developing systems at higher levels of abstraction.

The significance of modeling and abstraction becomes further strengthened in the process of Model Driven Software Engineering (MDSE) (Bran Selic, 2003), where they are considered the heart and soul of software development.

The core objective of this chapter is to enable the learners to visualize the above mentioned glitches in the vastly followed OOP practice. A better understanding on the concept of concerns will allow the learners to separate out the concerns from the core logic while developing a software product in the initial phase itself so that the glitches are not transferred to the latter phases which in turn will lead to less rework in the latter stages of the SDLC.

BACKGROUND

Aspects Case

Jacobson (2003) describes how the design aspects can be developed using use cases. It uses the concept of composition techniques that maps the work directly into the program level aspects. The proposed work does not provide details about models transformations, composition rules, structural relations, etc.

Using Meta-Models and UML Profiles for Separating Out Concerns (Garg, 2011)

UML is the most popular standard used for describing systems in terms of object concepts and mostly used in software design. UML allows for extensions to augment the meta-model in two ways:

- Direct extension of the UML meta-model through MOF elements and
- Extension of the meta-model through UML Profiles.

The first (also called ‘heavy-weight’) approach is more flexible, since it allows generous modification of the UML meta-model. Nevertheless, one is generally discouraged to use it, since the resulting UML meta-model does not conform to the OMG standard breaking, among other things, interoperability between tools.

The latter (‘light-weight’) approach uses Profiles, a built-in extension mechanism, to adapt UML to the desired domain. According to Kozak Ayse (2002), UML is not defined as a modeling language but rather as a language family, where Profiles are the individual family members, that is, and the actual languages. Nevertheless, neither approach allows for specific parts of the meta-model to be hidden, if they are not required for the specific purpose. If one wants just a glass of milk, one has to take the whole cow.

The UML Profile mechanism in particular is used heavily in Aspect Oriented Modeling (AOM) to introduce platform specific annotations as well as to define platform-independent language constructs. In AOM based MDSE, UML profile plays a very important role in expressing the PIM models, the PSM models, and the transformation
Related Content

Capturing Location in Process Models: Comparing Small Adaptations of Mainstream Notation
[www.igi-global.com/article/capturing-location-process-models/67579?camid=4v1a](www.igi-global.com/article/capturing-location-process-models/67579?camid=4v1a)

Agile Method Fragments and Construction Validation
[www.igi-global.com/chapter/agile-method-fragments-construction-validation/21074?camid=4v1a](www.igi-global.com/chapter/agile-method-fragments-construction-validation/21074?camid=4v1a)

Intentional Process Mining: Discovering and Modeling the Goals Behind Processes using Supervised Learning
[www.igi-global.com/article/intentional-process-mining/120172?camid=4v1a](www.igi-global.com/article/intentional-process-mining/120172?camid=4v1a)

Transitioning from Code-Centric to Model-Driven Industrial Projects: Empirical Studies in Industry and Academia
[www.igi-global.com/chapter/transitioning-code-centric-model-driven/26832?camid=4v1a](www.igi-global.com/chapter/transitioning-code-centric-model-driven/26832?camid=4v1a)